SCALABLE NEURO-SYMBOLIC MACHINE LEARNING

Blaž Škrlj

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Nada Lavrač, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Prof. Sašo Džeroski, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Kristian Kersting, Member, TU Darmstadt, Darmstadt, Germany
Prof. Marko Robnik-Šikonja, Member, Faculty of Computer and Information Science, Ljubljana, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Blaž Škrlj

SCALABLE NEURO-SYMBOLIC MACHINE LEARNING

Doctoral Dissertation

UČINKOVITO NEVRO-SIMBOLNO STROJNO UČENJE

Doktorska disertacija

Supervisor: Prof. Dr. Nada Lavrač

Ljubljana, Slovenia, January 2022

To Eva and my family

Acknowledgements

This thesis contains the results of multiple fruitful collaborations over the past few years. I would first like to thank my supervisor Prof. Nada Lavrač for countless interesting discussions and opened doors, support when starting new collaborations/projects and critical overview when necessary. On the technical side of things, I am grateful to Jan Kralj for his guidance when attempting to formalize some of the ideas, patience when correcting the constructed formalisms and oversight when occasionally implementing them. Finally, at this point, I would also like to thank Matej Martinc for helping me build competitive text-related approaches and introducing me to the world of shared tasks and competitive machine learning.

Sharing ideas with peers is one of the most critical aspects of research. I want to thank Matej Petković, Tomaž Stepišnik, Jure Brence, Martin Breskvar and Nika Eržen for being patient enough to listen to large amounts of random thought streams, which now and then appear in the main contributions of this thesis. Furthermore, I am confident that I have grown as a programmer due to joint coding endeavours with many of them, especially Matej. Implementing an idea is one thing, but implementing it robustly and stably can be a completely different endeavour. Finally, I have to thank Matic Perovšek, Anže Vavpetič, Barbara Krašovec and Janez Kranjc for fruitful discussions and feedback regarding how one should implement modern software solutions and what are good design practices. This aspect is not to be underrated in the era where implementations are the bedrock of the main breakthroughs.

During the last couple of years, I was honoured to pass some of the developed skills to the younger generation, from which I learned just as much. At this point, I would like to thank Boshko Koloski, Sebastian Mežnar, Timen Stepišnik Perdih, Ilija Tavchioski, Matej Bevec and possibly others for offering me an insight into how one can efficiently supervise/mentor and simultaneously build.

While learning a lot from my peers, I was lucky enough to have been in contact with more senior researchers from whom I learned how to think at a higher level, more critically, and plan for the long term. Here, I would first emphasize the collaboration with Prof. Marko Robnik Šikonja and Dr. Senja Pollak, who, apart from many exciting ideas, many times gave me the motivation to push a project across the final line and get things done. Furthermore, one of the first publications, which is also included in this thesis, was primarily due to an opportunity offered to me by Prof. Sašo Džeroski; jointly collaborating with him and my supervisor was a precious experience from which I learned a lot about how to write better; I slowly realize what a vital skill this is. My encounters with the department's theoretician France Dacar were also at times eye-opening; by seeing how a well-versed mathematician thinks and explores different abstractions, I gained a lot while simultaneously realizing how much more there is to learn.

Finally, I would like to thank my family for the support, exciting discussions, guidance and motivation, Link for reminding me that 3 am can be very peaceful, and Eva for being there when I needed you the most.

Abstract

With the resurgence of neural network-based learning in the last decade, machine learning methods are becoming critical components of many real-life intelligent systems. However, while being able to learn effectively and at scale, such systems are often non-interpretable and unable to exploit existing symbolic background knowledge. The paradigm that offers such endeavour is symbolic learning, which has been investigated for more than 50 years.

The main focus of this thesis is the recent paradigm of *neuro-symbolic* machine learning. This branch of learning investigates whether combining the techniques from neural (sub-symbolic) and symbolic machine learning can be used to develop better performing and more explainable predictive models. The notion of neuro-symbolic machine learning transcends individual input data types and can be considered when learning from tables, graphs, and text-based data. This thesis aims to investigate when, and to what extent, can the neuro-symbolic paradigm prove beneficial when learning network node representations, classifying texts and ranking features. Furthermore, we investigate different aspects of neuro-symbolic models: from predictive performance to scalability. The contributions of this thesis address different input data types. We first present the results of applying neuro-symbolic learning to relational learning, considering two different relational learning scenarios: learning from relational databases and network node embedding. We demonstrate that by using the neuro-symbolic paradigm, improved scalability of propositionalization-based approaches is achieved. Further, by considering neuro-symbolic node representation learning, we demonstrate competitive predictive capabilities, while, by systematic (symbolic) node pruning, reaching better scalability.

Next, we present autoBOT, a neuro-symbolic autoML system for automating *text classification*. By simultaneously considering both symbolic and sub-symbolic document representations, evolution-based optimization yields well-performing models that remain interpretable at two different granularities: at the level of feature types (i.e., which feature type is more relevant) and also at the level of individual features. To our knowledge, autoBOT is one of the first neuro-symbolic autoML systems aimed at optimizing both performance and explainability. A novel, previously unpublished contribution is also a computational framework, enabling us to scale autoBOT to a supercomputing grid, allowing two to three orders of magnitude more experiments to be conducted on a single machine at the same time.

The final part of the thesis focuses on of *feature ranking*. This task addresses the issue of identifying importance scores for features, which correspond to their capacity to separate parts of the target space – highly ranked features are commonly the ones that impact the learning process the most. We demonstrate that the neuro-symbolic paradigm helps better understand the relationship between neural attention and ranking and offers better scaling of existing feature ranking approaches, such as the Relief-based feature ranking approaches. The thesis concludes with an evaluation of the developed approaches, lessons learned and guidelines for potentially interesting future work.

Povzetek

Obujeno področje globokega učenja je v zadnjem desetletju ponudilo pristope, ki postajajo ključni sestavni deli sodobnih inteligentnih sistemov. Poleg učinkovitosti je za te metode značilna tudi njihova sposobnost procesiranja večjih količin podatkov (skaliranje). Glavni pomanjkljivosti globokega učenja sta neinterpretabilnost ter nezmožnost direktne izrabe uporabnega simbolnega predznanja. Veja strojnega učenja, ki uspešno naslavlja te probleme, je simbolno učenje, ki je v računalništvu prisotno že vsaj 50 let.

Glavna tematika te disertacije je novejše področje t. i. nevro-simbolnega strojnega učenja. Ta veja učenja raziskuje, ali s kombiniranjem oz. izrabo idej tako s področja nevronskega (podsimbolnega) učenja ter simbolnega učenja lahko obstoječe metode nadgradimo z ozirom na njihovo interpretabilnost, skaliranje ter napovedne sposobnosti. Področje nevro-simbolnega učenja je uporabno za učenje iz različnih tipov vhodnih podatkov, kot so npr. tabele, grafi ali tekstovni podatki. V disertaciji smo raziskali, kdaj ter do kakšne mere lahko paradigma nevro-simbolnega učenja pomaga pri učenju reprezentacij vozlišč v omrežjih, klasifikaciji tekstov ter rangiranju značilk. Raziskali smo različne lastnosti nevro-simbolnih modelov: od njihovega učinka na npr. klasifikacijsko točnost do njihovega skaliranja na računalniških gručah. Glavni prispevki, predstavljeni v tem delu, se osredotočajo na različne tipe vhodnih podatkov. Začnemo z razvitimi metodami za nevro-simbolno učenje iz relacijskih podatkov, kjer smo raziskali dve nalogi učenja: učenje iz relacijskih baz ter vložitve vozlišč v omrežjih. Rezultati kažejo, da lahko z uporabno nevro-simbolne paradigme učenja dosežemo boljše skaliranje metod propozicionalizacije, ter pridobimo kvalitetne vložitve vozlišč v omrežjih, ki omogočajo boljše skaliranje ter podobno klasifikacijsko točnost kot obstoječe metode v širši rabi.

V nadaljevanju predstavimo orodje autoBOT, razvito z namenom avtomatizacije naloge klasifikacije tekstov. V disertaciji pokažemo, da je s hkratno izrabo simbolnih ter podsimbolnih reprezentacij dokumentov, dodatno obteženih s pomočjo evolucijske optimizacije, mogoče pridobiti modele, zmožne dobre klasifikacijske točnosti, ki so hkrati interpretabilni na dveh nivojih: na nivoju tipov značilk kot tudi posameznih značilk. Gre za enega prvih nevro-simbolnih sistemov, razvitega za avtomatizacijo učenja iz tekstov. Poleg razvite metode v disertaciji predstavimo tudi še neobjavljeno računsko ogrodje, ki nam je omogočilo skaliranje evolucije na več različnih računalniških gruč, sestoječih iz stotin računskih vozlišč z različnimi specifikacijami. Ta nivo skaliranja nam je omogočil izvedbo enega do dveh velikostnih razredov več eksperimentov.

V zadnjem delu disertacije se osredotočimo na nalogo rangiranja značilk. Tu kot rezultat pričakujemo izračunane pomembnosti za vsako značilko. Značilke z večjimi pomembnostmi imajo pričakovano večji vpliv na sposobnost razločevanja elementov izhodnega prostora. V disertaciji prikažemo uporabnost nevro-simbolne paradigme za boljše razumevanje odnosa med nevronsko pozornostjo ter rangiranjem značilk kot tudi kako lahko z vložitvami vhodnega prostora pospešimo proces rangiranja z obstoječimi algoritmi, kot so npr. algoritmi iz družine Relief. Na koncu disertacije kritično ovrednotimo razvite pristope, njihove pomanjkljivosti ter potencialno zanimive razširitve.

Contents

Chapter 1

Introduction

> In order to make an apple pie from scratch,
> you must first create the universe.
>
> Carl Sagan

This chapter places the research of this thesis into the broader context of machine learning and representation learning. It introduces the two main paradigms of machine learning, the sub-symbolic and symbolic learning, their key scaling-related properties, and the open problems addressed in the thesis. The chapter concludes with an overview of the thesis and its scientific contributions of the thesis and an outline of its structure.

1.1 Background and Motivation

Machine learning has grown into one of the dominant fields of computer science and has found applications in biomedicine, the news media industry, engineering, physics and many other branches of science and industry (Dargan et al., 2020). With its origins in the previous century (1950s, logic theorem provers) (Russell & Norvig, 2002), the effort to mimic intelligent behavior focused predominantly on *symbolic learning* (Kolata, 1982). The field now known as *symbolic machine learning* has been at the time known also as *symbolic artificial intelligence*. The branch of symbolic machine learning methods has been popularized with algorithms such as decision tree learning (Quinlan, 1986), inductive logic programming (Muggleton, 1991) or learning predictive rules (Holte, 1993), and remains widely utilized in real-world scenarios where each decision needs to be traceable. For example, if we consider the following decision rule

$$\text{class(infected)} \leftarrow \text{hasFever}(\mathit{true}) \wedge \text{hasHeadache}(\mathit{true}) \wedge \text{hasPain}(\mathit{true}),$$

a domain expert can immediately inspect the collection of, in this case, binary features, identified by the learner as crucial for deducing a given class. This branch of methods is thus, in most cases, *explainable* – individual, human-understandable facts that constitute a decision rule can be inspected directly. One of the main caveats of symbolic methods is their language bias (i.e., how the decision space is split or the underlying representations constructed), which potentially constrains the space of possible descriptions and can result in performance loss. Symbolic methods, unless considered as parts of bagging (Breiman, 2001) or boosting-based ensemble models (Chen & Guestrin, 2016), commonly underperform on mainstream learning tasks, including multi-class learning and regression (Sagi & Rokach, 2018). Furthermore, this branch of methods does not necessarily learn incrementally, rendering their deployment on larger data sets problematic, where specialized,

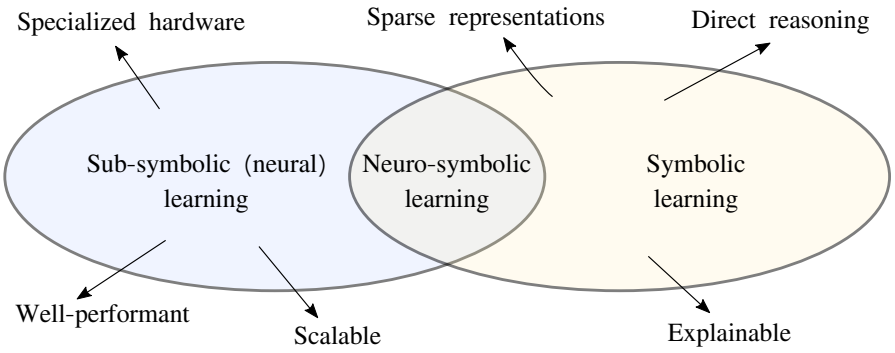


Figure 1.1: Overview of the three learning paradigms considered.

either algorithmic or hardware-level adaptations may be required when scaling to larger, realistic data sets (Gama et al., 2013; Osojnik et al., 2017).

Compared to symbolic methods, *sub-symbolic learners* have in recent years shown great promise in terms of scalability and overall performance for tasks from the fields of computer vision, language processing and relational learning (LeCun et al., 2015; Pouyanfar et al., 2018). The main contemporary sub-symbolic methods are mostly based on *deep neural networks* – multi-layered collections of real-valued weights with intermediary non-linear activations, trained via backpropagation (Kelley, 1960; Linnainmaa, 1976; Rumelhart et al., 1985) (first considered already around 50 years ago). Note that we use the terms neural/sub-symbolic interchangeably throughout this work. With the resurgence of deeper neural networks around 2009 (Deng et al., 2009; LeCun et al., 2015), systems capable of protein folding (Jumper et al., 2021), language generation (Brown et al., 2020) and translation (Vaswani et al., 2017), fraud detection (Roy et al., 2018), traffic control (Hatolkar et al., 2018) and similar have emerged, and are already being actively used throughout industry and academia (Le et al., 2020). Despite being powerful *function approximation engines*, neural networks have to this date not been able to fully conquer the types of tasks where explicit symbolic knowledge of relations and concepts is necessary, requiring abstract reasoning (Chollet, 2019; Nazarczuk & Mikolajczyk, 2020).

Exploring which aspects of neural-only learning prohibit a given system from solving such harder reasoning tasks has led research into the direction of *neuro-symbolic learning* (Garcez & Lamb, 2020; Lamb et al., 2020; Raedt et al., 2020), which is the key focus of this thesis. The neuro-symbolic machine learning paradigm emerged in the last two decades with the aim to explore to what extent can, e.g., existing, logic-based systems be improved when coupled with differentiable neural network-based components (Sarker et al., 2021). Furthermore, neuro-symbolic approaches to machine learning retain some degree of explainability, which can be a critical factor in high-risk scenarios, e.g., clinical diagnosis (Pisano et al., 2020).

Having discussed the main learning paradigms considered in this thesis, we next discuss the main *motivations* underlying the presented research. The symbolic learning paradigm has offered considerable improvement in understanding the main patterns underlying a given data set. As such, it served as an invaluable tool when trying to better understand the underlying process or to present the induced models to domain experts. One of the main caveats commonly observed when constructing symbolic models is sub-optimal predictive performance. A possible solution to this problem are ensemble-based methods, which, however, considerably reduce the interpretability of the final model. Another possibility, which was the key focus of this work, involves symbolic *feature construction*. Here,

human-understandable patterns appearing in the input data are used to construct the feature space, useful for learning. If the machine learning model learned from such space is symbolic, the model remains interpretable. Our motivation was to explore whether the obtained symbolic feature spaces can be efficiently combined with sub-symbolic ones, potentially offering better predictive performance while maintaining (at least partial) interpretability. As the constructed spaces are sparse, this thesis additionally addresses the problem of *implementing representations*, which is just as relevant for successful and efficient learning. To our knowledge, exploring how symbolic and sub-symbolic representation learning intertwines for different types of inputs (and hence tasks) has not been extensively studied.

Contemporary neural network models can learn from lightly pre-processed data (e.g., sensor data), making them very useful for many tasks that would otherwise require extensive feature engineering. Design of such *end-to-end* systems has its benefits; however, it does not necessarily offer insights into what was actually learned and whether the learned representations are noise resistant. In this thesis, we additionally explored the aspect of better understanding how and when a particular representation type is dominant and why this is the case. For example, when representing texts, as multiple different representations of the same document are possible when solving a given classification task, better understanding of how different representations interact and jointly enable a good solution is a relevant research endeavour. The paradigm of neuro-symbolic machine learning systematically investigates such scenarios, and was thus in the background of many of the ideas presented in this thesis.

Contemporary algorithms useful in practical scenarios must, apart from performing well, also *scale*. This aspect of learning was one of the key factors during the resurgence of neural network-based learning in the last decade. Interestingly, scaling symbolic approaches is not necessarily straightforward and needs to be addressed as a separate research question. In particular, we were motivated to explore whether existing propositionalization approaches (Lavrač et al., 2020) scale to multi-million instance data sets and which algorithmic advancements are needed to achieve this. Implementing efficient methods that offer both acceptable predictive performance and adequate explainability for a given use case is an exciting research endeavour, as these (multiple) objectives are not necessarily aligned. Given that parts of a learning process can be natively parallelized, a sensible research direction includes exploration of *grid-level* scaling, i.e. training a given neuro-symbolic approach not on one but on multiple machines. We were particularly interested in profiling the performance of evolution-based (meta)learning, its implementation and other *scaling laws*.

One of the key novelties introduced with the recent sub-symbolic methods is their capability to efficiently *learn* low-dimensional representations. Thus, instead of trying to replace existing algorithms with specifically adapted neural network-based methods, a promising research direction involves investigation of how the existing symbolic methods can *benefit* from learned representations of, e.g., instances. We identified the domain of *feature ranking* as a promising *testbed* for such experimentation. We were motivated to explore whether lower-dimensional representations of very high dimensional data sets can be directly incorporated into one of the learning paradigms for computing feature importances.

Even though the neuro-symbolic paradigm is a relatively young research endeavour (last decade), multiple approaches have already demonstrated that functionality, not achievable by using only sub-symbolic or symbolic methods, is possible. However, understanding the trade-offs regarding different input data types and the addressed tasks remains an interesting research endeavour. In this thesis, we aimed to push the boundary of our

current understanding of the interplay of the two paradigms on types of data sets seldom considered in the context of neuro-symbolic machine learning.

With the increasing amounts of available software solutions to solve different learning tasks, a newcomer to a given field can struggle and spend substantial amounts of time identifying which approach to representation learning and subsequent, e.g., down-stream classification should be chosen. To remedy this issue, the area of autoML systems has evolved in the last decade alongside the other mentioned paradigms (Feurer et al., 2019). The purpose of autoML approaches is to *simplify* learning when solving of a given task as much as possible. In this thesis, we were motivated to push this boundary further to the level of representation learning. This type of autoML system represents the new wave of approaches that do not assume the data to be in a learning-ready format but are capable of *deriving* appropriate representations on their own.

1.2 Purpose of the Dissertation

The neuro-symbolic paradigm has gained considerable traction in recent years. The main reasons for the increased interest in this branch of methods are the need for increased explainability of a given (in most cases neural) method, the need for inclusion of explicit background knowledge into a given learning process, logic-based constraining of sub-symbolic representations, the need for an explicit reasoning step when obtaining the final model output, or the need to improve the scalability of a given symbolic learner.

The purpose of this dissertation is to investigate when, and to what extent, can the neuro-symbolic paradigm offer either performance, scalability or explainability improvements over the existing neural- or symbolic-only methods, and to demonstrate the advances in a range of real-life problems.

The thesis also explores whether learnable zeroth-order relational features can serve as a competitive but more scalable alternative to the existing propositionalization-only learning schemes. This is achieved by an extensive overview of existing symbolic representation learning methods, which are compared to neural representation learners with respect to multiple criteria, ranging from the number of parameters, algorithm type, hardware constraints etc. Furthermore, the thesis' purpose is to explore the implications of adopting the neuro-symbolic paradigm for the task of scalable node embedding (for node classification). To our knowledge, this endeavour is one of the first systematic attempts to better understand the relationship between symbolic and sub-symbolic node representation learning, their algorithmic trade-offs and comparison to the existing state-of-the-art approaches for structural representation learning.

Next, the thesis also explores whether the neuro-symbolic learning paradigm offers superior performance for the task of text classification whilst maintaining parts of the learned representations in symbolic form. By being able to learn directly from such "hybrid" representations, a given classifier – that is able to provide feature importances as part of its training – offers direct insight into whether representation *types* are problem-dependent and whether particular representations can be automatically *re-weighted* to obtain better performance whilst maintaining at least partial explainability. Apart from considering learning from neuro-symbolic representations, the purpose of this last investigation is also to address the issue of hyperparameter optimization by adopting the autoML paradigm for text-based learning, which has, to our knowledge, not been considered at such scale before.

The final addressed problem is the computation of feature importances when considering tabular data sets. This task is of relevance when the end-user is interested in gaining valuable insights regarding which parts of the feature space are most relevant for solving

a given, e.g., classification task. In many real-life biological data sets, pinpointing only a handful of features out of tens of thousands of possible candidates can already provide valuable insights. The thesis explores two ideas which address the feature importance computation problems, solvable by adopting the neuro-symbolic paradigm. The first problem addresses the issue of maintaining the link between a given set of neural network weights and the individual attributes, offering the final (converged) weights as potential feature importances. As part of this thesis, we explore whether the recently introduced idea of neural attention layers (Vaswani et al., 2017) can be useful when designing such architecture for tabular data. Furthermore, we are interested in whether embedding-based instance representations can be useful to speed up the Relief branch of algorithms (Kira & Rendell, 1992) by considering distance computation exclusively in the derived, low-dimensional latent space. The purpose of both endeavours is to demonstrate how the neuro-symbolic paradigm can be applied in both ways: either to improve the existing symbolic method (Relief-based ranking in this case), or to address the issue of maintaining a bijection between the attributes and a part of a given neural network, offering attribute scoring similar to conventional feature ranking.

The thesis addresses the above-mentioned problems either at the level of representation learning or at the level of the whole system's direct learning capabilities. Overall, the purpose of this dissertation is not to promote/emphasize the relevance of neuro-symbolic machine learning but to assess its realistic compatibility with modern software/hardware solutions.

1.3 Hypotheses and Main Goals

This section gives an overview of the main hypotheses and goals of this dissertation. The central hypothesis of this work addresses the question of whether the neuro-symbolic learning paradigm can offer a more explainable, scalable and performant branch of methods for learning from tables, texts and relational data. More specifically, we hypothesize that neuro-symbolic machine learning can handle larger real-world problems and achieve advances in terms of

- (H1) improved *understanding* of the relationship between symbolic and sub-symbolic representation learning with the aim of developing algorithms capable of exploiting the suitable aspects of both paradigms,
- (H2) improved *explainability* compared to neural-only approaches,
- (H3) improved *predictive performance* compared to symbolic-only approaches, and
- (H4) improved *scalability* compared to symbolic-only approaches.

The main goal of this dissertation was to investigate machine learning ideas applicable in real life and whether they can benefit by adopting the neuro-symbolic learning paradigm. The dissertation intentionally focuses on different types of input data (graphs, relational databases, texts and tables) with the aim to demonstrate the general applicability of the neuro-symbolic paradigm.

In terms of *explainability*, we were interested whether increased explainability could be achieved for the tasks of representation learning for text classification and neural network-based learning from tabular data. First, our goal was to build an autoML system capable of automatic document representation prioritization, pinpointing what type of document representation is suitable for solving a given text classification task. By offering feature importances for individual representations, we aim to enable the user to directly inspect

the key features that play the main role (within a given representation). Second, we were interested in the link between the neural attention mechanism and feature importance estimation. We hypothesized that neural self-attention, the process which maintains a bijection between the attributes and a particular set of weights, resembles the process of feature importance computation. This hypothesis was addressed as the autoBOT approach (Škrlj, Martinc, et al., 2021) and self-attention networks for tabular data (Škrlj, Džeroski, et al., 2020).

In terms of *predictive performance*, we were interested whether neuro-symbolic learning algorithms could offer better classification performance when considering relational data and text-based data while being at least as scalable as their symbolic counterparts. Here, we hypothesized that there exist two main approaches to learning low-dimensional representations of either bags of logical statements or individual statements and that both approaches perform competitively to the state-of-the-art propositionalization approaches. Furthermore, we hypothesized that neural network-based distillation of propositional, PageRank-based node representations offers state-of-the-art performance. The contributions which address this hypothesis are the Propositionalization and Embeddings paper (Lavrač et al., 2020) and the Deep Node Ranking paper (Škrlj et al., 2021). Further, we hypothesized that the neuro-symbolic paradigm offers text classification performance superior to the existing symbolic baselines, however performs similarly or worse to the existing *foundation models* (Bommasani et al., 2021), which, however, contain orders of magnitude more parameters (and are thus consistently larger) (Škrlj, Martinc, et al., 2021).

Finally, in terms of *scalability*, we hypothesized that embedding-based feature ranking can offer better scaling to large, high-dimensional data sets and that neuro-symbolic node representation learning can offer a more scalable solution compared to adopted structural representation learners. Further, we hypothesized that the neuro-symbolic paradigm could offer more seamless scaling to grid infrastructure, comprised of different computing nodes, as it can be more easily containerized as it is not necessarily constrained by specialized hardware. The contribution related to this hypothesis concerns the construction of an autoML framework which, with minimal configuration, offers seamless scaling to larger computing grids, analysis of the resulting collections of result/optimization runs and selection of the best final models. The two contributions related to the discussed problems are the ReliefE paper (Škrlj, Džeroski, et al., 2021) and the autoBOT paper (Škrlj, Martinc, et al., 2021). Furthermore, this thesis includes a chapter that was previously not published, discussing a grid-scale solution for training autoML systems applied to autoBOT. A schematic overview of the hypotheses and their relation to the symbolic and sub-symbolic learning paradigms is shown in Figure 1.2.

1.4 Scientific Contributions

We next present the key scientific contributions which constitute this thesis. The contributions are split into two main parts: the *core* contributions, which directly address one of the mentioned hypotheses/problems related to scalable neuro-symbolic learning, and *other* (partially relevant) contributions, which address specific (mostly scalability) aspects of existing state-of-the-art methods, related to the presented work, and are (co-)authored by the author of this thesis, but not focusing exclusively on the neuro-symbolic paradigm which is the key focus of this thesis.

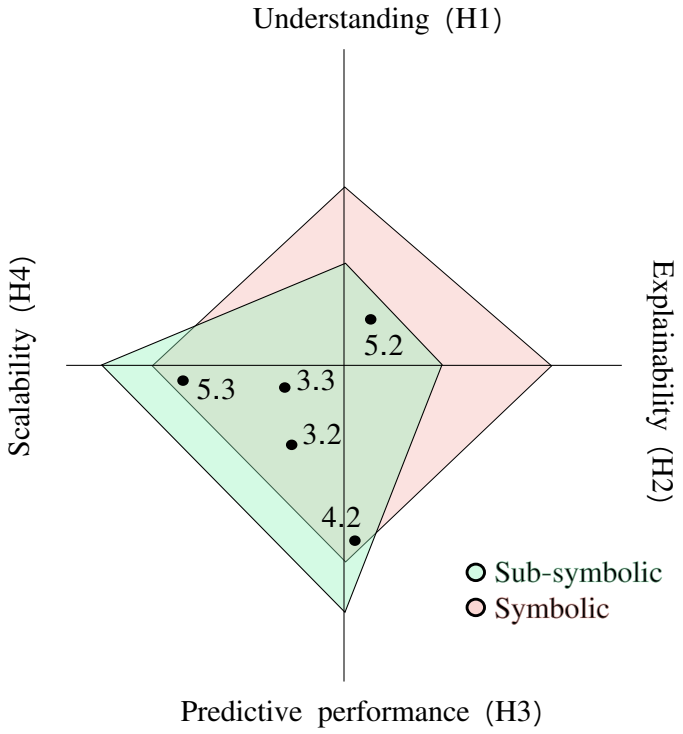


Figure 1.2: Schematic overview of the relations between the learning paradigms with the placement of the core thesis contributions, presented by the labeled dots corresponding to the sections of the thesis where these contributions are described. Note that 'Understanding' here corresponds to a paradigm's potential to better understand the addressed task itself (trade-offs concerning symbolic or sub-symbolic learning).

1.4.1 Core Contributions

We first present the *core* contributions of this thesis, following the common thread of studying aspects of neuro-symbolic machine learning.

Propositionalization and Embeddings (see Section 3.2)

Many real-life data sets contain relations, which are neglected by many standard learning algorithms. This contribution targets the relational domain, i.e., scenarios where a relational database is used as input, one of its tables is the *target* table, and the relational learner must be able to exploit all additional information linked to the target table via foreign keys. The contribution focuses on representation learning, the process of deriving a learning-ready representation from the original representation (e.g., an SQL file) that is not directly suitable as input for learning. The contribution first discusses the relation between symbolic and sub-symbolic representation learning for the relational domain and offers an extensive comparison of the two paradigms with respect to both their algorithmic properties and hardware-level constraints. The contribution also discusses the two main approaches to embedding-based learning when considering a relational database; either the approach embeds the sets of logical clauses which describe individual rows in the target table, or it embeds, in a self-supervised manner, individual clauses, making the process of classification a matter of comparison of the embeddings of such clauses in a learned la-

tent space. The contribution demonstrates that both paradigms scale better than existing state-of-the-art approaches.

Related journal paper

Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, *109*(7), 1465–1507. https://doi.org/10.1007/s10994-020-05890-8

Deep Node Ranking (see Section 3.3)

This contribution also revolves around learning from relational data. However, it focuses on simpler, homogeneous networks, i.e., directed/undirected weighted graphs $G = (N, E, w)$. Such graphs are a suitable representation when modelling, e.g., social or biological networks. The contribution addresses the issue of neuro-symbolic node representation learning. Even though methods such as, e.g., node2vec (Grover & Leskovec, 2016) or graph neural network-based models (Kipf & Welling, 2017), have been shown to perform well for the task of structural node classification (where nodes have no attributes), such systems either do not scale to larger networks or cannot be, at least partially, inspected and systematically perturbed. By building on previous work (Kralj et al., 2018) that has shown that symbolic structural representations based on personalized PageRank vectors offer adequate performance, we have improved the methodology to scale to much larger networks whilst retaining the origin representation's classification performance. As the proposed method is partially symbolic, we demonstrate that by adopting the neuro-symbolic paradigm, the relation between the performance and compression can be directly studied. Further, the proposed method adopts the notion of node pivoting, making it scale to very large networks (which we empirically evaluated).

Related journal paper

Škrlj, B., Kralj, J., Konc, J., Robnik-Šikonja, M., & Lavrač, N. (2021). Deep node ranking for neuro-symbolic structural node embedding and classification. *International Journal of Intelligent Systems*, 1–30. https://doi.org/https://doi.org/10.1002/int.22651

Automating text classification with autoBOT (see Section 4.2)

The next domain of interest were texts. This type of input does not adhere to a single, most suitable representation; for example, a document can be represented based on its part-of-speech-tag structure, keywords, subword information or more semantic representations learnable with neural language models. The key contribution of this work was the investigation of the intractable problem of constructing the appropriate representation ensemble, whilst maintaining the system's explainability via coefficients of simple (explainable) learners, as well as representation type-level scores obtained in the process of evolution. The key output is thus a simple-to-use system that operates with non-specialized hardware, offers better performance than naïve symbolic baselines, and can be competitive to the state-of-the-art language model-based learners, which are not explainable.

Related journal paper

Škrlj, B., Martinc, M., Lavrač, N., & Pollak, S. (2021). autoBOT: evolving neuro-symbolic representations for explainable low resource text classification. *Machine Learning*, 989–1028. https://doi.org/10.1007/s10994-021-05968-x

Self-attention and feature ranking (see Section 5.2)

We were exploring the link between feature importance computation and the neural attention mechanism. This contribution was, at the time of writing, one of the first papers that explored the implications of considering the neural attention mechanism, conventionally used in architectures like transformers, as a means to obtain feature importances. We were especially interested in the notion of self-attention, i.e., the set of weights that directly relate individual attributes to a real-valued score. The key insight related to this contribution is that feature ranking can be used to evaluate this idea quantitatively, as the classification performance of the top k features based on a given ranking can be directly evaluated.

Related conference paper

Škrlj, B., Džeroski, S., Lavrač, N., & Petkovič, M. (2020). Feature Importance Estimation with Self-Attention Networks. In G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, & J. Lang (Eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)* (pp. 1491–1498). IOS Press. https://doi.org/10.3233/FAIA200256

Embedding-based feature ranking (see Section 5.3)

Apart from studying the behaviour of neural-only systems, and trying to link it to a given symbolic space, a promising application of the neuro-symbolic paradigm is in speeding up existing learning algorithms. This is achieved by a novel feature ranking approach from the Relief family, capable of leveraging learned instance representations for seamless scaling to very high-dimensional, sparse data sets. The key idea of this contribution revolves around efficient representation learning paradigms available nowadays, capable of operating in high-dimensional sparse regimes and returning low-dimensional representations of the objects of interest, i.e., the instances, in the case of this contribution. As the Relief family of algorithms revolves around instance comparison for the purpose of ranking, we demonstrated that such comparisons can be made in a learned space, inducing provably lower computational complexity compared to variants that operate in the original space. The contribution also contains an extensive empirical evaluation of this claim, demonstrating that lower-dimensional representations are a promising endeavour, speeding up the Relief family of algorithms in order for them to handle contemporary-scale data sets (e.g., biological data sets that can contain tens of thousands of features).

Related journal paper

Škrlj, B., Džeroski, S., Lavrač, N., & Petković, M. (2021). ReliefE: feature ranking in high-dimensional spaces via manifold embeddings. *Machine Learning*, 1–45. https://doi.org/10.1007/s10994-021-05998-5

1.4.2 Other related Contributions

The following set of contributions impacted the development of the core thesis contributions, despite not being fully focused on the common thesis thread of neuro-symbolic learning.

Symbolic Node Embedding (see Section 6.1); Mežnar, S., Lavrač, N., & Škrlj, B. (2020). SNoRe: scalable Unsupervised Learning of Symbolic Node Representations. *IEEE Access*, *8*, 212568–212588. https://doi.org/10.1109/ACCESS.2020.3039541

Semantic Reasoning from Embedding-based Communities (see Section 6.2); Škrlj, B., Kralj, J., & Lavrač, N. (2020). Embedding-based silhouette community detection. *Machine Learning*, *109*(11), 2161–2193. https://doi.org/10.1007/s10994-020-05882-8

Semantic Feature Construction with tax2vec (see Section 6.3); Škrlj, B., Martinc, M., Kralj, J., Lavrač, N., & Pollak, S. (2020). tax2vec: constructing Interpretable Features from Taxonomies for Short Text Classification. *Computer Speech & Language*, 101104. https://doi.org/https://doi.org/10.1016/j.csl.2020.101104

Contextual Keyword Identification with TNT-KID (see Section 6.4); Martinc, M., Škrlj, B., & Pollak, S. (2021). TNT-KID: transformer-based neural tagger for keyword identification. *Natural Language Engineering*, 1–40. https://doi.org/10.1017/S1351324921000127

On Attention Vectors and Explanations (see Section 6.5); Škrlj, B., Sheehan, S., Eržen, N., Robnik-Šikonja, M., Luz, S., & Pollak, S. (2021). Exploring neural language models via analysis of local and global self-attention spaces. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 76–83. https://aclanthology.org/2021.hackashop-1.11

Interactive Exploration of Causal Drug-Target Interactions (see Section 6.6); Škrlj, B., Eržen, N., Lavrač, N., Kunej, T., & Konc, J. (2020). CaNDis: a web server for investigation of causal relationships between diseases, drugs and drug targets. *Bioinformatics*, *37*(6), 885–887. https://doi.org/10.1093/bioinformatics/btaa762

1.5 Structure

This chapter served as an overview of the conducted work, the problems addressed and the main contributions. The remainder of this dissertation is structured as follows. In Chapter 2 we present the related work relevant to this dissertation. The chapter includes selected concepts from the areas of symbolic learning, propositionalization, neural network-based learning and, more generally, representation learning. This chapter also focuses on the notion of *neuro-symbolic learning*, key contributions in this field in the last ten years and its relation to state-of-the-art neural/symbolic approaches. Finally, the chapter discusses the notion of automatic learning (autoML) systems, existing state-of-the-art in this domain, and the main implications of being able to use such systems on real problems supported with existing implementations/use cases.

We begin the description of these contributions by presenting our work on relational data in Chapter 3. This chapter includes two selected contributions to the better understanding of how the neuro-symbolic paradigm can be adopted to solve the tasks of relational classification and node embedding tasks. The chapter first introduces the reader to the problem of relational learning, typical applications and the current state-of-the-art approaches. Next, we present our work on propositionalization and embeddings, two dominant paradigms for transforming complex data, such as relational tables, into simpler, propositional form suitable for learning. By discussing both the relationship between the two paradigms and two implementations of the discussed ideas, we demonstrate that neuro-symbolic relational learning is one of the possible ways to improve the scaling of

existing methods beyond the reach of the current symbolic state-of-the-art. We continue by describing the proposed Deep Node Ranking (DNR) algorithm. This approach explores how propositionalization and embeddings intertwine when considering simpler relational inputs such as weighted graphs. We demonstrate that the existing symbolic (propositionalization via PageRank-based) method can be extended with a neural network that serves as a compression engine, offering much more compact representations which retain most of the information. By coupling symbolic with sub-symbolic representation learning, we demonstrate that the neuro-symbolic paradigm offers the answers to the questions that cannot be directly addressed with sub-symbolic-only learning; for example, by being able to control the subgraph that is the input to embedding learning, the user has fine-grained control over the representation learning process.

Next, we discuss the developed autoML system aimed at text-based neuro-symbolic classification in Chapter 4. The chapter introduces the reader to the notion of text classification, the relevance of this task, and the open problems addressed. This is followed by the contribution describing autoBOT, the in-house autoML system aimed at exploring how far low-resource learning can be pushed by adopting the idea of *representation evolution*. In the same chapter, we discuss a computational framework implemented to properly scale the developed autoML system across hundreds of machines. As individual evolutions are independent, the framework is able to asynchronously send, summarize and retrieve the best models completely automatically. The end goal of this contribution was to build a framework, which, once given the training and testing data, exploits the available computing power to find the best explainable (regularized) classifier. The contribution is one of the first to explore representation learning at this scale for the task of neuro-symbolic text classification. We also discuss our contributions related to neuro-symbolic learning from tabular data in Chapter 5. The chapter includes two main contributions. We first present our attempt at linking parts of the tabular input space (attributes) with corresponding neural network weights. We hypothesized that such bijections could offer direct attribute ranking, which we evaluated empirically, demonstrating that if the underlying neural network converges to a good classifier, the attention-based ranking is of higher quality. The second contribution presented explores how contemporary embedding learners such as UMAP (McInnes et al., 2018) can be of use when performing feature ranking directly. We demonstrated that both theoretically and empirically, by considering embeddings of instances instead of the original space instances, the process of ranking can be substantially faster. When considering very high-dimensional data sets, we demonstrated that the proposed ReliefE is one of the few solutions that are able to output rankings in a reasonable time. This method adopts the *neuro-symbolic* paradigm in the sense that sub-symbolic representations are used to obtain a symbolic output (ranking).

Further, in Chapter 6, we present the collection of other contributions (papers), which were not the key focus of this thesis even though they influenced the main set of contributions. The key ideas presented in the individual contributions are summarized. In Chapter 7, we discuss the implementation aspects of the presented contributions. Having stable, easy-to-use implementations of a given method is becoming a norm due to the increasing competition within different sub-fields of machine learning. Most of the presented work was implemented in the form of simple-to-use Python libraries, capable of exploiting available computing power, should it be available. We present the developed software libraries and discuss how the unit testing procedures helped us ensure more robust implementations in the (hopefully) longer run. We round up this work with our remarks collected in Chapter 8. We first offer an overview of the developed methods, key results and their applications. We conclude the thesis with a section discussing possible further work, and more broadly, the possible limitations of neuro-symbolic computing in general.

Chapter 2

Related Work

> I do not fear computers. I fear the lack of them.
>
> Isaac Asimov

This chapter provides an overview of the key topics covered in the main contributions of this dissertation. We begin by presenting the machine learning paradigms relevant to this work. We discuss the notion of symbolic learning, followed by deep learning and neuro-symbolic learning. We conclude the chapter with an overview of automatic machine learning and evolutionary computation principles relevant to this work.

2.1 Symbolic Learning

Ever since the first computers, the question of task automation has intrigued multiple generations of researchers. Some of the first discussions on constructing intelligent agents date back to 1950s (Turing, A. M., 1950). The first widely explored paradigm of automated learning was *symbolic learning*. Here, models in the form of a collection of logical rules (Holte, 1993; Michalski et al., 1986) or trees (Quinlan, 1986) are learned by using heuristics-guided algorithms. For example, decision tree learners were successfully employed to understand data from various sources better and are especially suitable when a better understanding of, e.g., clinical or biomedical data is desired. An example decision tree is shown in Figure 2.1. Decision trees can also be decomposed into simple rules. Continuing the example in Figure 2.1, example (propositional) rule is:

$$\text{trip}(true) \leftarrow equipment(true) \wedge weekend(true) \wedge sunny(true),$$

indicating how a decision tree is interpreted when considering different criteria. The common property of all symbolic learning approaches is that they are interpretable in the sense that the user can directly inspect which parts of the feature space were crucial for the final model's prediction. Scaling up, e.g., tree learning, has been an ongoing research endeavor (Gama et al., 2013), making this branch of models applicable in many contemporary scenarios, where the data sets cannot be processed without proper parallelism and potential multi-machine scaling or stream-based approaches.

Apart from symbolic *predictive* models, a learning paradigm which is often considered in the *pre-processing* phases also relevant to this work is *feature ranking* (Chandrashekar & Sahin, 2014; Kira & Rendell, 1992). Here, given mostly tabular inputs (e.g., table and the target space), the algorithm's task is to pinpoint which features are the *most relevant* for, e.g., differentiation between the target classes. The feature ranking process is illustrated in Figure 2.2. The main application of this branch of algorithms is when

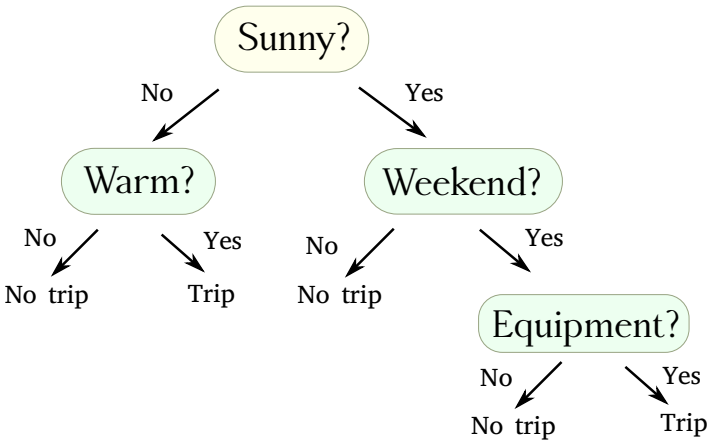


Figure 2.1: A decision tree determining a trip departure.

attempting to better *understand* the data itself, and not necessarily to construct superior predictive models. For example, contemporary biological single-cell sequencing data sets can consist of thousands of features (genes). Understanding which (often small) subset of them is relevant for differentiating between the observed phenotype is of high relevance when learning about a given phenomenon (Ibrahim & Kramann, 2019). Recent variants of feature ranking algorithms can, for example, also incorporate hierarchical information, which is often natively present when working with biological data (Slavkov et al., 2018). Furthermore, in recent years, simultaneous consideration of multiple targets during ranking has shown promising results (Petković et al., 2020). The next example of the symbolic

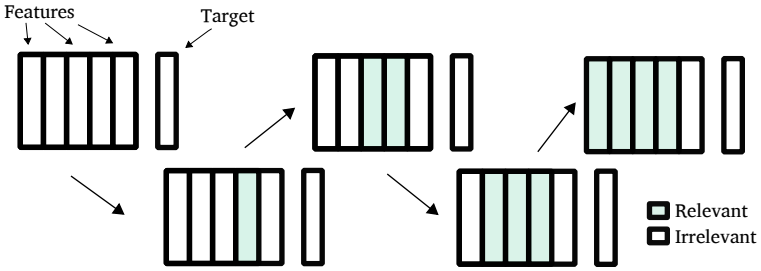


Figure 2.2: Schematic overview of feature ranking. After a sufficient number of iterations, the final set of relevant features can be selected based on the scoring of all features.

learning paradigm relevant to this work is *inductive logic programming* (ILP) (Cropper et al., 2020; Lavrač & Džeroski, 1994; Muggleton, 1992; Shapiro, 1981). The key property of these models is their symbolic nature – they consist of a collection of logical clauses, interpretable by e.g., a domain expert. Zeroth- or first-order logic is commonly used to construct such (relational) rules. An illustrative first-order rule is stated next.

$$\text{siblingPair}(A, B) \leftarrow \text{parentOf}(A, C) \wedge \text{parentOf}(B, C)$$

Note that the logical variables are in this case not boolean. These variables can also be predicates, possibly yielding higher-order relations. These models are trained via different search procedures (e.g., top-down, bottom-up, combination of the two), and offer discovery

of simple generalizations in both propositional and relational domains. An illustration of the two main search paradigms used in rule induction is shown in Figure 2.3.

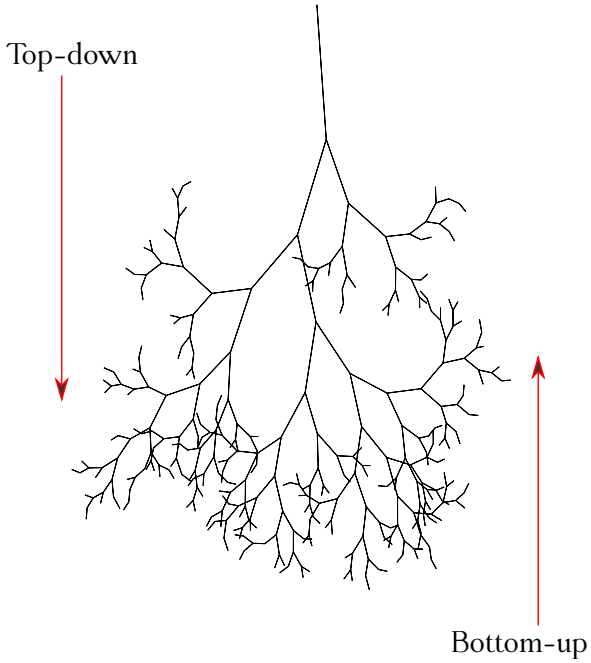


Figure 2.3: The two types of computational search.

This branch of learning has been, for example, used for exploration of codebases (Sivaraman et al., 2019) and in agriculture (Matsumoto et al., 2017) for yield optimization. One of the most widely used algorithms of this paradigm is Aleph (Srinivasan, 2001). Note that the ILP task generally attempts to find a logic program that covers all positive examples and does not cover the negative ones, whilst considering a collection of background knowledge.

The core focus of the ILP community is working directly with relational data. The focus of this work, however, is the part of ILP focusing on *propositionalization* – the process of transforming a relational database into a propositional (tabular) one. If such transformation is automated, we refer to it as *symbolic representation learning*. Symbolic representation learning has emerged as a sub-field of ILP (Kramer et al., 2001). The field of symbolic representation learning mostly considers the transformation between relational and propositional data structures, i.e., the process of *propositionalization* (see, e.g., (Kramer et al., 2001) for an overview). Here, a learning-ready (tabular) representation is obtained via, e.g., sampling of individual relations, which jointly form a single vector that describes a given instance. Existing methods which adopt a similar idea when considering learning from relational databases include SINUS (Lavrač & Flach, 2001), 1BC (Flach & Lachiche, 1999) and RSD (Železný & Lavrač, 2006). The common point of the mentioned propositionalization approaches is that they attempt to perform intermediary generalization and thus yield features that are complex relational structures (first-order). The most recent advances in the field of propositionalization aim to exploit as many computational resources as possible via direct propositionalization of whole relational databases (Perovšek et al., 2013). A schematic overview of Wordification is shown in Figure 2.4. This work builds on the insights from Wordification, namely that simpler table-attribute-value triplets, when exhaustively sampled, already offer similar performance to state-of-the-art

if considered by a learning algorithm capable of capturing non-linear relations. This thesis pushes Wordification-based propositionalization ideas further to the boundary between neural and symbolic representation learning, both when considering information-rich relational databases (with limited structure), and structure-rich real-life networks.

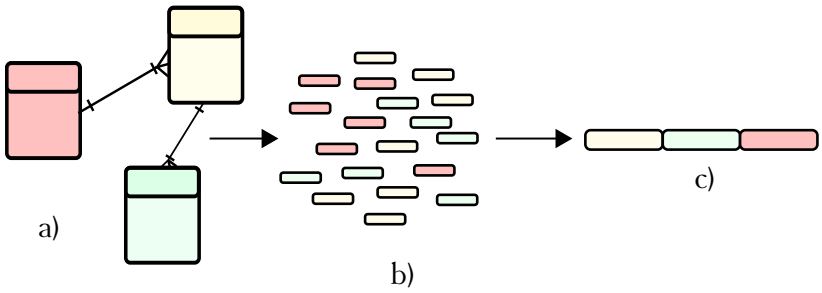
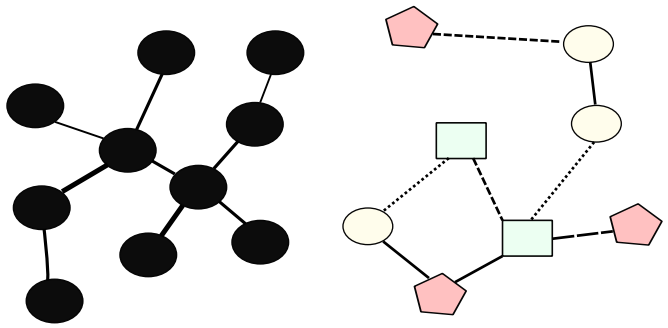


Figure 2.4: Wordification of a relational database. First (a), foreign-key-based traversal is conducted to identify linked instances. Next (b), a joint space with respect to different tables is constructed and (c), vectors with respect to a single instance from the target table are concatenated.

While relational learning has been, from the ILP perspective, mostly concerned with relational databases, symbolic representation learning from simpler relational structures, such as, e.g., *complex networks* also offers an interesting research venue. Relevant to this work are two main types of such networks: homogeneous and heterogeneous information networks, illustrated in Figure 2.5. The main difference between the two network types is that homogeneous networks are either directed or undirected, with their edges being weighted. However, the heterogeneous networks commonly consist of typed nodes, links, weights on the links and optional node/link features.



(a) Homogeneous. (b) Heterogeneous.

Figure 2.5: The two network types. Heterogeneous networks can entail richer structure, considering also node/link types/features. A common approach to working with heterogeneous network includes their transformation into homogeneous ones (Kralj et al., 2018).

Examples of previous work, where propositionalization-like representation learning was used for learning from heterogeneous information networks, include a methodology by Grčar et al. (Grčar et al., 2013) and HINMINE (Kralj et al., 2018). These methodologies explored how heterogeneous information networks can be first *decomposed* into multiple

homogeneous networks, which can be subsequently propositionalized via the application of personalized PageRank algorithm (Page et al., 1999). The methodologies differ with respect to the type of aggregation of the representations prior to learning and the type of decomposition used. Further, systems that perform *graph propositionalization* (Karunaratne & Boström, 2009) were shown to perform well when considered simultaneously with random forest-based learners. One of the key contributions of this work (Lavrač et al., 2020) explores how these existing approaches could be scaled/improved by also considering embedding-based representation learning.

2.2 Sub-symbolic Learning

The second widely-adopted paradigm of learning has re-emerged recently due to the increased computational power available to the research community via, e.g., Graphics Processing Units (GPUs) (Goodfellow et al., 2016; LeCun et al., 2015). The paradigm considers the construction of *neural networks*, a type of machine learning model that iteratively optimizes a plethora of real-valued weights ordered in layers (e.g., a multipartite graph). In contrast to the symbolic paradigm, neural networks are mostly trained via the process of *backpropagation*, a forward-backwards procedure that gradually learns to associate the input space with the output space via scalar products of intermediary weights spaces. This paradigm of learning, initially explored more than 50 years ago (Rosenblatt, 1957), has resurged as *deep learning* due to multiple hidden layers considered in contemporary neural network architectures (Vaswani et al., 2017).

The process of backpropagation can be understood as follows. First, matrix multiplications offer a direct transfer of information from the input to the output (forward pass). Next, the errors, once computed, can be *back-propagated* from the last layers to the input layer. This step requires the computation of derivatives of each weight with respect to the output, which is possible due to the chain rule. Let $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, and $f : \mathbb{R}^n \to \mathbb{R}$. Let $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$. Then the general expression for computing a given partial derivative can be stated as

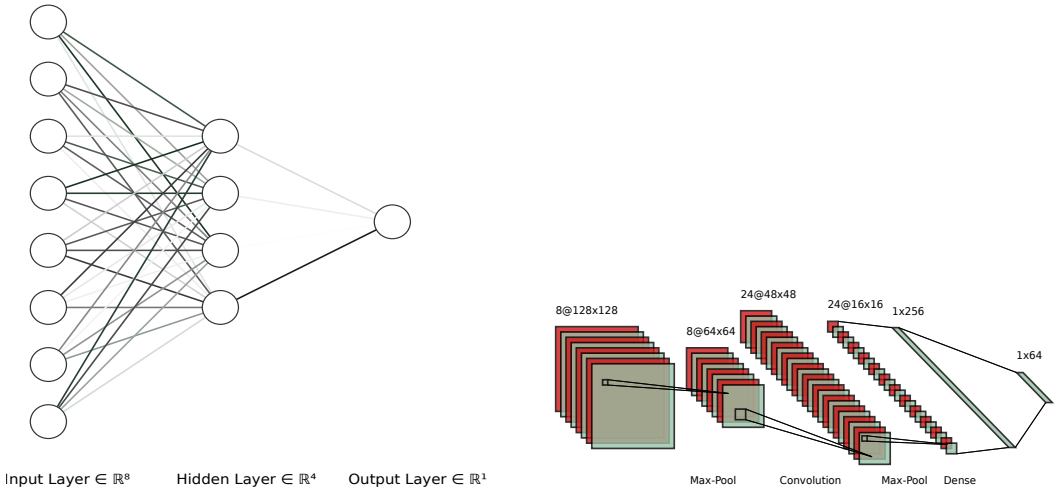
$$\nabla_{\boldsymbol{x}} z = \boldsymbol{J}^t \cdot \nabla_{\boldsymbol{y}} z.$$

Here, $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ is the Jacobian matrix of g (i.e. $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$). Note that even though neural networks are commonly comprised of *tensors*, i.e., multi-dimensional objects that are not just vectors, as shown in the formula above, the gradient computation works the same way with an additional step of *tensor flattening* (Goodfellow et al., 2016). The obtained gradients are finally used (third step) to update the weights via an optimization algorithm such as, e.g., the stochastic gradient descent.

Nowadays, neural networks can be built and trained seamlessly via systems such as PyTorch (Paszke et al., 2019). As deeper architectures are commonly adopted in practice, diagrams like the one shown in Figure 2.6 are commonly used, where nodes represent individual vector products (with activations), and the edges represent the weights.

One of the first modern tasks where deep learning prevailed were computer vision tasks such as image recognition, segmentation and similar tasks (Voulodimos et al., 2018). In recent years, however, state-of-the-art performance was also observed in the domains of natural language processing (Devlin et al., 2019) and graph-based learning (Kipf & Welling, 2017).

Neural networks learn to associate the input to output spaces via iterative updates of real-valued weights. As such, the *intermediary* weight spaces can be already considered as *representations*. Albeit the first breakthrough in adopting deep learning-based



(a) Feedforward architecture. (b) Convolutional architecture.

Figure 2.6: Two examples of commonly used neural network architectures. For the case of basic single-hidden-layer feedforward neural network architecture (a); weights are emphasized based on random initializations (no edge $\implies w = 0$). The second architecture (b) represents a contemporary multi-hidden-layer neural network with multiple convolutional and pooling layers.

methods at scale were in the field of computer vision, the first endeavours to better understand the learned representations first raised considerable attention with the representation learners such as word2vec (Mikolov et al., 2013) and later doc2vec (Le & Mikolov, 2014). This branch of, e.g., word representation learners opened multiple novel research venues, focused on compressibility, semantic change in time and overall usefulness of such representations for downstream tasks such as document classification of similar. Another significant improvement in the area of natural language processing (NLP) emerged with the transformer-based architectures (Devlin et al., 2019). Here, the focus was not, as e.g., when considering doc2vec, to obtain a static, non-contextual representation, but to obtain *contextual* representations or perform learning directly. Even though their initial focus was on solving downstream tasks of language understanding, transformer-based representations have become a research area of their own (Reimers & Gurevych, 2019, 2020).

Apart from the advances in the field of NLP, the area of relational learning has similarly been subject to significant improvements in the last 15 years. Broadly, two distinct branches of deep-learning-inspired relational representation learning have emerged, which have started to intertwine in recent years.

The first type of learning was inspired by the ever more available collections of subject-predicate-object triplets, forming *knowledge graphs*. This field of knowledge graph embedding explores how representations of both entities and relations can be *learned* and used to solve downstream tasks such as graph completion. The canonical example of this branch of methods is TransE (Bordes et al., 2013) – its key idea is that it models the relations via linear combinations of entity representations. Other more recent variations of this method include, for example, its extension to the space of quaternions (S. Zhang et al., 2019), exploration of rotation-aware embeddings (Sun et al., 2019), inclusion of complex number-based representations (Trouillon et al., 2017) and similar.

In parallel, the field of relational representation learning focusing on more global network structures emerged. Inspired by the word2vec-based branch of algorithms, Deep-

Walk (Perozzi et al., 2014) was one of the first node representation learners which scaled to larger networks, comprised of thousands of nodes. Here, the idea of random walk-based sampling is exploited to estimate which neighbours are present around a given node and how they can be encoded into a compact, dense real-valued representation. In the following years, multiple generalizations/optimizations of this type of algorithm were proposed. Examples include node2vec (Grover & Leskovec, 2016), NetMF (Qiu et al., 2018), a heterogeneous network-based adaptation metapath2vec (Y. Dong et al., 2017) and similar. The common point of these methods is a sampling-based estimation of node/edge representations. With some delay, however, an alternative representation learning framework emerged, based on the idea that the adjacency structure of a given network can be directly exploited during backpropagation. The graph-convolutional neural networks (Kipf & Welling, 2017) was one of the first to propose a stable normalization-based layer that effectively offered direct learning of node representations. In the following years, multiple adaptations were proposed, including the graph-attention networks (Veličković et al., 2018), graph-isomorphism networks (Xu et al., 2019) and many extensions to heterogeneous networks (C. Zhang et al., 2019).

2.3 Neuro-symbolic Learning

The two discussed paradigms of learning, symbolic and sub-symbolic, are nowadays able to solve different problems with little overlap. The symbolic branch of methods remains in widespread use when obtaining compact explainable patterns directly associated with, e.g., a given target space. Such models offer direct inspection, are easier to debug and validate. Ensembles of such models, are less (or not at all) explainable, but can offer performance competitive to neural network-based learners. One of the problems of symbolic learning is its inability to fully exploit the developments at the hardware level. Even though tree-based systems such as, e.g., XGBoost (Chen & Guestrin, 2016) are able to exploit GPUs, symbolic learners are mostly CPU-based and often do not support multi-threading.

On the other hand, the neural paradigm mitigates these issues and is arguably as successful as it is today to a large extent due to the engineering effort directed at the individual parts of, e.g., backpropagation (e.g., gradient computation). Neural network-based models continue to dominate most tasks with high-dimensional input spaces such as texts and images; their dominance revolves around their ability to automatically create abstractions of the (raw) input signal space, which is more resemblant to how, e.g., humans perceive their surroundings. Albeit able to handle such fine-grained inputs with ease, neural network-based models continue to underperform on tasks requiring symbolic reasoning, regardless of their size.

The domain of neuro-symbolic learning aims to mitigate some of these issues by investigating how the two paradigms can co-exist as naturally as possible as parts of the same solution/optimization. The interest in neuro-symbolic learning resurged recently (Mao et al., 2019) by the development of a neuro-symbolic system that partially operates in a symbolic and partially in a sub-symbolic space, used to distil human-understandable concepts from images. In addition, the recent work on closing the loop between recognition (neural) and reasoning (symbolic) (Q. Li et al., 2020) introduced a grammar model as a symbolic prior to bridging the neural perception and symbolic reasoning, alongside a top-down, human-like induction procedure. This work demonstrated that such a combined approach significantly outperforms the conventional reinforcement learning-based baselines; by including explicit symbolic constraints, parts of the unsuitable search space were automatically omitted.

Despite the neural network-based models' dominance when working with image-based

data, these models' capability of performing reasoning directly from the learned representations remains a challenging open problem (see, for example, the recent ARC challenge (Chollet, 2019)). To address this issue, the Microsoft research division recently explored the interplay between visual recognition and reasoning (Amizadeh et al., 2020). They introduced a framework to isolate and evaluate the reasoning aspect of visual question answering separately from its perception, followed by a calibration procedure that explores the relationship between reasoning and perception. Further, a neuro-symbolic approach to logical deduction was proposed as *Neural Logic Machines* (H. Dong et al., 2019). This architecture was shown to have inductive logic learning capabilities, which was demonstrated on simple tasks such as sorting. Finally, the two recent approaches from the field of inductive logic programming (ILP) explored the interplay between the logical input structures and how they perform when associated via neural network learning. The Deep Relational Machines (Lodhi, 2013) were one of the first approaches to showcase the utility of combining the two paradigms. Further, recently, researchers (Srinivasan et al., 2019) explored how Deep Relational Machines can be *explained*, emphasizing that being able to explain what a given pattern discovery/recognition system does is highly relevant, e.g., in the field of biomedicine.

Apart from exploring the propositional setting, neuro-symbolic learning has also been actively explored when considering first-order rules. For example, TensorLog (Cohen et al., 2020) is a recent advancement, which offers a thorough investigation of the relation between tensor-like representations of logical clauses and their discovery/identification. Prior to this work, the relation between Boltzmann machines and relational learning was also explored (Kaur et al., 2018). The branch of research between ILP and neural network learning has also gained significant traction in recent years (Marra et al., 2020). Finally, the research that explores how to employ logical constraints during neural network learning also offered promising results (Pinkas & Cohen, 2019).

Although actively explored, the notion of neuro-symbolic representation learning has, to our knowledge, not yet been considered in the context of node representation learning, which is the key focus of this work.

2.4 Automating Machine Learning

In this section, we discuss the notion of *automatic learning* – the study of creating systems capable of solving a given task with *little-to-no* human intervention. The process of machine learning and representation learning, regardless of it being of symbolic or neural nature, is in most cases governed by *hyperparameters*, i.e., *tunable variables* that govern the learning progress/properties (He et al., 2021). With the increasing availability of computing power, manual tuning of hyperparameters is gradually being overtaken by automatic processes, i.e., *meta-learning* algorithms. The purpose of this second layer of learning is to automate the redundant and time-consuming manual optimization of a learning process and rely more on the available computing resources; the amount of available computing resources keeps increasing, even though this might not remain the case (Theis & Wong, 2017). The notion of meta learning can be understood as learning from *prior* experience in a systematic manner (Lemke et al., 2015; Vanschoren, 2019). Even though general (naïve) solutions for meta-learning tasks are – due to the no free lunch theorem – practically impossible (Wolpert & Macready, 1997), optimization within subspaces of the relevant solution space can lead to efficient and scalable solutions. For example, when designing a routing search engine, by incorporating the historical data on city-to-city traversals, the algorithm designers do not need to treat all candidate paths as equiprobable and jump-start the search from the existing solution(s). Furthermore, greedy search is also commonly used to drastically re-

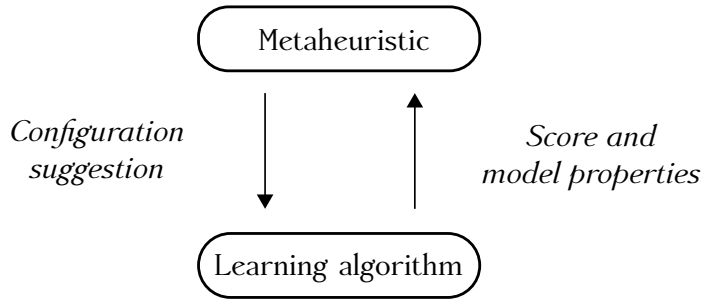


Figure 2.7: Schematic overview of an autoML system. The process is iterative, optimally yielding better and better solutions.

duce the search space size, even if it is neglecting parts of the space containing reasonable solutions. Design and development of systems capable of automatic model configuration and data preprocessing has been an active research area in the last years. We refer to a system capable of automatic model tuning/data configuration as an *autoML system* (Kotthoff et al., 2019). Examples of existing autoML systems which have already shown promising performances include autoWEKA (Kotthoff et al., 2019), auto-sklearn (Feurer et al., 2019) and TPOT (Olson & Moore, 2016). Many autoML systems can be understood as *search* across a non-convex space comprised of configurations/representations. The minimization problem addressed can be in its *general form* stated as follows;

$$\text{Solution} \approx \underset{\substack{\Theta \in \text{algorithmSpace} \\ \times \text{hyperParamSpace} \\ \times \text{transformationSpace}}}{\arg\min} \mathbb{E}\Big[\textsc{Loss}\big(\mathit{learnerClass}, \Theta, \text{data}\big)\Big].$$

The stated problem is a generalization of the one discussed in (Kotthoff et al., 2019), as it also includes the space of possible transformations; this space includes e.g., dimensionality reduction and embedding construction. Note that the expected value of the loss function for a given configuration Θ is commonly estimated via cross-validation. Even if the stated criterion addresses the construction of the final learner, note that if the Loss is not representative, the solution obtained as the result of the minimization will not necessarily *generalize*/perform well on unseen data. Note that the Loss function is task-independent; autoML systems are capable of addressing supervised/unsupervised learning problems. In recent years, substantial research effort has been focused on the design and optimization of autoML systems for different domains. Widely used autoML libraries include for example TPOT (Olson & Moore, 2016), OBOE (Yang et al., 2019), H2O autoML[1], FLAML (Wang et al., 2021), ML-Plan (Mohr et al., 2018), auto-XGBoost (Thomas et al., 2018), GAMA (Gijsbers & Vanschoren, 2019) and others. Even though the early systems focused mostly on tabular data due to the previously available algorithm libraries for this domain, other, less structured data sources are being actively explored. Examples include e.g., automatic exploration of neural network topologies for the task of computer vision (Tan & Le, 2019), graph neural network topologies for relational regression/classification (Pareja et al., 2020). Further, meta learning packages built around widely used deep learning libraries such as Keras have gained popularity in recent years

[1] https://github.com/h2oai/h2o-3

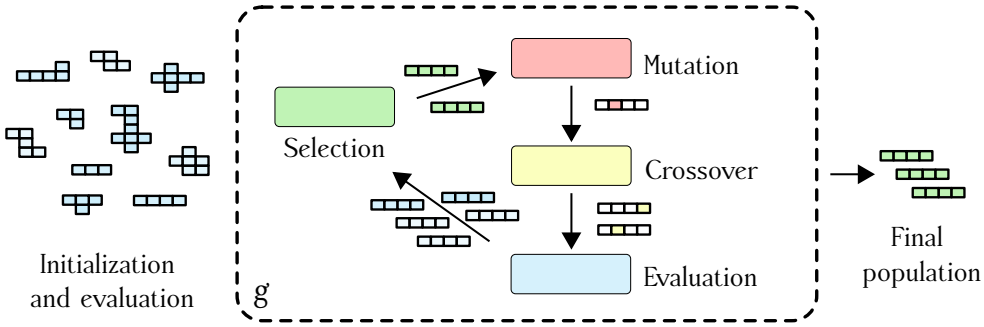


Figure 2.8: An overview of the the early genetic algorithm idea. The g corresponds to the number of iterations of population refinement.

(Auto-Keras) (Jin et al., 2019). Recently, many novel autoML methods have been introduced and offered in a form usable to machine learning practitioners. For example, auto-sklearn (Feurer et al., 2019) and autoWEKA (Thornton et al., 2013) are approaches for automatic learning from tabular data. Their goal is to minimize the user's input during hyperparameter tuning and model selection, which they achieve via Bayesian optimization. Further, the process of identifying a suitable deep learning architecture was shown to be suitable for optimization; an example is the NASnet project (Zoph et al., 2018), where large-scale exploration of neural network architectures is conducted automatically.

The field of neural architecture search has grown significantly in recent years (Elsken et al., 2020). Finally, recent trends indicate that understanding the *transferrability* in the latent space might offer novel and faster ways for neural network model training. An example of this new paradigm, also relevant to the final part of this dissertation, is dataset2vec (Jomaa et al., 2021).

A prominent source of freely available information are texts, which are the key focus of the autoML-related part of this thesis. More concretely, the final part of this dissertation explores the implications of meta-learning for the task of representation learning. We continue with a more detailed overview of the metaheuristic optimization algorithm relevant to this thesis. One of the algorithm groups which have stood the test of time are genetic algorithms, which are part of a broader spectrum of methods termed *evolutionary computation*. These algorithms mimic the behaviour of, e.g., cell division/DNA replication and offer a highly parallelizable metaheuristic optimization procedure suitable for most optimization problems. Even though there are no real guarantees regarding their general performance (the no free lunch theorem (Wolpert & Macready, 1997)), they consistently offer a simple-to-implement and efficient automation of many real-life optimization endeavours. Moreover, with the increasing amounts of available computing resources, the relevance of this and similar types of algorithms are gaining traction in the broader machine learning community. Widely used already in the 1980s (Davis, 1991), the first variants of genetic algorithms explored simple search schemes that can be summarised as shown in Figure 2.8.

Later developments in this field focus more on multi-objective optimization of the exploration of Pareto fronts, efficient implementations and scalability (Corus & Oliveto, 2017; Doerr et al., 2017). Their applications are becoming increasingly more relevant due to the larger amounts of available computing resources available. Practical applications include

energy management (Leonori et al., 2020), and recently autonomous driving research (D. Li et al., 2020). We were mostly interested in how the fields of genetic algorithm design and the design of automated learning systems come together – the recently introduced field of automatic machine learning (autoML) is becoming a promising research direction with solutions already deployed at scale.

2.5 Scalability Aspects of Learning

The key focus of this work is a collection of methods which *scale* with the available computing power at a given researcher's disposal. There are multiple aspects to a learner's *scalability*. These include the study of a given algorithm's asymptotic behaviour with respect to consumed space and time, the implementation and its relation to the hardware layer, type of parallelism needed to scale to, e.g., threads within the same machine, but also across multiple machines, which is currently near the limit of physically implementable solutions. The study of the computational complexity of machine learning offers analytical, often closed-form insights into the asymptotic behaviour of how a given learning process behaves. Further, it concerns *scaling laws* – the behaviour of a given system when varying, e.g., data/model size. In this work, we investigate whether parallelism can offer faster/better performance of existing algorithms and whether algorithmic improvements of existing approaches can be obtained. The notion of scalability is becoming an increasingly relevant topic of interest due to the increasing amounts of available data and available computing resources. If a given method does not scale – cannot process large amounts of data – it cannot benefit from additional data, which is a limiting factor that can determine its usefulness.

The neural network-based learning paradigm commonly adopts minibatch-based learning, meaning that instances can be fed into the algorithm in small, tractable batches that fit into RAM. On the contrary, e.g., propositionalization-based approaches often first construct a large matrix (mostly sparse), which is next used for learning. If the implementation of the considered downstream learner is not able to handle the matrix in a sparse data format, its space complexity quickly becomes intractable in many real-life cases. Furthermore, many ILP learners are not necessarily optimized for speed – even though parts of e.g., rule search could be parallelized to tens or hundreds of threads simultaneously with minimal overhead, this is seldom implemented in practice.

Finally, recent machine learning algorithms are capable of exploiting not only CPU-based resources, but also GPU-based ones. This hardware-level capability often forces the algorithm designer to think in terms of matrix factorization-like procedures rather than considering direct search. Finally, the aspect of scalability can transcend a single machine-based computation and concerns grid-level computation. By default, not many methods are computing grid-ready – their computing environments need to be specifically adapted and optimized first. This aspect is also explored as part of this thesis, as many of the experiments would not have been possible without the national supercomputing grid infrastructure SLING[2].

[2]https://www.sling.si/sling/

Chapter 3

Neuro-symbolic Learning from Relational Data

> In the midst of chaos, there is also opportunity.
>
> Sun Tzu

In this chapter, we focus on the domain of *relational data*. We first give a more broad overview of the types of relational learning considered and the reasons why we pursued the presented work. We then describe our contributions to the domain of neuro-symbolic relational learning.

3.1 Overview of Relational Learning

Relational learning concerns the analysis of data sets that include explicit relations. Examples include relational databases (foreign key-based relations) and homogeneous/heterogeneous complex networks (links correspond to relations). Compared to traditional learning, where no adjacency structure between the instances of interest is known, relational learners are able to *exploit* the information of relations directly, resulting in better performance or solving a task not addressable with conventional learning directly (e.g., structural node classification).

A schematic overview of the problems discussed in this chapter can be seen in Figure 3.1. The scheme demonstrates the key problem addressed in the presented contributions. We are interested in finding a mapping between a given relational structure such as, e.g., a relational database or an attributed (sub)network and a designated output space. Example output spaces include the set of possible protein functions, people's properties, named entity flags, or similar.

The learning task addressed with the presented contributions is *classification* – mapping the entities of interest, such as nodes in a network, to one or more *labels* from a finite label space. In particular, we address *binary*, *multiclass* and *multilabel* classification. The main difference between these three paradigms is that the multilabel classification corresponds to a scenario where multiple labels can be *simultaneously* assigned to a given entity. Multiclass and binary classification correspond to the process of assigning only a single label per entity, with binary classification corresponding to the assignment of one of two possible labels, and multiclass classification corresponding to having more than two possible labels.

Such scenarios include *node classification* or instance classification when considering a relational database. The general motivation for exploring relational machine learning revolves around the fact that real-life instances can be in interaction; for example, people

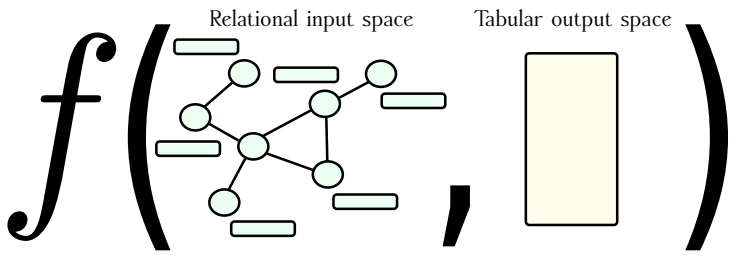


Figure 3.1: An overview of the considered relational learning problem.

in a social network are seldom independent. Similarly, proteins in regulatory networks form *functional clusters*. Developing learning methods capable of directly exploiting such rich relational structure is becoming increasingly relevant, as open knowledge bases comprised of tens of millions of triplets are becoming available for a multitude of practical fields, ranging from biology and medicine to engineering. This chapter includes two main contributions, each focusing on a particularly interesting (open) problem considering relational learning at the boundary between symbolic and sub-symbolic learning. We continue by presenting individual contributions and their implications, each followed by a discussion relating the contribution to the set of hypotheses stated in Section 1.3.

3.2 Propositionalization and Embeddings for Relational Data

The final contribution related to relational learning addresses the issue of representation learning for relational domains from an ILP perspective. Here, we explore the relationship between propositionalization (the process of *symbolic representation learning*) and embedding construction (in this context understood as the process of *sub-symbolic representation learning*). The purpose of this paper is two-fold; first, a comprehensive survey of existing propositionalization and different relational embedding methods is offered. Next, we explore how the existing approach of Wordification (Perovšek et al., 2013) could be linked with an embedding-producing procedure, offering sub-symbolic representations of either features (PropStar), or instances (PropDRM). We demonstrate the competitive performance of the methods and further pinpoint that they scale better (due to minibatch-based learning and sparse matrix algebraic operations used to implement all space-intensive intermediary steps). The paper relevant to this section is the following one:

> Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, *109*(7), 1465–1507. https://doi.org/10.1007/s10994-020-05890-8

3.2.1 Key Contributions

We next present the key contributions of the conducted work.

1. A comprehensive *survey* of relational embedding and propositionalization techniques.

2. Two new approaches to neuro-symbolic relational embeddings; one based on representation learning of relational bags (PropDRM) and the other on representation learning of individual conjuncts (PropStar).

3. Updated and refactored *benchmarking library* PyRDM, which now includes most major baselines from this domain.

4. A discussion of the possibilities for considering neuro-symbolic learning *at scale*.

3.2.2 Addressed Hypotheses and Discussion

As part of this study, we were interested in the following questions. First, we hypothesized that it is possible to summarize the differences between neural and symbolic representation learning for relational data (Hypothesis 1, Section 1.3). Moreover, we hypothesized that by adopting two leading black-box representation learning paradigms alongside the existing ILP-based propositionalization approaches, we could obtain competitive performance but achieve better scalability (Hypothesis 4, Section 1.3).

We confirmed the first hypothesis by demonstrating the key differences between the two representation learning paradigms. We have, however, realized that the two learning paradigms do not differ with respect to only a single aspect (e.g., interpretable vs. non-interpretable features), but with regards to computational and hardware-level aspects too. During this study, one of the main realizations was that it is highly likely that the output representation will be a high-dimensional sparse matrix when considering symbolic representation learning. This observation results from the fact that propositionalization-based approaches do not perform any feature smoothing or similar aggregations which would aggregate and compress parts of the space in order to obtain a lower-dimensional representation. Interestingly, if considering propositionalization-based representations with regard to their actual memory footprint, they are surprisingly *compact* if the correct data structures are used for their storage during learning (e.g., CSR matrices – Column-Separated Rows). Here, we experimented with different sparse matrix formats, identifying CSR as the most suitable one.

The second hypothesis (scalability of neuro-symbolic learners) addresses the issue of linking the existing symbolic representation learners with contemporary relational representation learning approaches, which mostly revolve around *embedding construction*. The rationale for this line of research was that the existing approaches such as the Deep Relational Machines (Lodhi, 2013) demonstrated compatibility between the well known Aleph (Srinivasan, 2001) relational learning tool and contemporary deep neural networks. As part of the addressed hypothesis, we explored whether the DRM-based combination of symbolic with sub-symbolic, is the only conceptual approach to neuro-symbolic relational learning via propositionalization. By incorporating the idea of StarSpace (and similar approaches), we demonstrated that there exists another way of embedding-based learning from propositionalized relational databases. Here, relational features are *embedded* individually based on their co-occurrences determined by the instance space. Subsequent learning is, in this case, different to, e.g., DRM-based approaches; embeddings of features are compared directly to *embeddings of labels*; the closer a given label to a collection of relational feature embeddings, the more likely it is to be selected as the final label. One of the purposes of this paper was also to evaluate to what extent the existing symbolic representation learners *scale* to larger data sets that transcend hundreds of instances. The conducted experiments demonstrate that only Wordification (Perovšek et al., 2013) and the proposed two neuro-symbolic approaches scale to millions of instances. One of the key problems we identified regarding the scaling of other methods is their insufficient handling of sparse matrices, yielding space demanding intermediary processing. This paper, however, rejects the hypothesis that superior performance can be obtained via neuro-symbolic learners – at best, we observed (statistically) competitive performance (with existing state-of-the-art).

Machine Learning (2020) 109:1465–1507
https://doi.org/10.1007/s10994-020-05890-8

Propositionalization and embeddings: two sides of the same coin

Nada Lavrač[1,2] · Blaž Škrlj[3] · Marko Robnik-Šikonja[4]

Received: 15 February 2019 / Revised: 29 May 2020 / Accepted: 8 June 2020 / Published online: 28 June 2020

Abstract

Data preprocessing is an important component of machine learning pipelines, which requires ample time and resources. An integral part of preprocessing is data transformation into the format required by a given learning algorithm. This paper outlines some of the modern data processing techniques used in relational learning that enable data fusion from different input data types and formats into a single table data representation, focusing on the propositionalization and embedding data transformation approaches. While both approaches aim at transforming data into tabular data format, they use different terminology and task definitions, are perceived to address different goals, and are used in different contexts. This paper contributes a unifying framework that allows for improved understanding of these two data transformation techniques by presenting their unified definitions, and by explaining the similarities and differences between the two approaches as variants of a unified complex data transformation task. In addition to the unifying framework, the novelty of this paper is a unifying methodology combining propositionalization and embeddings, which benefits from the advantages of both in solving complex data transformation and learning tasks. We present two efficient implementations of the unifying methodology: an instance-based PropDRM approach, and a feature-based PropStar approach to data transformation and learning, together with their empirical evaluation on several relational problems. The results show that the new algorithms can outperform existing relational learners and can solve much larger problems.

Keywords Inductive logic programming · Relational learning · Propositionalization · Embeddings · Knowledge graphs

Editors: Dimitar Kazakov and Filip Železny.

✉ Blaž Škrlj
blaz.skrlj@ijs.si

Extended author information available on the last page of the article

1 Introduction

Data preprocessing for machine learning is a great challenge for a data scientist faced with large quantities of data in different forms and sizes. Most of the modern data processing techniques enable data fusion from different data types and formats into a single table data representation, which is expected by standard machine learning techniques including rule learning, decision tree learning, support vector machines (SVMs), deep neural networks (DNNs), etc. The key element of the success of modern data transformation methods is that similarities of original instances and their relations are encoded as distances in the target vector space.

Two of the most prominent data transformation approaches outlined in this paper are propositionalization and embeddings. While propositionalization (Kramer et al. 2001; Železný and Lavrač 2006) is a well known data transformation technique used in relational learning (RL) and inductive logic programming (ILP) (Muggleton 1992; Lavrač and Džeroski 1994; De Raedt 2008), embeddings (Mikolov et al. 2013; Wu et al. 2018) have only recently been recognized by RL and ILP researchers as a powerful technique for preprocessing relational and complex structured data. In the relational learning context of this paper, both approaches take as input a relational data set (e.g., a given relational database) and transform it into a single data table format, which is then used as an input to a propositional learning algorithm of choice.

The first aim of this paper is to present a unifying survey of propositionalization and embedding data transformation approaches. While both approaches aim at transforming data into a tabular data format, the approaches use different terminology and task definitions, claim to have different goals, and are used in very different contexts. This paper contributes an improved understanding of these data transformation techniques by presenting a unified terminology and definitions, by explaining the similarities and differences of the two definitions as variants of a unified complex data transformation task, by exploring the apparent differences between the two approaches, and by outlining some of their advantages and disadvantages.

In addition to the unifying survey, the main novelty of this paper is a unifying methodology that combines propositionalization and embeddings, which benefits from the advantages of both in solving complex data transformation and learning tasks. The unifying methodology resulted in two new pipelines, PropDRM and PropStar, which implement an instance-based and a feature-based approach to data transformation and learning, respectively. Both approaches are computationally efficient and can successfully solve much larger tasks than the existing relational learning approaches. We made their code publicly available.

The paper starts by motivating the need for transforming heterogeneous relational data into a tabular format in Sect. 2. Section 3 introduces the data transformation approaches in the context of information representation levels proposed by Gärdenfors (2000). Section 4 presents the related work, focusing on selected propositionalization and embeddings methods relevant to the relational learning context of this paper. Section 5 presents a unifying framework for propositionalization and embeddings, allowing for the analysis of characteristic properties of these data transformation approaches. Section 6 proposes a unifying methodology that combines propositionalization and embeddings, which benefits from the advantages of both, and presents two implementations of the proposed unifying framework: an instance-based embedding approach PropDRM based on the existing Deep Relational Machines (DRM) (Srinivasan et al. 2019; Lodhi 2013), followed by a novel feature-based

embedding approach PropStar proposed in this paper, using the StarSpace entity embedding approach (Wu et al. 2018). Experimental evaluation of the proposed implementations is presented in Sect. 7. The paper concludes by a summary and some ideas for future work in Sect. 8.

2 Motivation

Machine learning is the key enabler for computer systems to progressively improve their performance when helping humans to solve difficult problem solving tasks. Nevertheless, current machine learning approaches only come half-way in helping humans, as humans still have to formulate the problem and prepare the data in the form that is best suited to the powerful machine learning algorithms.

Most of the best performing machine learning algorithms, like Support Vector Machines (SVMs) or deep neural networks, assume numeric data and outperform symbolic approaches in terms of predictive performance, efficiency, and scalability. The dominance of numeric algorithms started in 1980s with the advent of backpropagation and neural networks (Rumelhart et al. 1986), continued in late 1990s and early 2000s with SVMs (Cortes and Vapnik 1995), and finally reached the current peak with deep neural networks (Goodfellow et al. 2016). Deep neural networks are currently considered the most powerful learners for solving many of previously unsolvable learning problems in computer vision (face recognition rivals humans' performance), game playing (a program has beaten a human champion in the game of Go), and natural language processing (successful automatic speech recognition and machine translation).

While the most powerful machine learning approaches are numeric, humans perceive and describe real-world problems mostly in symbolic terms, using various data representation format, such as graphs, relations, texts or electronic health records, all involving discrete representations. However, if we are to harness the power of successful numeric deep learning approaches for discrete learning problems, discrete data should be transformed into a form suitable for numeric learning algorithms. The viewpoint of addressing real-world problems as numeric has a rationale even for discrete domains, as many symbolic learners perform generalizations based on object similarity. For example, in graphs, nodes can represent similar entities or have connections with similar other nodes; in text, words can appear with similar contexts or play the same role in sentences; in medicine, patients may have similar symptoms or similar disease histories. Such similarities are used by numerous machine learning algorithms to generalize and learn, including classical bottom-up learning approaches such as hierarchical clustering, as well as symbolic learners adapted to top-down induction of clustering trees (Blockeel et al. 1998). If we want to exploit the power of modern machine learning algorithms, like SVMs and deep neural networks, to process the inherently discrete data, one has to transform discrete data into (numeric) vectors in such a way that similarities between objects are preserved and encoded as distances in the transformed (numeric) space.

Contemporary preprocessing approaches that prepare numeric vector data for machine learning algorithms are called *embeddings*. Nevertheless, as demonstrated in this paper, symbolic data transformations, as ancestors of the contemporary embedding approaches, remain relevant: the role of *propositionalization*, a symbolic approach to relational data transformation into feature vectors, is not only to enable contemporary machine

learning algorithms to induce better predictive models, but to allow descriptive data mining approaches to discover interesting human-comprehensible patterns in symbolic data.

As this paper demonstrates, albeit propositionalization and embeddings represent different types of data transformations, these approaches actually represent the *two sides of the same coin*. The main unifying element they have in common is that they transform the data into a vector format and encode the relations between objects in the original space as distances in the new vector space.

3 Data transformations and information representation levels

As this section will show, we consider data transformations as a particular subprocess of data preprocessing. Data preprocessing aims to handle missing attribute values, control out-of-range values and impossible attribute-value combinations, or handle noisy or unreliable data, to name just some of the types of data irregularities addressed in processing real-life data. Data preprocessing may include data cleaning, instance selection, normalization, feature engineering (feature extraction and/or feature construction), data transformation, feature selection, etc. The result of data preprocessing is the final training set, which is used as input to a machine learning algorithm.

Data preprocessing can be manual, automated, or semi-automated. We focus on automated transformations of data, present in heterogeneous types and formats, into a uniform tabular data representation. We refer to this specific automated data preprocessing task as *data transformation*, and define it as follows.

Definition 1 (*Data transformation*) *Data transformation* is a step in the data preprocessing task that automatically transforms the input data and the background knowledge into a uniform tabular representation, where each row represents a data instance, and each column represents one of the dimensions in a multi-dimensional feature space.

In the above definition, we decided to distinguish between *data* and *background knowledge*. This is an intentional decision, although it could be argued that in some settings, we could refer to both as data. Let us provide an operational distinction between data and background knowledge. *Data* is considered by the learner as the target data from which the learner should learn a model (e.g., a classifier in the case of class labeled data) or a set of descriptive patterns (e.g., a set of association rules in the case of unlabeled data). *Background knowledge* is any additional knowledge used by the learner in model or pattern construction from the target data. Simplest forms of background knowledge define hierarchies of features (attribute values), such as color *green* being more general than *light green* or *dark green*. More complex background knowledge refers to any other declarative prior domain knowledge, such as knowledge encoded in relational databases, knowledge graphs or domain specific taxonomies and ontologies, such as the Gene Ontology, in its 2020-05-02 release including 44,508 GO terms, 7,765,270 annotations, 1,464,358 gene products and 4,593 species.

This data transformation setting is applicable in various data science scenarios involving relational data mining, inductive logic programming, text mining, graph and network mining as well as tasks that require fusion of data of a variety of data types and formats and their transformation into a joint data representation formalism.

3.1 Information representation levels

As currently the most powerful machine learning (ML) algorithms take as input numeric representations, users of ML algorithms tend to transform other forms of human knowledge into the numeric representation space. Interestingly, even if this is countering a standard RL and ILP viewpoint, this is true also for symbolic representations, which are currently used to store most of the human knowledge.

The distinction between the symbolic and numeric representation space mentioned above can be further clarified in terms of the *levels of cognitive representations*, introduced by Gärdenfors (2000), i.e. the neural, spatial and symbolic representation levels. In his theory, Gärdenfors assumes that when modeling cognitive systems in terms of information processing, all three levels are connected: starting from the sensory inputs at the lowest neural representation level, resulting in spatial representations at the middle conceptual spaces level, up to symbolic representations at the level of language.

Neural This representation level corresponds to the sub-conceptual connectionist level. At this level, information is represented by activation patterns in densely connected networks of primitive units. This enables concepts to be learned from the observed data by modifying the connection weights between the units.

Spatial This representation level is modeled in terms of Gärdenfors' conceptual spaces. At this level, information is represented by points or regions in a conceptual space built upon some dimensions that represent geometrical, topological or ordinal properties of the observed objects. In spatial representations, the similarity between concepts is represented in terms of the distances between the points or regions in a multidimensional space, where concepts are learned by modeling the similarity between the observed objects.

Symbolic At this representation level, information is represented by the language of symbols (words), where the meaning is internal to the representation itself (i.e. symbols have meaning only in terms of other symbols, while their semantics is grounded in the spatial level), and concepts are learned by symbolic generalization rules.

From the perspective of this paper, the above levels of cognitive representations introduced by Gärdenfors (2000) provide a theoretical ground to separate the learning approaches as well as the data transformation approaches into three categories based on the levels of their output representation space: neural, spacial and symbolic. However, given the scope of this paper, we do not consider *neural transformations*, and focus only on two data transformation types:

- *symbolic transformations*, in this paper referred to as *propositionalization*, denoting data transformations into a symbolic representation space, and
- *numeric transformations*, in this paper referred to as *embeddings*, denoting data transformations into a spatial representation space.

These two data transformation approaches are briefly introduced below, and further described in the related work (Sect. 4).

3.2 Transformations into symbolic representation space

The past decades of machine learning were characterized by symbolic learning, where the result of a machine learning or data mining algorithm was a predictive model of a set of patterns described in a symbolic representation language, resulting in symbolic human-understandable patterns and models. Symbolic machine learning approaches include rule learning (Michalski et al. 1986; Clark and Niblett 1989), decision tree learning (Quinlan 1986) and learning logical representations by relational learning and inductive logic programming (ILP) algorithms (Muggleton 1992; Lavrač and Džeroski 1994; De Raedt 2008).

To be able to apply a symbolic learner, the data is typically transformed into a single tabular data format, where each row represents a single data instance, and each column represents an attribute or a feature. Such transformation into symbolic vector space (i.e. a symbolic data table format) is well known in the ILP and relational learning community, where it is referred to as *propositionalization*. Propositionalization approaches are presented in Sect. 4.2.

3.3 Transformations into numeric representation space

In the last 20 years we have been witnessing increasing dominance of statistical machine learning and pattern-recognition methods, including neural network learning (Rumelhart and McClelland 1986), Support Vector Machines (SVMs) (Vapnik 1995; Schölkopf and Smola 2001), random forests (Breiman 2001), and boosting (Freund and Schapire 1997). These statistical approaches are quite different from the symbolic approaches mentioned in Sect. 3.2, however there are many approaches that cross these boundaries, including e.g., the CART decision tree learning algorithm (Breiman et al. 1984), the Bump hunting rule learning algorithm (Friedman and Fisher 1999), which are firmly based in statistics. Moreover, ensemble techniques such as boosting (Freund and Schapire 1997), bagging (Breiman 1996) or random forests (Breiman 2001) also combine the predictions of multiple logical models on a sound statistical basis (Schapire et al. 1998; Mease and Wyner 2008; Bennett et al. 2008). All these are also considered to belong to the family of statistical learning approaches.

To be able to apply a statistical learner, the data is typically transformed into a single tabular data format, where each row represents a single data instance, and each column is a numeric attribute or a numeric feature, with some predefined range of numeric values. Such transformation into numeric vector space (i.e. a numeric data table format) is well known in the deep learning community, where it is referred to as *embedding*. Approaches to embedding relational structures are presented in Sect. 4.3.

4 Related work

In this section we first outline various transformation methods in Sect. 4.1, followed by a more detailed description of the data transformation methods relevant for the context of relational learning, i.e. propositionalization and embeddings, in Sects. 4.2 and 4.3, respectively.

4.1 Outline of data transformation methods

While there are many algorithms for transforming data into a spatial representation, it is interesting that recent approaches rely on deep neural networks, thereby harnessing the neural representation level as the means to transform symbolic representations into the spatial representation. Below we list the main types of approaches that perform transformations between representations.

Community detection and graph traversal methods. Many complex data sets can be represented as graphs, where nodes represent data instances and edges represent their relations. Graphs can be homogeneous (consisting of a single type of nodes and relations) or heterogeneous (consisting of different types of nodes and relations). To encode a graph in a tabular form by preserving the information about the relations, various graph encoding techniques were developed, such as propositionalization via random walk graph traversal, representing nodes via their neighborhoods and communities (Plantié and Crampes 2013). These approaches are frequently used for data fusion in mining heterogeneous information networks. Neural network approaches (presented below) are also very competitive as means for encoding graphs.

Matrix factorization methods. When data is not explicitly presented in the form of relations but the relations between objects are implicit, given by a similarity matrix, the objects can be encoded in a numeric form using matrix factorization. As an example take Latent Semantic Analysis used in text mining, which factorizes a word similarity matrix to represent words in a vector form. Another example is factorization of graph adjacency matrices. These types of embeddings were largely superseded by deep neural networks which, instead of observing similarity between different objects, construct a prediction task and forecast similarity. For example, for text, given a word, the word-2vec embedding method (Mikolov et al. 2013) predicts words in its neighborhood.

Propositionalization methods are used to get tabular data from multirelational databases as well as from a mixture of tabular data and background knowledge in the form of logic programs or networked data, including ontologies. These transformations were mostly developed within the Inductive Logic Programming and Relational Learning community, and are still actively researched and used. Propositionalisation methods do not perform dimensionality reduction and are most often used with data mining and symbolic machine learning algorithms. We discuss these methods in Sect. 4.2.

Neural networks based methods. In neural networks the information is represented by activation patterns in interconnected networks of primitive units. This enables that concepts are gradually learned from the observed data by modifying the connection weights between the hierarchically organized units. These weights can be extracted from neural networks and used as a spatial representation that transforms relations between entities into distances. Recently, this approach became a prevalent way to build representation for many different types of entities, e.g., texts, graphs, electronic health records, images, relations, recommendations, etc. In Sect. 4.3 we describe the data types and approaches, which are capable of embedding relational structures and are therefore most relevant for the context of this paper. These include knowledge graph embeddings (presented in Sect. 4.3.1), entity embeddings capable of forming (both supervised and unsupervised) representations based on the similarity of entities (presented in Sect. 4.3.2), and Deep Relational Machines methodology that links symbolic representations to deep neural networks (presented in Sect. 4.3.3).

Other embedding methods. Other forms of embeddings were developed by different communities that observed the need to better represent the (symbolic) data. For example, Latent Dirichlet Allocation (LDA) (Blei et al. 2003) used in text analysis learns distributions of words for different topics. These distributions can be used as an effective embedding for words, topics, and documents. Feature extraction methods form a rich representation of instances by projecting them into a high dimensional space (Lewis 1992). Another example of (implicit) transformation into high dimensional space is the kernel convolutional approach proposed by Haussler (1999), which introduces the idea that kernels can be used for discrete structures by iteratively applying convolution and kernels to smaller parts of the data structure. Convolutional kernels exist for sets, graphs, trees, strings, logical interpretations, and relations (Cumby and Roth 2003). This allows methods such as SVM or Gaussian Processes to work with relational data. Most of these embeddings are recently superseded or merged with neural networks.

All the above approaches perform data transformations from different data formats to a single table representation. However, their underlying principles are different: while factorization and neural embeddings perform dimensionality reduction, resulting in lower-dimensional feature vector representations capturing the semantics of the data, propositionalization results in a vector representation using relational features with a higher generalization potential than the features used in the original data representation. Note that there exist also other approaches to data transformation and fusion, including HINMINE (Kralj et al. 2018), metapath2vec (Zhu et al. 2018) and OhmNet (Žitnik and Leskovec 2017), which are out of the main scope of this paper.

4.2 Propositionalization

In propositionalization, relational feature construction is the most common approach to data transformation. LINUS (Lavrač et al. 1991) was one of the pioneering propositionalization approaches using automated relational feature construction. LINUS was restricted to generation of features that do not allow recursion and existential local variables, which means that the target relation cannot be many-to-many and self-referencing. The second limitation was more serious: the queries could not contain joins (conjunctions of literals). The LINUS descendant SINUS (Lavrač and Flach 2001) incorporates more advanced feature construction techniques inspired by 1BC (Flach and Lachiche 1999). The LINUS approach had many followers, including relational subgroup discovery system RSD (Železný and Lavrač 2006), which is outlined also in the list of propositionalization approaches below. Alternatives to relational feature construction include the construction of aggregation queries.

In this section we first clearly define the distinction between attributes and features, followed by an outline of selected propositionalization approaches and of the specific Wordification approach used in the algorithms developed in this work.

4.2.1 Features

To be able to apply a symbolic propositional learner, the data should be represented in a single table data format, where each row represents a single data instance, and each column represents an attribute or a feature. For the sake of clarity, let us distinguish between *attributes* and *features* below.

Attributes that describe the data instances can be either numeric variables (with values like 7 or 1.5) or nominal/discrete variables (with values like *red* or *female*). In contrast to attributes, a *feature* describes the presence or absence of some property of an instance. As a result, features are always Boolean-valued (values *true* or *false*). For example, for attribute *gender* with values *female* and *male*, two separate features can be constructed: f_1: *gender=female* and f_2: *gender=male*, and only one of these features is assumed to be *true* for an individual data instance. Note that features are different even from binary-valued attributes: e.g., for a binary attribute a_i with values *true* and *false*, there are two corresponding features: f_3: $a_i = true$ and f_4: $a_i = false$. Furthermore, features can test a value of a single attribute, like $a_j > 3$, or they can represent complex logical and numerical relations, integrating properties of multiple attributes, like f_5: $a_k < 2 \cdot (a_j - a_i)$.

Previous feature types are referred to as *propositional features*. On the other hand, *relational features* relate the values of different attributes to each other. In the simplest case, for example, they test for the equality or inequality of the values of two attributes of the same type, such as *Length* and *Height*. More complex relational features can use the background relations, e.g., f_6: *adjacent(NodeX, NodeY)*. Even more advanced, relational features can introduce new variables. For example, if relations are used to encode a graph, a relational feature such as f_7: *color(CurrentNode, blue)* $\wedge$ *link(CurrentNode, NewNode)* $\wedge$ *color(NewNode, red)*, can introduce a new variable *NewNode* to subsequently test whether there exists a previously not visited node in the graph that is colored red.

Take a simple toy trains example learning problem illustrated in Appendix A, and two complex relational features describing trains:

f_8: hasCar(T,C) $\wedge$ carLength(C,short) $\wedge$ carRoof(C,peaked)

f_9: hasCar(T,C1) $\wedge$ carLength(C1,short) $\wedge$ hasCar(T,C2) $\wedge$ carRoof(C2,peaked)

Feature f_8 is a single complex relational feature, while f_9 contains two distinct relational features. Formally, a feature is defined as a minimal set of literals such that it introduces at most one local (i.e. existential) variable in the feature set composing the relational feature.

The main point of relational features is that they localize variable sharing: this can be made explicit by naming the features:

f_{10}: hasShortCar(T) $\leftarrow$ hasCar(T,C) $\wedge$ clength(C,short)

f_{11}: hasPeakedroofCar(T) $\leftarrow$ hasCar(T,C) $\wedge$ carRoof(C,peaked)

The propositionalization approach to relational learning captures exactly this idea: generating complex features, such as f_8, f_{10} and f_{11}, which will allow multi-relational data representation of properties of target instances (such as trains *T*) through representations of properties of their components (such as cars *C*). Selected propositionalization approaches, which use complex feature construction in the automated multi-relational data transformation process are outlined below.

4.2.2 Outline of selected propositionalization algorithms

Below we outline a selection of propositionalization approaches, while an interested reader can find extensive overviews of different feature construction approaches in the work of Kramer et al. (2001) and Krogel et al. (2003).

Relaggs (Krogel and Wrobel 2001) stands for *rel*ational *agg*regation. It is a propositionalization approach that takes the input relational database schema as a basis for a declarative bias, using optimization techniques usually used in relational databases (e.g.,

indexes). The approach employs aggregation functions in order to summarize non-target relations with respect to the individuals in the target table.

1BC (Flach and Lachiche 1999) strives to enable the propositional naive Bayes classifier to handle relational data. It does so by a transformation in which a set of first-order conditions is generated and then used as attributes in the naive Bayes classifier. The transformation, however, is done in a dynamic manner, as opposed to standard propositionalization, which is performed as a static step of data preprocessing. This approach is extended by 1BC2 (Lachiche and Flach 2003), which allows distributions over sets, tuples, and multisets, thus enabling the naive Bayes classifier to consider also structured individuals.

Tertius (Flach and Lachiche 2001) is a top-down rule discovery system, incorporating first-order clausal logic. The main idea is that no particular prediction target is specified beforehand, hence Tertius can be seen as an ILP system that learns rules in an unsupervised manner. Its relevance for this survey lies in the fact that Tertius encompasses 1BC, i.e. relational data is handled through 1BC transformation.

RSD (Železný and Lavrač 2006) is a relational subgroup discovery algorithm composed of two main steps: the propositionalization step and the (optional) subgroup discovery step. The output of the propositionalization step can be used also as input to other propositional learners. RSD effectively produces an exhaustive list of first-order features that comply with the user-defined mode constraints, similar to those of Progol (Muggleton 1995) and Aleph (Srinivasan 2007). Furthermore, RSD features satisfy the connectivity requirement, which imposes that no feature can be decomposed into a conjunction of two or more features. Mode declarations define the algorithm's syntactic bias, i.e. the space of possible features.

HiFi (Kuželka and Železný 2008) is a propositionalization approach that constructs first-order features with hierarchical structure. Due to this feature property, the algorithm performs the transformation in polynomial time of the maximum feature length. Furthermore, the resulting features are the shortest in their semantic equivalence class. The algorithm is shown to perform several orders of magnitude faster than RSD for higher feature lengths.

RelF (Kuželka and Železný 2011) is the most relevant of the algorithms in the Tree-Liker software (Kuželka and Železný 2011). It constructs a set of tree-like relational features by combining smaller conjunctive blocks. RelF preserves the monotonicity of feature reducibility and redundancy (instead of the typical monotonicity of frequency), which allows the algorithm to scale far better than other state-of-the-art propositionalization algorithms.

Cardinalization (Ahmed et al. 2015) is specifically designed to enable more than just categorical attributes in propositionalization. Specifically, it can handle a threshold on numeric attribute values and a threshold on the number of objects satisfying the condition on the attribute simultaneously. Cardinalization can be seen as an implicit form of discretization. While in discretization one sets a threshold on a numeric attribute and see how many objects satisfy the threshold later, and the cardinality follows implicitly from the attribute value threshold; on the other hand, in cardinalization, we set a threshold on the cardinality, and let an attribute-value learner decide where the threshold value on the numerical attribute should lie. Hence, Cardinalization allows for context-aware discretization. Quantiles (Ahmed et al. 2015) is a variant of Cardinalization. Instead of choosing an absolute number as cardinality threshold, Quantiles uses a relative number.

CARAF (Charnay et al. 2015) approaches the problem of large relational feature search space by aggregating base features into complex compounds, which makes CARAF

similar to Relaggs. Complex aggregates run the risk of overfitting. While Relaggs tackles this problem by restricting itself to relatively simple aggregates, the distinguishing feature of CARAF is that instead it incorporates more complex aggregates into a random forest, which ameliorates the overfitting effect.

Aleph (Srinivasan 2007) is the most popular ILP algorithm and is actually an ILP toolkit with many modes of functionality: learning of theories, feature construction, incremental learning, etc. Aleph uses mode declarations to define the syntactic bias. Input relations are Prolog clauses, defined either extensionally or intensionally. Aleph's feature construction functionality also means it is a propositionalization approach.

Wordification (Perovšek et al. 2013, 2015) is a propositionalization method inspired by text mining that can be viewed as a transformation of a relational database into a corpus of text documents. The distinguishing property of Wordification is its efficiency when used on large relational data sets and the potential for using text mining approaches on the transformed propositional data. While most of the outlined propositionalization algorithms construct complex relational features including variables in the arguments of relational features, Wordification constructs simple, easily interpretable features that are treated as 'words' in the transformed Bag-Of-Words representation. It constructs features of the kind $a_i = v_{ij}$ (formulated as $a_i\_v_{ij}$). In addition to such simple features, it constructs also conjuncts (of size 2) of such features, e.g., $a_i = v_{ij} \wedge a_k = v_{kl}$, formulated as $a_i\_v_{ij}\_\_a_k\_v_{kl}$. To avoid confusion in case the same attribute name appeared in several tables, the actual form of features is $t\_a_i\_v_{ij}$ including the indicator of the name of table t in which attribute a_i appears. For a simple example of how such features are generated, the reader is referred to Appendix A.

4.2.3 Wordification

Given that in a previous experimental evaluation of propositionalization algorithms (Perovšek et al. 2013, 2015) the Wordification algorithm was shown to be the most effective, we selected Wordification as the propositionalization algorithm of choice in the proposed implementations combining propositionalization and embeddings in Sect. 6, where the Wordification algorithm was adapted to handle large data sets.

In the Wordification implementation, described in detail in Sect. 6.2.1, the original feature representation *TableName_AttributeName_AttributeValue* was—for implementational convenience—replaced by a tuple representation (*t.name, c, v*), where *t.name* refers to a table name, *c* to a given colon (attribute) in the table *t*, and *v* to a given value *v* of attribute *c*. Such features will be referred to as *features* or as *relational items* in the algorithm description, as appropriate.

Using this feature representation, Wordification of a multi-relational database can be summarized as the following operation:

$$\mathrm{DB}_i = \biguplus_{t \in \mathcal{T}} \mathrm{WORDIFY}(t(m(i)))$$

where m maps a given table t's indices to target (initial) table indices (i) and $\mathcal{T}$ is the set of all tables from which a foreign key path exists to the target table. The $\uplus$ operator represents a disjoint union of multisets (sum), yielding a single multiset (duplicates are allowed).

Foreign keys are designated columns that link data between distinct tables. Value of a foreign key in a given table is referred to as the instance id (the row is uniquely determined by this value). Let C represent the set of all columns that are not foreign keys, ids or target

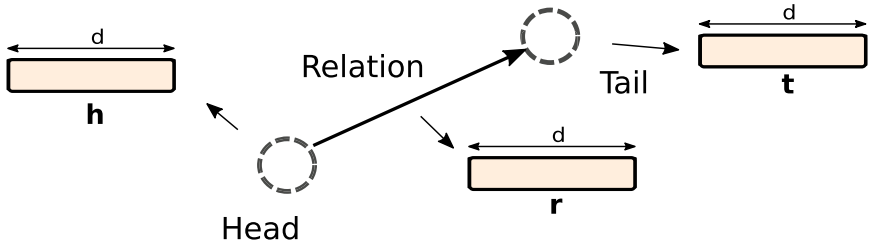


Fig. 1 Schematic representation of knowledge graph embedding. Head-Relation-Tail (h, r, t) triplets are used as inputs. Triplets are embedded in a common d-dimensional vector space

classes. The WORDIFY method returns a multiset (a bag) of relational items (for the i-th instance) constructed as follows:

$$\text{WORDIFY}(t(m(i))) = \biguplus_{v \in t[m(i)][c \in C]} (t.name, c, v)$$

where $t[c]$ represents the values v of table t in column c, and t.name is the name of table t. Thus, Wordification is naïve in the sense that it simply concatenates attribute values across tables by maintaining the column and table name information in constructing features. The original implementation, however, can become spatially intractable (see (Perovšek et al. 2013), proof of complexity) as its spatial complexity is $\mathcal{O}(\text{row} \cdot \text{tables} \cdot 2^{\text{col}})$. Details of a more efficient implementation of Wordification are available in Sect. 6.2.1.

4.3 Embedding relational structures

In this section, we discuss methodologies capable of embedding relational structures. We start with an introduction to knowledge graph embeddings, an emerging group of methods that operate on large, real-world, annotated graphs, in Sect. 4.3.1. We proceed by the presentation of entity embeddings, a more general methodology capable of supervised, as well as unsupervised embeddings of many entities, including texts and knowledge graphs in Sect. 4.3.2. Finally, in Sect. 4.3.3, we present Deep Relational Machines, an emerging methodology that links symbolic representations to deep neural networks.

4.3.1 Knowledge graph embeddings

In knowledge graphs (KG), edges correspond to relations between entities (nodes) and the graphs present Subject-Predicate-Object *triplets*. The KG handling algorithms attempt to solve the problems like triplet completion, relation extraction, and entity resolution. The KG embedding algorithms, briefly discussed below, outline some of the key ideas which render these methods highly scalable and useful for large, semantics-rich graphs. For detailed description and a recent, extensive overview of the field, we refer the reader to Wang et al. (2017), from where we next summarize some of the key ideas underlying knowledge graph embedding.

In the below description of KG embedding algorithms, the Subject-Predicate-Object triplet notation is replaced by the (h, r, t) triplet notation, where h is referred to as the *head* of a triplet, t as the *tail*, and r as the *relation* connecting the head and the tail. A schematic representation of triplet embedding is shown in Fig. 1. The embedding

methods briefly outlined below optimize the total plausibility of the input set of triplets, where plausibility of a single triplet is denoted with $f_r(h, t)$.

- The first group of KG embedding algorithms are termed *translational distance models*, as they exploit distance-based scoring functions. They measure the plausibility of a fact as the distance between the two entities, usually after a translation carried out by the relation. One of the representative methods for this type of embedding is transE (Bordes et al. 2013), where the cost function being optimized can be stated as:

$$f_r(h, t) = -||\mathbf{h} + \mathbf{r} - \mathbf{t}||^2.$$

  For vectors $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ in the obtained embedding, score $f_r(h, t)$ is high if triplet (h, r, t) is present in the data.
- The second group of KG embedding algorithms is not deterministic, as it takes into account the uncertainty of observing a given triplet. A representative method for this type of embeddings is KG2E (He et al. 2015), which models the triplets with multivariate Gaussians. It models individual entities, as well as relations as vectors, drawn from multivariate Gaussians, assuming that $\mathbf{h}$, $\mathbf{r}$ and $\mathbf{t}$ vectors are normally distributed, with mean vectors $\mu_h, \mu_r, \mu_t \in \mathbb{R}^d$ and covariance matrices $\Sigma_h, \Sigma_r, \Sigma_t \in \mathbb{R}^{d\times d}$, respectively. KG2E uses Kullback-Liebler divergence to directly compare the distributions as follows:

$$\begin{aligned} f_r(h, t) &= \mathrm{KL}(\mathcal{N}(\mu_t - \mu_h), \mathcal{N}(\mu_r)) \\ &= \int \mathcal{N}_x(\mu_t - \mu_h, \Sigma_t + \Sigma_h) \ln \frac{\mathcal{N}_x(\mu_t - \mu_h, \Sigma_t + \Sigma_h)}{\mathcal{N}_x(\mu_r, \Sigma_r)} dx, \end{aligned}$$

  where $\mathcal{N}_x$ denotes the probability density function of the normal distribution.
- Semantic matching models exploit similarity-based scoring functions. They measure plausibility of facts by matching latent semantics of entities and relations embodied in their vector space representations. One of the representative algorithms for learning by semantic matching is RESCAL (Nickel et al. 2011). RESCAL optimizes the following expression:

$$f_r(h, t) = \mathbf{r}^T \cdot M_r \cdot \mathbf{t},$$

  where $\mathbf{h}$ and $\mathbf{t}$ are representations of entities, and $M_r \in \mathbb{R}^{d\times d}$ is a matrix associated with relations.
- Matching using neural networks. Deep neural networks model triplets via training of neural network architectures. One of the first approaches was Semantic Matching Energy (SME) (Bordes et al. 2014). This method first projects entities and their relations to their corresponding vector embeddings. The relation's representation is next combined with the relation's head and tail entities to obtain $g_1(\mathbf{h}, \mathbf{r})$ and $g_2(\mathbf{t}, \mathbf{r})$ entity-relation representations in the hidden layer. Finally, a dot product is used to score the triplet relation matching

$$f_r(h, t) = g_1(\mathbf{h}, \mathbf{r})^T \cdot g_2(\mathbf{t}, \mathbf{r}).$$

  The simplest version of SME defines the g_1 and g_2 as:

$$g_1(\mathbf{h}, \mathbf{r}) = W_1^{(1)} \cdot \mathbf{h} + W_1^{(2)} \cdot \mathbf{r} + b_1$$
$$g_2(\mathbf{t}, \mathbf{r}) = W_2^{(1)} \cdot \mathbf{t} + W_2^{(2)} \cdot \mathbf{r} + b_2.$$

Here, $W_1^{(1)}, W_1^{(2)}, W_2^{(1)}$ and $W_2^{(2)}$ are $\mathbb{R}^{d \times d}$ dimensional weight matrices and b_1 and b_2 are bias vectors.

Recent advances in embeddings of knowledge graphs show interesting research directions. For example, hyperbolic geometry could be used to better capture latent hierarchies, commonly present in real-world graphs (Nickel and Kiela 2017). Further, KG embedding methods are increasingly tested on large, multi-topic data collections, for example, the Linked Data (LD) which standardize and fuse data from different resources. Knowledge graph embeddings, such as RDF2vec (Ristoski and Paulheim 2016) attempt to exploit vast amounts of information in LD and transform it into a learning-suitable format. As knowledge graphs are not necessarily the only source of available information, algorithms exploit also other information, e.g., textual information available for each triplet (Wang et al. 2014). Recent trends in knowledge graph embeddings also explore how symbolic, logical structures could be used during embedding construction. Approaches such as KALE (Guo et al. 2016) construct embeddings by taking into account logical rules (e.g., Horn clauses) related to the knowledge graph, thus increasing the quality of embeddings. Similar work was proposed by Rocktäschel et al. (2015), where pairs of embeddings were considered during optimization. The same group also showed how relations can be modeled without grounding the head and tail entities for simple implication-like clauses (Demeester et al. 2016). Wang et al. (2015) demonstrated that logical rules can aid in knowledge graph completion on large knowledge bases. They showed that inclusion of rules can reduce the solution space and significantly improve the inference accuracy of embedding models.

4.3.2 Entity embedding with the StarSpace approach

The guiding principle behind all embeddings, described in the previous section, is the persistence of similarity, i.e. that entities which are similar in the knowledge graph must be represented by vectors that are similar in the embedding space. A general approach implementing this principle is to use any similarity function between entities to form a prediction task for a neural network. Below we describe a successful example of this approach, called StarSpace (Wu et al. 2018). As this approach assumes discrete features from a fixed dictionary, it is particularly appealing to relational learning and inductive logic programming.

The idea of StarSpace is to form a prediction task where a neural network is trained to predict the similarity between an entity and its related entity (e.g., its label or some other entity). The resulting neural network can be used for several purposes: directly in classification, to rank instances by their similarity, or weights of the trained network can be used as pretrained embeddings.

In StarSpace, each entity has to be described by a set of discrete features from a fixed-length dictionary and forms a so called Bag-Of-Features. This representation is general enough to cover texts (documents or sentences can be described by bags-of-words or bags-of-n-grams), users (described by bags of documents, movies, or items they like), relations and links in graphs (described by semantic triples), etc. During training, entities of different kinds are embedded *in the same* latent space, suitable for various down-stream learning tasks, e.g., a user can be compared with the recommended items. Note that entities can be embedded along with target classes, resulting in *supervised embedding learning*. This type

of representation learning is the key element of the proposed PropStar algorithm outlined in Sect. 6.1.2 and presented in detail in Sect. 6.2.3.

The StarSpace approach trains a neural network model to predict which pairs of entities are similar and which are dissimilar. Two kinds of training instances are formed, positive $(a, b) \in E^+$, which are task dependent and contain correct relations between entities (e.g., document a with its correct label b), and negative instances $(a, b_1^-), \ldots, (a, b_k^-) \in E_a^-$. For each entity a (e.g., a document) appearing in the positive instances, negative instances are formed using k-negative sampling from labels $\{b_i^-\}_{i=1}^k$ as in word2vec (Mikolov et al. 2013). In each batch, the neural network tries to minimize the loss function L, defined as follows:

$$\mathrm{L} = \sum_{(a,b)\in E^+} \left(\mathrm{Loss}(\mathrm{sim}(a,b)) + \frac{1}{k} \sum_{\substack{i=1 \\ (a,b_i^-)\in E_a^-}}^{k} \mathrm{Loss}(\mathrm{sim}(a,b_i^-)) \right)$$

For each batch update in the training of neural network, k negative examples (a parameter) are formed by randomly sampling labels b_i^- from within the set of entities that can appear in b. For example, in the document classification task, document a has its correct label b, while k negative instances have their labels b_i^- sampled from the set of all possible labels. Similarity function *sim* represents the similarity between the vector representations of the two entities; typically a dot product similarity is used. Within one batch, loss function Loss sums the losses of the positive instance (a, b) and the average of the k negative instances $(a, b_i^-), i \in 1 \ldots k$. To asses the loss, margin ranking loss is used, $\mathrm{Loss} = \max(0, m - \mathrm{sim}(a, b'))$, where m is the margin parameter, i.e. the similarity threshold, and b' is a label.

The trained network can be used for several purposes. To classify a new instance a, one iterates over all possible labels b' and chooses $\arg\max_{b'} \mathrm{sim}(a, b')$ as the prediction. For ranking, entities can be sorted by their predicted similarity score. The embedding vectors can also be extracted and used for some other downstream task. Wu et al. (2018) recommend that the similarity function $sim(\cdot, \cdot)$ is shaped in such a way that it will directly fit the intended application, so that training will be more effective.

A few examples of tasks successfully tackled with the StarSpace feature transformation approach are described below.

- In *multiclass text classification* the positive instances (a, b) are taken from the training set of documents E^+, represented with bags-of-words and their labels b. For negative instances, entities b_i^- are sampled from the set of possible labels.
- In *recommender systems* users are described with a bag of items they liked (or bought). The positive instances use a single user ID as a and one of the items that user liked as b. Negative instances take b_i^- from the set of possible items. Alternatively, to work for new users, the a part of user representation is composed of all the items that user liked, except one, which is used as b.
- For *link prediction* the concepts in a graph are represented as triples head-relation-tail (h, r, t), e.g., gene-generates-protein. A positive instance a consists either of h and r, while b consists of t; alternatively, a consists of h, and b consists of r and t. Negative instances b_i^- are sampled from the set of possible concepts. The trained network can then predicted links, e.g., gene-generates-what.

Fig. 2 Example input data for a deep relational machine that operates on the instance level

Instance	$f_1 \wedge f_2$	$f_3 \wedge f_2$	$f_1 \wedge f_3$	$f_5 \wedge f_2$	$f_4 \wedge f_1 \wedge f_5$	Class
1	1	1	1	1	0	+
2	0	1	0	0	1	+
3	0	0	1	0	0	-
4	0	1	0	0	1	-
5	1	0	0	0	1	-

- For *sentence embedding* in an unsupervised fashion, a collection of documents, containing sentences, is turned into a training set. For positive instances, a and b are sentences from the same document (or are close together in a document), while for negative instances, sentences b_i^- are coming from different documents. This definition of a task tries to capture the semantic similarity between sentences in a document.

In the PropStar algorithm proposed in this work, we use StarSpace similarly to the first case mentioned above (multiclass text classification). Namely, Wordification returns a bag of features (relational items) for each instance in the target table. The embeddings are learned for each feature separately, and class labels are also embedded in the same space. During classification, representations of relational items associated with a given instance (bag of features) are averaged to obtain the representation of the instance—a similar idea as in the document representation adopted in the highly efficient doc2vec branch of algorithms aimed at document classification (Le and Mikolov 2014). The embedded instances, now located in the same vector space as the embeddings of class labels, are directly used for classification. The label, closest to the representation of a given target instance is selected as the final prediction.

4.3.3 Deep relational machines

Deep neural networks are effective learners in numeric space, capable of constructing intermediate knowledge constructs and thereby improve semantics of baseline input representation. Training deep neural networks on propositionalized relational data were explored by Srinivasan et al. (2019), following the work of Lodhi (2013), where Deep Relational Machines (DRMs) were first introduced. In Lodhi's work, the DRMs used bodies of first order Horn clauses as input to restricted Boltzmann machines, where conjuncts of bonds and other molecular structure information compose individual complex features; when all structural properties are present in a given instance, the target's value is true, and false otherwise. For example, consider the following propositional representation of five instances (rows), where complex features are comprised of conjuncts of atoms f_i, as illustrated in Fig. 2.

Note that the propositionalized data set P is usually a sparse matrix, which can represent additional challenge for neural networks. The DRMs proposed by Lodhi (2013) were used for prediction of protein folding properties, as well as mutagenicity assessment of small molecules. This approach used feature selection with information theoretic measures such as information gain as the sparse matrix resulting from the propositionalization was not

suitable as an input to the neural network. The initial studies regarding DRMs explored how deep neural networks could be used as an extension of relational learning.

Recently, promising results were demonstrated in the domain of molecule classification (Dash et al. 2018) using ILP learner Aleph in its propositionalization mode for feature construction. After obtaining propositional representation of data, the obtained data table was fed into a neural network that associated such representations with the output space (e.g., a molecule's activity). Again, sparsity and size of the propositionalized representation is a problem for deep neural networks. Again, stochastic feature selection of relational features that are used as input to deep relational machines can improve the performance and interpretability (Dash et al. 2019).

The work of Srinivasan et al. (2019) is relevant for the interpretability of deep relational machines, proposing a logical approximation of well-known prediction explanation method LIME (Ribeiro et al. 2016) and showing how it can be efficiently computed.

In summary, DRMs address the following issues at the intersection of deep learning and relational learning:

- DRMs demonstrated that deep learning on propositionalized relational structures is a sensible approach to relational learning.
- Their input is comprised of logical conjuncts, offering the opportunity to obtain human-understandable explanations.
- DRMs were successfully employed for classification and regression.
- Emerging ideas in the area of representation learning have only recently been explored in the ILP context (Dumančić et al. 2018), indicating there are many possible improvements both in terms of execution speed, as well as more informative feature construction on the symbolic side of computation.

We further discuss DRMs in the context of efficiency of their implementation in Sects. 6.1.1 and 6.2.2. Development of DRMs that are efficient with respect to both space and time is an ongoing research effort. Building on the ideas of DRMs, we implemented a variant of this approach, capable of learning directly from large, sparse matrices that are returned from Wordification of a given relational database, rather than using feature selection or the output of Aleph's feature construction approach. Our novel, efficient DRM implementation is presented in Sect. 6.2.2.

5 Unifying framework for propositionalization and embeddings

The connection we made between different information representation levels and different transformation techniques shows that propositionalization and embeddings are two sides of the same coin. If we view embeddings as transformations for texts, graphs, recommendations, electronic health records, and other entities with defined similarity function, we can conclude that all these transformation present a multifaceted approach to feature construction.

To this end, the paper contributes a novel understanding of these data transformation techniques. In Sect. 5.1, we first present a unified terminology and definitions, and explain the apparent differences between the definitions of propositionalizationa and embeddings as variants of a complex data transformation task. In further sections we explore the apparent differences between the two approaches. In Sects. 5.2, 5.3, and 5.4 we discuss

Table 1 Unifying and differentiating aspects of propositionalization and embeddings in terms of data representation

Representation	Propositionalization	Embeddings
Vector space	Symbolic	Numeric
Features/variables	Symbolic	Numeric
Feature values	Boolean (0 or 1)	Numeric
Sparsity	Sparse	Dense
Space complexity	Space consuming	Mostly efficient
Interpretability	Interpretable features	Non-interpretable

differences in data representation, learning, and use. Finally, in Sect. 5.5 we summarize strengths and limitations of propositionalization and embeddings.

5.1 Unifying definitions

Below we present a unified view on the definitions of propositionalization and embedding tasks, as instances of a general data transformation task defined in Sect. 1 via Definition 1.

Definition 2 (*Propositionalization*)

Given: Input data of a given data type and format, and heterogeneous background knowledge of various data types and formats.

Find: A tabular representation of the data enriched with the background knowledge, where each row represents a single data instance, and each column represents a feature in a d-dimensional symbolic[1] vector space F^d.

Definition 3 (*Embedding*)

Given: Input data of a given data type and format, and heterogeneous background knowledge of various data types and formats.

Find: A tabular representation of the data enriched with the background knowledge, where each row represents a single data instance, and each column represents one of the dimensions in the d-dimensional numeric vector space $\mathbb{R}^d$.

5.2 Unifying propositionalization and embeddings in terms of data representation

Both data transformation techniques result in a vector space representation. The unifying dimensions of propositionalization and embeddings in terms of data representation, which are summarized in Table 1, are explained below.

In propositionalization, the transformation results in a binary matrix of sparse binary vectors, where rows corresponds to training instances and columns correspond

[1] In the case of binary valued features, each value in each column is $\in \{0, 1\}$.

Table 2 Unifying and differentiating aspects of propositionalization and embeddings in terms of learning context

Learning	Propositionalization	Embeddings
Meaning capturing	Via symbols	Via distances
Search strategy	Heuristic search	Greedy
Search goal	Global optimum	Local optimum
Typical algorithms	Symbolic, linear regression, SVM	Deep neural networks
Parameters	Few	Many
Hardware	CPU	CPU/GPU

to symbolic features constructed by a particular propositionalization algorithm. These features are human interpretable, as they are either simple logical features (such as attribute values), conjunctions of such features, relations among simple features (such as e.g., a test for the equality or inequality of values of two attributes of the same type), or relations among entities (such as links among nodes in a graph). Given that the number of constructed features is usually large, such transformation results in a sparse binary matrix with few non-zero elements.

Embeddings output is usually a dense matrix of a user-defined dimensionality, composed of vectors of numeric values, one for each object of interest. For neural network based embeddings, vectors usually represent the activation of neural network nodes of one or more levels of a deep neural network. Given a relatively low dimensionality of these vectors (from 100 to 1000) this dense representation is efficient in terms of space. However, the features/dimensions are non-interpretable, therefore a separate explanation mechanisms and visualizations are required.

5.3 Unifying propositionalization and embeddings in terms of learning

For both data transformation techniques, the resulting vector space representation is used as an input to a learning algorithm of the user's choice. The unifying dimensions of propositionalization and embeddings in terms of most frequently used learners (summarized in Table 2) are explained below.

After propositionalization, any learner capable of processing symbolic features can be used. Typical learners include rule learning, decision tree learning, random forests for a supervised setting, or association rules and symbolic clustering algorithms applied in a non-supervised learning setting. Learners usually use heuristic search to find a global optimum in terms of heuristics to be optimized (exceptions being, e.g., association rule learners using exhaustive search with constraints). Typical algorithms are decision tree learners, rule learners, linear regression and SVMs. Learners require some parameter tuning to achieve optimal results, but parameters are relatively few. Learning is typically performed on CPUs.

The embedded vectors are best suited for distance-based learners, such as neural networks, and to a lesser degree for kernel methods or logistic regression. Deep neural networks use greedy search to find locally optimal solutions, and are usually trained on GPUs, but can be used for prediction on both CPUs or GPUs. As a weakness, deep learning algorithms require substantial (hyper)parameter tuning.

Table 3 Unifying and differentiating aspects of propositionalization and embeddings in terms of use

Use	Propositionalization	Embeddings
Problems/context	Relational	Tabular, texts, graphs
Data type fusion	Enabled	Enabled
Explanation	Directly interpretable	Special approaches

5.4 Unifying propositionalization and embeddings in terms of use

The unifying dimensions of propositionalization and embeddings in terms of their use (summarized in Table 3) are explained below.

Propositionalization (Kramer et al. 2001) is one of the established methodologies used in relational learning (Džeroski and Lavrač 2001; De Raedt 2008) and ILP (Muggleton 1992; Lavrač and Džeroski 1994; De Raedt 2008) (see the propositionalization methods outlined in Sect. 4.2). The propositionalization approach was applied also in the semantic data mining where ontologies are used as a background knowledge in relational learning (Podpečan et al. 2011; Lavrač et al. 2009; Vavpetič and Lavrač 2011).

The embedding technologies are mostly used in the context of deep learning for various data formats, including tabular data, texts, images, and graphs (including knowledge graphs). In addition to knowledge graph embedding approaches (see Sect. 4.3.1), we outline some other approaches to graph embeddings below.

The first studies of graph embeddings were influenced by embedding construction from textual data. For example, the well known skip-gram model, initially used as part of word2vec (Mikolov et al. 2013) was successfully applied to learn node representations. DeepWalk (Perozzi et al. 2014) was one of the first learners that treats short random walks in graphs as sentences (or short phrases) to learn latent node embeddings. DeepWalk was revisited as node2vec (Grover and Leskovec 2016) to take into account different types of random walks, parameterized by breadth, as well as depth-first search. LINE (Tang et al. 2015b) performs similarly well for the tasks of classification and link prediction by attempting to optimize both local, as well as global network structure.

As for fusing heterogeneous data types, a propositionalization approach was proposed as a mechanism for heterogeneous data fusion (Grčar et al. 2013). As for data type fusion using embedding-based methods, PTE (Tang et al. 2015a) exploits heterogeneous networks of texts for supervised embedding construction. NetMF (Qiu et al. 2018) is a generalization of Deepwalk, node2vec, LINE and PTE, re-formulating them as a matrix factorization problem. Furthermore, struc2vec (Ribeiro et al. 2017) builds on two main ideas: representations of two nodes must be close if the two nodes are structurally similar, and the latent node representation should not depend on any node or edge attribute, including the node labels. Examples of approaches to heterogeneous graph embeddings include HINMINE (Kralj et al. 2018), metapath2vec (Zhu et al. 2018) and OhmNet (Žitnik and Leskovec 2017), an extension of node2vec to a heterogeneous biological setting. Heterogeneous data embeddings (Chang et al. 2015) of images, videos and text were also formulated as a task of heterogeneous graph embedding.

Concerning the interpretability of results, propositionalization approaches are mostly used with symbolic learners whose results can be interpretable, given the interpretability of features used in the transformed data description. For embedding-based methods, given the non-interpretable numeric features/dimensions, specific mechanisms need to be implemented to ensure results explanation (Robnik-Šikonja and Kononenko 2008; Štrumbelj and

Kononenko 2014). A recent well-known approach, which can be used in a post-processing phase of an arbitrary prediction model, is named SHAP (Lundberg and Lee 2017). In this approach, Shapley values offer insights into instance-level predictions by assigning fair credit to individual features for participation in prediction-explaining interactions. Explanation methods such as SHAP are commonly used to understand and debug black-box models. We refer the reader to Lundberg and Lee (2017) for a detailed overview of the method.

5.5 Summary of strengths and limitations of propositionalization and embeddings

Let us summarize the unified presentation of propositionalization and embeddings by presenting the strengths and weaknesses of the two approaches. The main strength of propositionalization is the interpretability of the constructed features, while the main strength of embeddings is high performance of classifiers learned from embeddings due to their compact representation in a vector space.

In terms of their strengths, both approaches to data transformation are: (a) automated, (b) fast, (c) semantic similarity of instances is preserved in the transformed instance space (as a remark, due to a more compact representation, embeddings preserve semantic similarity of features even better than propositionalization), (d) transformed data can be used as input to standard propositional learners, as well as to contemporary approaches.

In addition to these characteristics, embeddings have other favorable properties: (a) embedded vectors representations allow for transfer learning, e.g., for cross-lingual applications in text mining or image classification from different types of images, (b) cover a very wide range of data types (text, relations, graphs, images, time series), and (c) have a very wide community of developers and users, including industry.

In terms of their limitations when used in a multi-relational setting, both approaches to data transformation: (a) are limited to 1-many relationships (cannot handle many-to-many relationships between the connected data tables), (b) cannot handle recursion, and (c) cannot be used for predicate invention.

In addition to these characteristics, limitations of propositionalization include: (a) only boolean values are used in the transformed vector space, (b) generated sparse vectors can be memory inefficient, (c) limited range of data types are handled (relations, graphs), and (d) a small community of developers and users (mainly from ILP).

Embeddings also have several limitations: (a) loss of explainability of features and consequently of the models trained on the embedded representations, (b) many user-defined hyper-parameters, (c) high memory consumption due to many weights in neural networks, and (d) requirement for specialized hardware (GPU) for efficient training of embeddings, which may be out of reach for many researchers.

6 Proposed unification methodology and its two implementations

The unifying aspects analyzed in Sect. 5 can be used as a basis for a unifying methodology that combines propositionalization and embeddings, and benefits from the advantages of both. The propositionalization successfully captures relational information through complex relational feature construction, but results in a sparse symbolic feature vector representation. This weakness can be successfully overcome by embedding the constructed feature

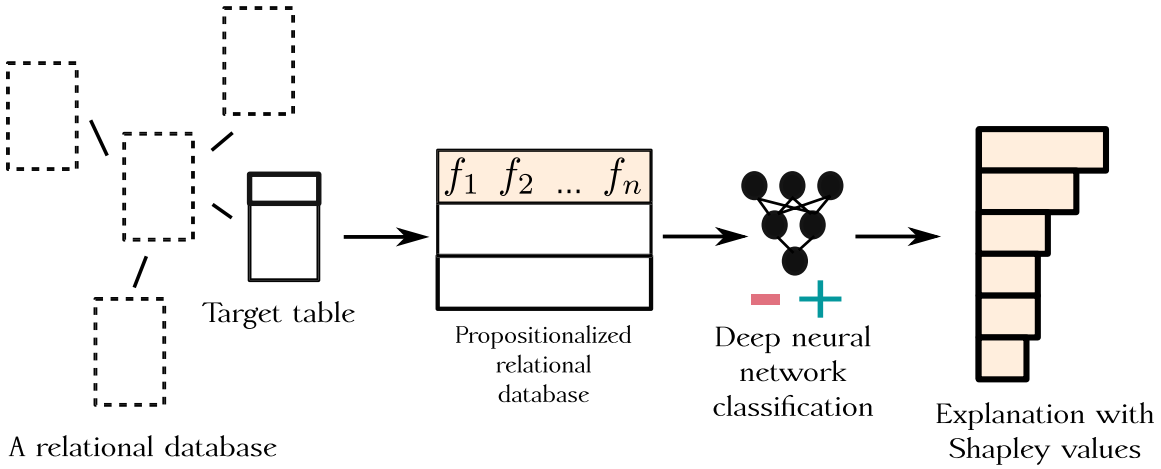


Fig. 3 Overview of the PropDRM instance-based embedding methodology, based on DRMs. Note that features in the propositionalized relational database represent either single features f_i or conjuncts of features, e.g., $f_i \wedge f_j$, given that Wordifications constructs both feature forms. For simplicity, the propositionalized database shows only two instances

vectors into a lower dimensional numeric vector space, resulting in a condensed numeric feature vector representation appropriate for use by modern deep learning algorithms.

To this end, we describe two novel data transformation algorithms, combining propositionalization and embedding based transformations into a joint data transformation framework. We first briefly outline the two approaches in Sect. 6.1, followed by their detailed descriptions in Sect. 6.2.

6.1 Outline of proposed data transformation and learning methods

We first overview the proposed unifying data transformation approaches. The first, named PropDRM, is an instance-based data transformation approach. The second one is a feature-based data transformation pipeline, called PropStar. The approaches are outlined in the next two subsections.

6.1.1 PropDRM: an instance-based approach

The first unifying approach for embedding of multi-relational databases is based on Deep Relational Machines (Dash et al. 2018) (DRMs), presented in Sect. 4.3.3. Rather than using the output of Aleph's feature construction approach, as was the case in the DRM implementation of Dash et al. (2018), we implemented a variant of this approach, capable of learning directly from large, sparse matrices that are returned by the Wordification (Perovšek et al. 2015) approach to propositionalization of relational databases. In this work, following the paradigm of propositionalization by Wordification, each instance is described by a bag (a multiset that allows for multiple appearances of its elements) of features of the form *TableName_AttributeName_Value*. Wordification treats these simple easily interpretable features as 'words' in the transformed Bag-Of-Words representation. In this work, they represent individual 'relational items' and we use the notation (table.name, column.name, value).

Relational representations are thus obtained for individual instances, resulting in embeddings of instances (e.g., molecules, persons, companies etc). Batches of instances are then

fed to a neural network, which performs the desired down-stream task, such as classification or regression. Schematically, the approach is illustrated in Fig. 3.[2]

Note that although propositionalization and subsequent learning are conceptually two distinct steps, they are not necessarily separated when implemented in practice: as neural networks operate with small batches of input data, if propositionalization is capable of similar batch functionality, relational features can be generated in a lazy manner when needed by the neural network. The technical details of the proposed PropDRM implementation are presented in Sect. 6.2.2.

When compared to our PropStar algorithm presented in Sects. 6.1.2 and 6.2.3 below, the key difference of the outlined DRM-based implementation of the unifying methodology is the type of embeddings: PropDRM embeds instances (i.e. whole bags of constructed features), whereas PropStar embeds features along with the class values in the same vector space.

6.1.2 PropStar: a feature-based approach

In this section, we outline the proposed PropStar algorithm for classification via feature embedding. Its details and implementation are presented in Sect. 6.2.3. Unlike the PropDRM algorithm, where each embedding vector represents a single data *instance*, the idea of PropStar is to use embedding vectors to represent the *features* of the data set. Here, individual relational features, obtained as the result of propositionalization by Wordification, are used by a supervised embeddings learner to obtain representations, co-located with instance labels. This approach is conceptually different in the sense that representations are not learned for individual instances (as is the case of DRMs); instead, they are learned for every single relational feature that is the output of the selected propositionalization algorithm (i.e. Wordification).

The fact that PropStar produces vector representations of features means that the labels (label=true and label=false) are also represented by vectors in the same dense space as the other vectors. This leads to an intuitive direct classification of new examples. We can observe the set of vectors representing the relational items present in the itemset representing the new example. To classify a new instance, the embeddings of the set of its features (i.e. true values) are averaged and the result is compared to the embedding of class labels. The nearest class label is chosen as the predicted value.

Figure 4 illustrates how new instances are classified by direct comparison of the representations of their features in the latent dense semantics-preserving space that also contains the information on labels. The classification is based on the proximity to a given label (in the latent space). If the center of feature vectors of a given instance is closer to the vector representing the feature label=true, then the example is classified as positive.

In contrast to the instance-based embeddings discussed in Sect. 6.1.1, which relies on batches, the whole data set is needed to obtain representations for individual features. To avoid high spatial complexity, this class of algorithms would ideally operate on sparse inputs. An example of feature-based embeddings are items that are to be recommended to users, where the representation of a given item is obtained by jointly optimizing the item's co-occurrence with other items, as well as other user's properties. In

[2] As its last step, the methodology includes the explanation of results using the SHAP approach. However, as Sect. 6 focuses on our research contributions, this well known approach and its results are presented in Appendix D.

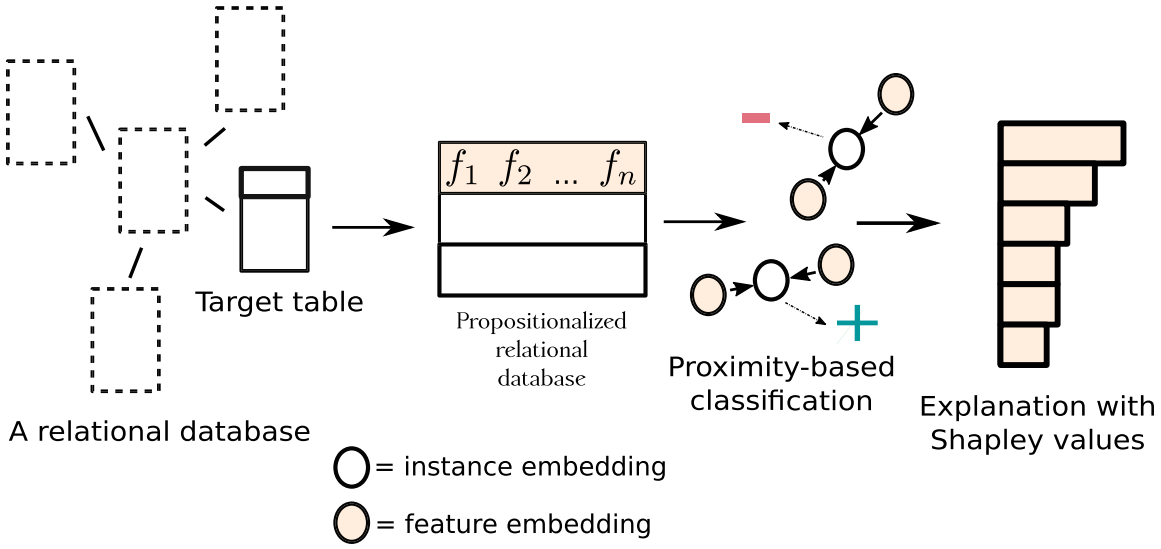


Fig. 4 Overview of the proposed feature-based embedding methodology PropStar. Note that embedded features represent embeddings of single features f_i or of conjuncts of features, e.g., $f_i \wedge f_j$, given that Wordifications constructs both feature forms. For simplicity, the propositionalized database shows two instances Blank and shaded circles correspond to embedded representations of instances and features, respectively

a relational setting considered in this work, we follow the paradigm of propositionalization by Wordification, where each instance is described by a bag of features of the form (table.name, column.name, value). Consequently, in the PropStar approach the embeddings represent bags of such features and their conjunctions (of size 2). There are as many embeddings as there are unique *features* in the propositionalized representation of a given relational database. As such embeddings by themselves do not contain any information which relates them to the desired output space, target values get embedded alongside other features in a supervised manner.

6.2 Detailed description of proposed data transformation and learning methods

This section presents the implementations of the proposed methods, preceded by the description of the updates to the Wordification algorithm (Perovšek et al. 2015 for multi-propositionalization algorithm presented in Sect. 6.2.1. In Sect. 6.2.2 we discuss how Deep Relational Machines (described briefly in Sect. 4.3.3), which use neural networks for learning from relational databases, were adapted to operate on sparse matrices generated by an improved Wordification algorithm. In Sect. 6.2.3 we describe a novel algorithm, called PropStar, which embeds relational features, extracted as part of propositionalization.

6.2.1 Improving the efficiency of Wordification

In this work we significantly extend the ideas proposed in Wordification (Perovšek et al. 2013, 2015) with the aim to maintain the classification performance, yet improve its *scalability*. Both proposed algorithms build on the idea of Wordification, yet its use in our algorithms is differentiated by the following design decisions:

1. Inputs do not need to be hosted in relational databases. PropStar operates on .sql files directly. The algorithm supports SQL conventions, as commonly used in the ILP com-

munity.[3] This modification renders the method completely local, enabling offline execution without additional overhead. Such setting also offers easier parallelism across computing clusters.
2. Algorithm is implemented in Python 3 with minimum dependencies for computationally more intense parts, such as the Scikit-learn (Pedregosa et al. 2011), Pandas, and Numpy libraries (Van Der Walt et al. 2011). All database operations are implemented as array queries, filters or similar, unlocking the potential to run PropDRM and PropStar also on GPUs.
3. As shown by Perovšek et al. (2015), Wordification's caveat is extensive sampling of (all) tables. We relax this constraint to close (up to second order) foreign key neighborhood, notably speeding up the relational item sampling part, but with some loss in terms of relational item diversity. For larger databases, minimum relational item frequency can be specified, constraining potentially noisy parts of the feature space.

One of the original Wordification's most apparent problems is its spatial complexity. In this work we address this issue as follows:

1. Relational items are hashed for minimal spatial overhead during sampling.
2. During construction of the final representation, a sparse matrix is filled based on relational item occurrence.
3. The matrix is serialized directly into list-like structures, suitable for StarSpace algorithm and thus we maintain minimal spatial overhead.
4. Only the final representation is stored as a low-dimensional (e.g., 32) dense matrix.

6.2.2 Detailed description of the proposed PropDRM implementation

The novelty of the proposed implementation of DRM instance-based embedding, inspired by the work of Dash et al. (2018), concerns its capability to effectively handle the sparseness of the data with deep neural networks. The main novelty of the proposed implementation is that it is indeed capable of operating on larger, sparse matrices directly. Such capability is necessary for DRMs to be compatible with propositionalization, which yields large sparse matrices as the main output. Below we discuss the neural network architecture and its adaptations.

Let P represent a sparse item matrix, as returned by Wordification (discussed in Sects. 4.2.3 and 6.2.1). Note that Wordification is unsupervised, and thus does not include any information on instance labels. The neural network we use (termed ω) represents the mapping $\omega : P \rightarrow \mathcal{C}$, where $\mathcal{C}$ is the set of classes. In this work, we experimented with dense feed-forward neural networks, regularized using dropout (Srivastava et al. 2014), and ELU activation function (Clevert et al. 2016) (of intermediary weights). The output weights are activated using sigmoid activation (σ) in order to obtain binary predictions.

$$ELU(x) = \begin{cases} c(e^x - 1), & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases},$$

where c is the user-specified constant. For a given input matrix P, an example of a single hidden-layer neural network is defined as follows.

[3] https://relational.fit.cvut.cz/.

$$\omega = \sigma(W_o^T(\text{ELU}(\text{Drop}(W_1^T P + b_1))) + b_o).$$

Here, the σ is a sigmoid activation, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. The W_1 is the weight matrix, P the sparse input space, and b_l the bias vector of a given layer $l \in \{0, 1\}$. The described neural network returned satisfactory results, hence, we did not perform neuroevolution or similar large-scale search for potentially better performing architectures. Throughout this work, we use the binary cross-entropy loss, referred to as *Loss*. For a given probabilistic classifier, which returns a probability p_{ij} of an instance i belonging to a class j, the loss function is defined as follows:

$$\text{Loss}(i) = \sum_{j \in \mathcal{C}} y_{ij} \cdot \log p_{ij}.$$

Here y_{ij} is a binary value (0 or 1) indicating whether class j is the correct class label assigned to instance i, and $\mathcal{C}$ is a set of all the target classes. In the case of DRMs, where the instances of a relational database (one of the tables) are classified, each of the $|\mathcal{C}|$ output neurons predicts a single probability p_{ij} for a given target class $j \in \mathcal{C}$. If the neural networks are trained in small batches, the results of the Loss function are averaged to obtain the overall loss of a given batch of instances.

Neural networks are adapted for dense inputs such as images and texts, and are not necessarily suitable for large sparse matrices, as considered in this work (i.e. P). The proposed variant of DRMs is adapted as follows. Once the batch size bs (a free parameter) is determined, propositionalized representation P is traversed (in chunks of bs instances). Note that each instance is effectively a d-dimensional vector. As the neural network operates with dense batches, each batch is converted to a dense matrix of $bs \cdot d$ elements that is used during matrix multiplication within the neural network. The spatial complexity is thus at most $\mathcal{O}(bs \cdot d)$. We observed that even by considering batch size of one, the DRMs are stable and efficient.

6.2.3 Detailed description of the PropStar algorithm

We next present the novel feature-based embedding algorithm that can operate directly on the propositionalized relational databases. The proposed PropStar algorithm merges symbolic and non-symbolic representations as part of a single procedure for obtaining real-valued representations of features in arbitrary relational databases. The pseudocode of the PropStar algorithm is given in Algorithm 1.

Algorithm 1: The pseudocode of PropStar algorithm.

```
Data: A Relational database R, foreign key map m
Parameters : Entity embedding parameter set E, representation dimension d,
             target table T, target attribute t
itemContainer ← empty bag of items ;                          ▷ Begin Wordification.
foreach instance i ∈ T do
    relationalItems ← {} ;
    candidateKeys ← getForeignKeys(R,i) ;                     ▷ Links to other tables.
    candidateTables ← getCandidateTables(candidateKeys, R) ;  ▷ Linked tables.
    foreach table ∈ candidateTables do
        while not final number of items do
            bagOfItems ← WORDIFY(table(m(instance))) ;
            add bagOfItems to relationalItems ;               ▷ Store sampled items.
        end
    end
    itemContainer[i].add(relationalItems) ;                   ▷ Store relational items.
end
relevantFeatures ← frequencySelection(itemContainer.values, d) ;
symRep← [] ;                          ▷ Sparse vector representations of instances.
foreach instance i ∈ targetTable do
    instanceItems ← itemContainer[i] ;
    propRep ← RelationalFeatures(relevantFeatures, instanceItems) ;
    symRep.append(propRep) ;
end
featureEmbeddings ← StarSpace(symRep, T[t], E) ;          ▷ Input is a sparse matrix.
return featureEmbeddings ;
```

The algorithm consists of two main steps. First, a relational database is transformed into sets of features describing individual instances. The WORDIFY method constructs features of the form (table.name, column.name, value) and uses them to describe each individual instance (see Sect. 6.2.1 for a detailed formulation of this step).

Second, sets of relational items (features) are used as input to the StarSpace entity embedding algorithm (described in Sect. 4.3.2), producing embeddings for each distinct relational item. The StarSpace embeddings are computed using efficient C++ implementation. We wrote a wrapper which seemingly integrates the first part of PropStar (sampling and propositionalization) with the embedding construction. The problem is formulated as a multiclass classification, where the positive pair generator comes directly from a training set of labeled data specifying $(a, b) \in E^+$ pairs where a are relational item 'documents' and b are labels (singleton features). Negative entities b_i^- are sampled from the set of possible labels. Inputs can be described as (multi) sets comprised of both relational items f_i, their conjuncts, as well as class labels c_i. For example,

$$\{f_1, f_2, f_6, f_6 \wedge f_2, c_1\}$$

represents a simple input consisting of three relational items, a conjunct and the target label c_1. Note that we apply StarSpace in such manner that the representations are learned for *individual relational items*. A representation matrix of dimension $\mathbb{R}^{|W| \times d}$ is produced as the final output ($|W|$ represents the number of unique relational items considered). Intuitively, the embedding construction can be understood as determining relational item locations in a latent space based on their co-occurrence with other items present in all training instances. The wrapper can be called via 'fit' and 'predict' methods, commonly used in contemporary data science and machine learning. In this work, we consider the inner product similarity

between a pair of vectors $\boldsymbol{e}_1, \boldsymbol{e}_2$ for the construction of embeddings.[4], i.e. The complexity of obtainingsim$(\boldsymbol{e}_1, \boldsymbol{e}_2) = \boldsymbol{e}_1^T \cdot \boldsymbol{e}_2$. As the class labels are embedded in *the same* space as individual relational items, classification of novel bags of relational items is possible by direct comparison, as common tasks operating on word embeddings. We discuss this classification below.

Let M represent a novel instance to be classified. Note that M (without additional index) is considered a multiset of relational items. For prediction purposes, StarSpace averages the representations of relational items present in a given input instance (a bag). The representation is normalized (as during training) and compared to label embeddings in the common space. Representation of a relational bag $\boldsymbol{e}_M$ is computed (with considered hyperparameters) as:

$$\boldsymbol{e}_M = \frac{\bigoplus_{f_i \in M} \boldsymbol{e}_{f_i}}{\sqrt{|M_{\text{unique}}|}},$$

which is a d-dimensional, real-valued vector. Note that $\oplus$ in this expression denotes element-wise summation. The M_{unique} represents the set of all (unique) relational features currently considered. Note that original bags of features can be redundant (multisets), yet representations are learned for unique features. Next, the similarity of this vector is compared to the label embeddings in the same space. The label that is the most similar to e_M is the top-ranked prediction, the second most similar label is the second-ranked prediction, etc. In this work we consider only the top-ranked prediction, resulting in the following label assignment:

$$\text{label}(e_M) = \arg\max_{c \in \mathcal{C}} \left[\text{sim}(\boldsymbol{e}_M, \boldsymbol{e}_c)\right].$$

The complexity of obtaining a single prediction is hence $\mathcal{O}(|\mathcal{C}|)$, not taking the complexity of scalar product for function sim into account. The PropStar algorithm first samples the relational items with respect to the target table (lines 2-11 in Algorithm 1). Binary indicator function (relationalFeatures) is applied to obtain the propositionalized representation of the target table (line 12). Here, zeros represent absence of a given relational items, and ones their presence.[5] Finally, StarSpace is used to embed the table into a low-dimensional, real valued embedding (line 19).

The spatial complexity of PropStar is linear with respect to the number of non-zero elements in the propositionalized version of a relational database. The exact spatial complexity can be formulated as follows. Let row represent the average number of rows per table. Let n_t represent the number of tables and col the average number of columns per table. We improve the original spatial complexity of $\mathcal{O}(\text{rows} \cdot n_t \cdot 2^{\text{col}})$ by introducing a constraint, which determines the maximum number of relational items that can be considered. The exponential term in the initial complexity thus reduces to col times some constant, yielding the complexity of $\mathcal{O}(\text{rows} \cdot \text{col} \cdot n_t)$. This formulation yields a scalable propositionalization.

[4] Note that e_1, e_2 represent vector representations of relational items (i.e. features) in the output of propositionalization.

[5] Note that in the actual implementation CSR format of sparse matrices is used to reduce the spatial overhead of storing zeros.

7 Experimental evaluation

In this section we describe the implementation details of the proposed methods, the relational data sets used in the experiments, and the experimental evaluation of the proposed methods.

7.1 Implementation and hyperparameters

We discuss how the proposed methods were implemented, along with the hyperparameters explored. Both new methods (PropDRM and PropStar) are implemented in Python, with the following exceptions. In PropDRM, the DRMs are implemented in PyTorch. For PropStar we used the efficient StarSpace implementation written in C++, for which we build a wrapper offering basic embedding training and prediction functionality.

We used 10-fold stratified cross validation, which was conducted for individual hyperparameter settings. The best setting is reported, other are discussed in ablation studies. Experiments were performed on an of-the-shelf workstation with no GPUs (even though PropDRM and PropStar can exploit them). We intentionally omit the GPU-based training to explore the minimum hardware, required to perform competitively on the selected data sets—ILP baselines, such as Aleph and RSD are Prolog-based, and are to our knowledge not able to use multiple GPU threads simultaneously. The machine on which experiments were conducted had 128GB of RAM and 12 CPUs (Intel i8 series).

In PropDRM, we varied the dropout rate, learning rate, number of epochs, and the hidden layer size. In PropStar, we varied the number of negative samples, embedding dimension, learning rate, and the number of epochs.

The source code of our implementation is publicly available[6].

7.2 Relational data sets

Five relational database sources[7] (Motl and Schulte 2015) were used in the experiments. Their characteristics are summarized in Table 4.

Trains (Michie et al. 1994) data set is used in the East-West trains challenge problem, which is well-known in ILP. The East-West trains challenge is to predict whether a train is eastbound or westbound, based on the properties of eastbound and westbound cars. Trains contain variable number of cars, each having one of various shapes and carrying various loads.

Carcinogenesis (Srinivasan et al. 1997) task is to predict carcinogenicity of a diverse set of chemical compounds. The data set was obtained by testing different chemicals on rodents, where each trial would take several years and hundreds of animals. The data set consists of 329 compounds, of which 182 are carcinogens.

Mutagenesis (Debnath et al. 1991) task addresses the problem of predicting mutagenicity of aromatic and heteroaromatic nitro compounds. Predicting mutagenicity is an important task as it is very relevant to the prediction of carcinogenesis. The compounds from the data are known to be more structurally heterogeneous than in any other ILP

[6] https://github.com/SkBlaz/PropStar.

[7] Freely accessible at https://relational.fit.cvut.cz/.

Table 4 Properties of the experimental data tables

Trains	#rows	#attributes
Cars	63	10
trains	20	2
Carcinogenesis	#rows	#attributes
atom	9,064	5
canc	329	2
sbond_1	13,562	4
sbond_2	926	4
sbond_3	12	4
sbond_7	4,134	4
Mutagenesis 42	#rows	#attributes
atoms	1,001	5
bonds	1,066	5
drugs	42	7
ring_atom	1,785	3
ring_strucs	279	3
rings	259	2
Mutagenesis 188	#rows	#attributes
atoms	4,893	5
bonds	5,243	5
drugs	188	7
ring_atom	9,330	3
ring_strucs	1,433	3
rings	1,317	2
IMDB	#rows	#attributes
actors	7,118	4
directors	130	3
directors_genres	1,123	4
movies	166	4
movies_directors	180	3
movies_genres	408	3
roles	7,738	4
MovieLens	#rows	#attributes
actors	99,129	3
directors	2,201	3
movies	3,832	5
movies2actors	152,532	3
movies2directors	4,141	3
u2base	946,828	3
users	6,039	4

data set of chemical structures. The database contains 230 compounds of which 138 have positive levels of mutagenicity and are labeled as 'active'. Others have class value 'inactive' and are considered to be negative examples. We took the data sets from the original paper (Debnath et al. 1991), where the data was split into two subsets: a 188 compound data set and a smaller data set with 42 compounds.

IMDB database is publicly available in the SQL format.[8] This database contains tables of movies, actors, movie genres, directors, and director genres. The data set used in our experiments encompasses only movies whose titles and years of production appear in the IMDB's top-250 and bottom-100 chart (Snapshot taken on July 2, 2012). The snapshot contains 166 movies, along with all of their actors, genres and directors. We designate movies present in the IMDB top-250 chart as positive examples, and those in the bottom-100 as negatives.

MovieLens data set from the UC Irvine machine learning repository.[9] The data set is similar to IMDB above, however is much larger. Overall, the database consists of more than 1.2 million instances. The task is to predict gender of the movie database's users.

7.3 Results

We present the results of the empirical evaluation of the proposed methodologies on the presented set of standard benchmark ILP data sets. The accuracies of individual learners are given in Table 5, and the AUC scores are reported in Table 6. The results for Aleph, RSD, RelF and Wordification were taken from previous work of Perovšek et al. (2015).

It can be observed that the proposed unifying approaches perform competitively on most data sets. We can observe a distinct difference in performance on the Mutagenesis data sets, where both PropDRM as well as PropStar do not outperform the baselines on the smaller data set (Mut42), yet notably outperform the (best) baselines on the larger one (Mut188). Further, minor improvement over the baseline algorithms is also achieved on Carcinogenesis data set.

In terms of spatial complexity, the proposed methodology greatly outperforms the alternatives under a given set of constraints. Only PropDRM and PropStar scale to very large relational databases without specialized hardware. Detailed studies regarding the sensitivity of PropDRM and PropStar to their parameters are discussed in Appendices B and C, respectively.

We consider the presented results as very favorable for the two proposed approaches. In particular, PropStar is better than current state-of-the-art methods on 3 out of 6 data sets, and is therefore a method to take into consideration when attempting to solve any new relational problem.

7.3.1 Study of propositionalization projections

The considered propositionalization is entirely *unsupervised*. Only once the symbolic representations of instances are obtained, PropDRM and PropStar learn the associations to

[8] http://www.webstepbook.com/supplements/databases/imdb.sql.

[9] https://relational.fit.cvut.cz/dataset/MovieLens.

Table 5 Classification accuracy on different relational data sets

Propositionalization	Learner	Carc.	IMDB	Mut188	Mut42	Trains	MovieLens
	MajorityVote	0.55	0.73	0.67	0.69	0.50	0.72
Aleph (Perovšek et al. 2015)	J48	0.55	0.73	0.60	0.69	0.55	–
Aleph (Perovšek et al. 2015)	SVM	0.55	0.73	0.60	0.69	0.70	–
RSD (Perovšek et al. 2015)	J48	0.60	0.75	0.68	**0.98**	0.60	–
RSD (Perovšek et al. 2015)	SVM	0.56	0.73	0.71	0.69	0.80	–
RelF (Perovšek et al. 2015)	J48	0.60	0.70	0.75	0.76	0.65	–
RelF (Perovšek et al. 2015)	SVM	0.56	0.73	0.69	0.76	0.80	–
Wordification (Perovšek et al. 2015)	J48	0.62	0.82	0.67	**0.98**	0.50	–
Wordification (Perovšek et al. 2015)	SVM	0.61	0.73	0.82	0.79	0.50	–
Aleph (replicated)	J48	0.55	–	0.80	0.76	0.70	–
Aleph (replicated)	SVM	0.55	–	0.80	0.79	0.60	–
RSD (replicated)	J48	0.56	0.84	0.88	0.92	0.60	–
RSD (replicated)	SVM	0.60	0.82	0.89	0.84	0.80	–
Wordification (replicated)	J48	0.47	**0.85**	0.91	0.88	**0.90**	0.60
Wordification (replicated)	SVM	0.39	0.80	0.83	0.33	0.50	0.72
Treeliker	J48	0.58	–	0.77	0.81	0.50	–
Treeliker	SVM	0.60	–	0.90	0.80	0.70	–
PropDRM		0.63	0.73	0.91	0.86	0.70	0.72
PropStar		**0.66**	0.74	**0.92**	0.90	0.80	**0.74**

The best score for each dataset is in bold

For the proposed methods, we report average performance over 5 runs. The runs, marked with—were unable to finish in 12 h

individual classes. A good representation, however, already contains relevant information on the instance space. In Fig. 5, we projected the propositionalized Mutagenesis 188 and Trains instance space to two dimensions to qualitatively explore whether instances group or any meaningful patterns emerge. Understanding whether the symbolic space exhibits distinct structure on its own could offer insights into why the proposed methods perform well. For projecting the 10,000 dimensional space to two dimensions we used UMAP, a recently introduced non-linear dimensionality reduction method based on insights from manifold theory (McInnes et al. 2018).

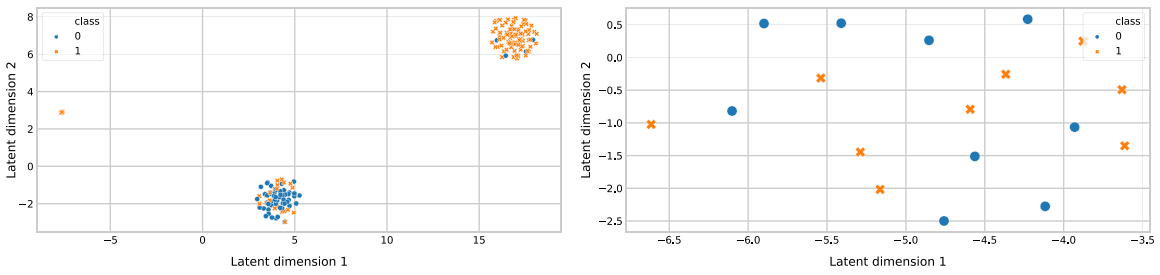
We can observe an apparent distinction in the clustering of the UMAP projections of the two propositionalized data sets. The Mutagenesis 188 data set consists of two distinct clusters that, when colored according to the class labels, approximately correspond to the two classes (Fig. 5a). On the other hand, the clustering is not apparent in the case of the Trains data set (Fig. 5b), where the instances do not group distinctly. The purpose of the considered visualizations is twofold. First, we show how the symbolic space can exhibit clustering properties, related to properties of instances such as class labels. Next, we show that projections do not necessarily exhibit such properties, indicating potentially harder classification problems. We believe that UMAP and similar tools offer insights into representation structure.

Table 6 AUC scores on individual data sets

Propositionalization	Learner	Carc.	IMDB	Mut188	Mut42	Trains	Movies
Aleph (from Perovšek et al. 2015)	J48	0.50	0.50	0.68	0.50	0.55	–
Aleph (from Perovšek et al. 2015)	SVM	0.50	0.50	0.68	0.50	0.70	–
RSD (from Perovšek et al. 2015)	J48	0.59	0.59	0.54	**0.96**	0.60	–
RSD (from Perovšek et al. 2015)	SVM	0.52	0.50	0.58	0.50	0.80	–
RelF (from Perovšek et al. 2015)	J48	0.59	0.66	0.68	0.68	0.75	–
RelF (from Perovšek et al. 2015)	SVM	0.52	0.50	0.54	0.62	0.75	–
Wordification (from Perovšek et al. 2015)	J48	0.61	**0.75**	0.55	0.96	0.95	–
Wordification (from Perovšek et al. 2015)	SVM	0.58	0.50	0.78	0.65	0.50	–
Alpeh (replicated)	J48	0.50	–	0.71	0.72	0.70	–
Aleph (replicated)	SVM	0.50	–	0.75	0.73	0.60	–
RSD (replicated)	J48	0.55	0.71	0.87	0.92	0.60	–
RSD (replicated)	SVM	0.58	0.65	0.90	0.73	0.80	–
Wordification (replicated)	J48	0.48	0.72	**0.90**	0.86	0.90	0.52
Wordification (replicated)	SVM	0.42	0.62	0.81	0.50	0.50	0.50
Treeliker	J48	0.58	–	0.75	0.71	0.50	–
Treeliker	SVM	0.58	–	0.88	0.68	0.70	–
PropDRM		**0.63**	0.68	**0.90**	0.87	0.80	0.54
PropStar		**0.63**	0.66	0.87	0.87	**0.95**	**0.56**

The best score for each dataset is in bold

We report average performance over 5 runs. The runs, marked with—were unable to finish in 12 h



(a) Mutagenesis 188. **(b)** Trains.

Fig. 5 Two UMAP projections of selected propositionalized data sets

7.3.2 Statistical comparison of PropDRM and PropStar

In previous sections, we demonstrated that both PropDRM and PropStar perform well on the considered data sets, indicating that both approaches are successfully unifying propositionalization and embeddings. We further study the differences in performances of the two approaches. For this purpose, we employ the hierarchical Bayesian t-test, a Bayesian test capable of comparing a pair of classifiers across multiple data sets (Benavoli et al. 2017; Corani et al. 2017). For this comparison, we selected the overall best performing hyperparameter sets for each method, and conducted ten repetitions of stratified ten-fold cross validation (for each data set). The results are visualized as probability distributions across the space of both classifiers and the 'rope' region (region of practical equivalence) within which the two classifiers perform the same. The size of this region is a free parameter of the hierarchical

t-test, and was set to 0.05 in this work. Other parameters of the test were left as defaults. The exact methodology for the interested reader is explained by Benavoli et al. (2017).

In terms of AUC, the probabilities returned by the Bayesian test were as follows: p(PropStar) = 0.07 and p(PropDRM) = 0.54), and in terms of classification accuracy, p(PropStar) = 0.96 and p(PropDRM) = 0.04. The results of statistical analysis indicate that with respect to AUC performance, the two approaches perform similarly, even though the probability that PropDRM will outperform PropStar is higher. With respect to the classification accuracy, PropStar outperforms PropDRM in majority of comparisons. Thus, considering the 95% or higher as the probability denoting significance boundary, we can determine that PropStar is (significantly) more suitable choice if accuracy is being optimized for. As Bayesian comparisons are computationally expensive, we compared the two methods using default hyperparameter settings. The PropStar's default configuration is not necessarily optimal when AUC is considered.

8 Conclusions and further work

This paper first provides a critical survey of propositionalization and embedding techniques, especially relevant for relational learning and inductive learning programming. While both data approaches, propositionalization and embeddings, aim at transforming data into the tabular data format, the research papers describing the approaches use different terminology and task definitions, claim to have different goals, and are used in very different contexts. In this paper, we define the main categories of data transformation techniques based on the representation they use and approaches employed. Propositionalization approaches produce tabular data from multirelational databases as well as from a mixture of tabular data and background knowledge in the form of logic programs or networked data, including ontologies. Knowledge stored in graphs can be assessed with community detection and graph traversal methods. Relations described with similarity matrices are encoded in a numeric form using matrix factorization. Currently, the most promising approach to data transformations are neural networks based methods which can be applied to text, graphs, and other entities for which we can define a suitable similarity function.

One of the main strategic problems machine learning has to solve is better integration of knowledge and models across different domains and representations. While the research area of embeddings can unify different representations in a numeric space, symbolic learning may be an essential ingredient for integration of different knowledge areas. We see our PropStar approach that combines advantages of propositionalization and neural embeddings in the same data fusion pipeline as a step in that direction.

The first minor contribution of the paper is that our exposition is based on three cognitive representation levels introduced by Gärdenfors (2000), i.e. neural, spatial, and symbolic. As most of human knowledge is stored in the symbolic form, while the most powerful machine learning algorithms take as input spatial representations, this explains a plethora of techniques that transform other forms of human knowledge into the spatial representation space. The next contribution is the unifying framework in which we describe propositionalization and embedding techniques in terms of their joint properties and their differences. We show how the propositionalization techniques can be merged with deep neural network based embedding to produce a joint embedding, such that spatial representation can be used with any deep learning algorithm and the predictions can be comprehensively explained. The main contributions of our work are thus the two implementations that merge propositionalization and embeddings in the same unifying methodology. The first is an efficient reimplementation

of existing Deep Relational Machines, while the second one is the novel Deep Propositionalization algorithm. We also contribute an experimental evaluation of the two algorithms and show favorable results in terms of predictive performance, as well as time and space requirements. The source code of both algorithms, DeepProp and PropDRM, is publicly available.[10]

In further work, it is worth investigating the integration of symbolic and deep learning, considering them as multitask learning approaches which try to integrate many different learning tasks and use embeddings as input representations. The issue is that different embedding methods have so far only been used in isolation. We already address this challenge in the current work of the authors, where we combine complementary embedding methods from different classes: in particular, to use network traversal methods to produce initial embeddings that are then refined using a deep neural network (Škrlj et al. 2019).

Acknowledgements We acknowledge the financial support of the Slovenian Research Agency through core research programmes P2-0103 and P6-0411 and project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078). The authors have received funding also from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825153 (EMBEDDIA). The work of the second author was funded by the Slovenian Research Agency through a young researcher grant. We wish to thank Jan Kralj for his insightful comments on the formulation of the proposed framework and for mathematical proofreading. Further, we are grateful to Vid Podpečan and Nika Eržen for their help with the implementation of the new version of the PyRDM library. Finally, we would like to thank the anonymous reviewers for careful reading, many insightful observations, and the encouragements to expand the initial work.

Appendix A: Wordification example

The Wordification approach is illustrated on a modified and substantially simplified version of the well-known East-West Trains domain (Michie et al. 1994). Our input database consists of just two tables shown in Fig. 6, where we have only one east-bound and one west-bound train, each with just two cars with certain properties[11].

The TRAIN table is the main table and the trains are the instances, with a class label denoting the direction of the train (east of west). As Fig. 7 shows, a multiset (a bag) of features is generated for each train t1 and t5 with the class label appended to the resulting feature vector (bag of features). Both single features and conjunctive features are shown in this example.

[10] https://github.com/SkBlaz/PropStar.

[11] Note that in the experiments we use the standard version of the East-West Trains domain.

TRAIN

trainID	eastbound
t1	east
...	...
t5	west
...	...

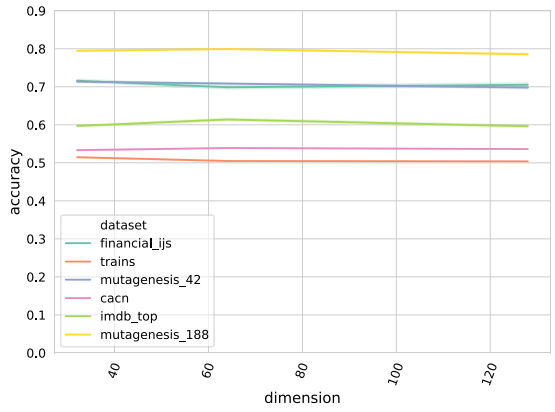
CAR

carID	shape	roof	train
c11	rectangle	none	t1
c12	rectangle	peaked	t1
...	...	...	...
c51	rectangle	none	t5
c52	hexagon	flat	t5
...	...	...	...

Fig. 6 Example input for Wordification in the East-West Trains domain

```
t1: {car_shape_rectangle, car_roof_none, car_shape_rectangle__car_roof_none,
     car_shape_rectangle, car_roof_peaked, car_shape_rectangle__car_roof_peaked}, east
    ...
t5: {car_shape_rectangle, car_roof_none, car_shape_rectangle__car_roof_none,
     car_shape_hexagon, car_roof_flat, car_shape_hexagon__car_roof_flat}, west
    ...
```

Fig. 7 The database from Fig. 6 in the bag-of-features representation (as in the original Wordification implementation, conjunctions of features are denoted by a long underscore instead of ∧)



(a) Dependence on hidden layer size.

(b) Dependence on the number of epochs.

(c) Dependence on the learning rate.

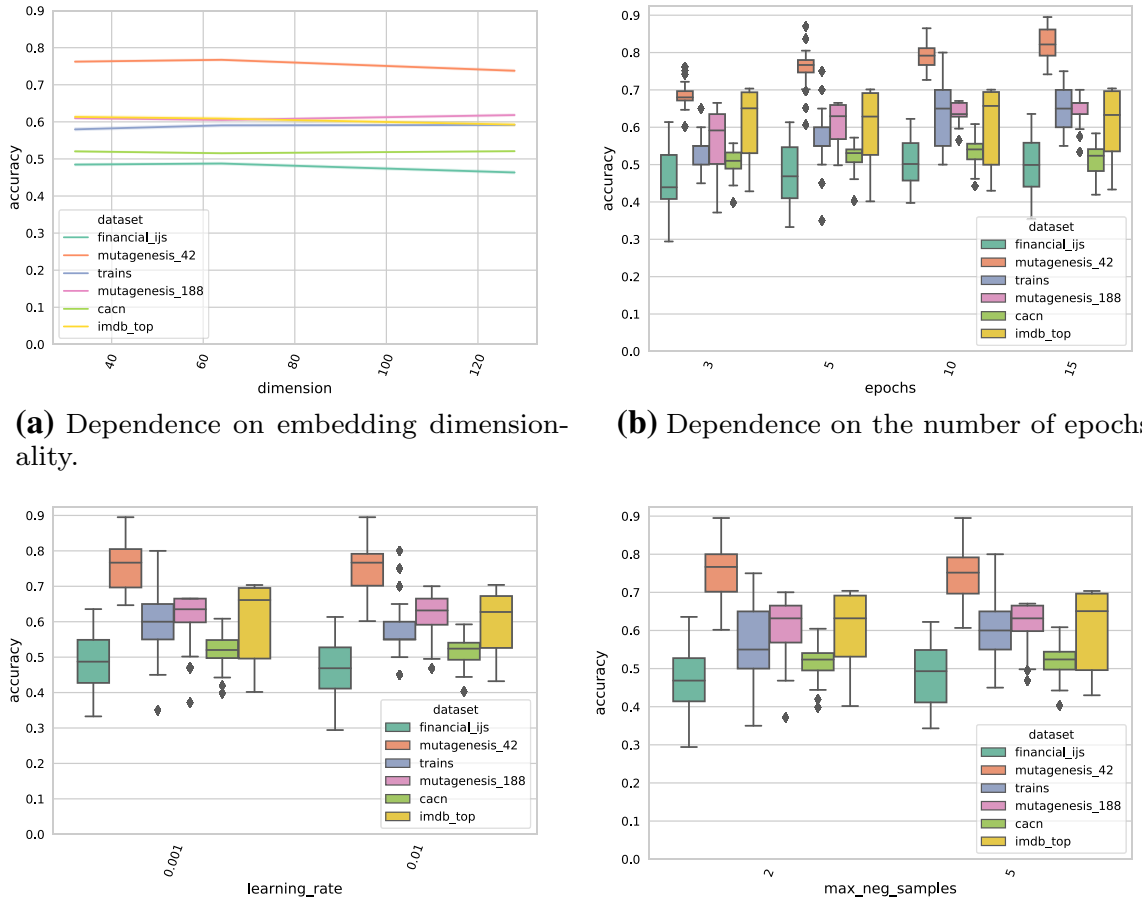
(d) Dependence on Dropout.

Fig. 8 Sensitivity of PropDRM to hyperparameter settings

Appendix B: Ablation study—PropDRM

We discuss the impact of individual hyperparameters on the performance of PropDRM. We first visualize the performance of PropDRM w.r.t. individual hyperparameters in Fig. 8.

We can observe that the relevance of individual hyperparameters varies from data set to data set. The learning rate, when too small, decreases the performance. In terms of embedding dimension, even smaller dimensions are sufficient for the considered data sets. This result potentially implies that the considered data sets are relatively small and contain only a small set of relevant features (when propositionalized). Thus, if the neural network detects the correct features as relevant, not many parameters are needed for a successful classification. An alternative explanation would imply that PropDRM learns hierarchical representations efficiently, albeit not optimized with their hierarchical nature in mind, which was previously demonstrated to capture hierarchical relations well (Nickel and Kiela 2017).



(a) Dependence on embedding dimensionality.

(b) Dependence on the number of epochs.

(c) Dependence on the learning rate.

(d) Dependence on the maximum negative sampling number.

Fig. 9 Sensitivity of PropStar to various hyperparameter settings

Appendix C: Ablation study—PropStar

We first explore the behavior with respect to various hyperparameter settings and visualize them in Fig. 9. We can observe that the amount of negative samples (Subfigure 9d)) impacts the PropStar's performance the most on the mutagenesis 42 data set, overall reducing the performance, even though a handful of models (outliers marked as dots) perform well. This indicates the importance of negative sample selection. As StarSpace does not use any sophisticated technique for sampling negative examples, the variability in performance could be notable due to this parameter.

It can be observed that a relatively small relational item embedding dimensionality is needed for successful performance. The dependence on other parameters varies from data set to data set. For example, the learning rate does not impact the larger Mutagenesis data set (Mut188) as much as it does the Trains data set. As the proposed methodology is not well adapted to such small data sets (e.g., tens of instances), large variability in performance could be linked to potential overfitting. Further, sufficient number of epochs are needed for PropStar to converge on individual data sets.

Appendix D: Interpretability of embedding-based methods using SHAP

The approximation power of deep neural network which are commonly used with embeddings comes at a cost of lesser interpretability. Compared to symbolic relational (or propositional) learners, one cannot understand the deep relational models' deductive process by inspecting the model. However, *post hoc* explanation methods for prediction models can be used to better understand which parts of the feature space are relevant for the neural network's individual predictions. In this work, we leverage the state-of-the-art explanation tool SHAP (Lundberg and Lee 2017), based on the coalitional game theory. This tool captures the importance of interactions between features with Shapley values.

When considered in a feature importance scenario, the contribution of the i-th instance τ_i, is approximated by SHAP with the following expression:

$$\tau_i = \underbrace{\sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!}}_{\text{All possible subsets}} \underbrace{\left[f(x_{S \cup \{i\}}) - f(x_S)\right]}_{\text{Difference in predictive performance}}$$

where S is a subset of all features F, f is the used predictive model, and x_S is an instance containing only features from the subset S. Shapley valufs offer insights into instance-level predictions by assigning fair credit to individual features for participation in interactions. They are commonly used to understand and debug black-box models.

In this work, we use the SHAP kernel approximator, the recently introduced, model-agnostic method for explaining model outputs. The used SHAP kernel explainer is considered an additive feature attribution method. Such methods are characterized as having an explanation model g that is a linear function of binary variables:

(a) Mean of SHAP explanations over the instances for PropDRM.

(b) Mean of SHAP explanations over the instances for PropStar.

Fig. 10 SHAP Kernel explanations of the two developed approaches

$$g(z') = \phi_0 + \sum_{i=1}^{|F|} \phi_i \cdot z'_i$$

where $z' \in \{0,1\}^{|F|}$, $|F|$ is the number of input features and $\phi_i \in \mathbb{R}$. This class of models assign an effect ϕ_i to each feature, and summing the effects of all such feature attributions approximates the output $f(x)$ of the original model. Detailed theoretical analysis of how this idea can be extended to approximation of outputs via a kernel is given in Lundberg and Lee (2017).

As an example demonstrating the explainability of the two paradigms, we visualize the Shapley values as explanations of learned classifiers for Mutagenesis 188 problem in Fig. 10. Explanations indicate parts of the feature space that have the largest impact on the model's output. Even though the considered SHAP kernel explainer is known to be a computationally expensive variant of SHAP (it is also the most general one), explanations were obtained in the order of minutes, indicating the potential of this methodology for explanations of predictors in larger relational databases.

References

Ahmed, C. F., Lachiche, N., Charnay, C., Jelali, S. E., & Braud, A. (2015). Flexible propositionalization of continuous attributes in relational data mining. *Expert Systems with Applications*, *42*(21), 7698–7709.

Benavoli, A., Corani, G., Demšar, J., & Zaffalon, M. (2017). Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, *18*(1), 2653–2688.

Bennett, K. P., Buja, A., Freund, W. S. Y., Schapire, R. E., Friedman, J., Hastie, T., et al. (2008). Responses to [52]. *Journal of Machine Learning Research*, *9*, 157–194.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, *3*(Jan), 993–1022.

Blockeel, H., Raedt, L. D., & Ramon, J. (1998). Top-down induction of clustering trees. In *Proceedings of the 15th international conference on machine learning*, pp. 55–63. Morgan Kaufmann.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems*, pp. 2787–2795.

Bordes, A., Glorot, X., Weston, J., & Bengio, Y. (2014). A semantic matching energy function for learning with multi-relational data. *Machine Learning*, *94*(2), 233–259.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Pacific Grove, CA: Wadsworth & Brooks.

Chang, S., Han, W., Tang, J., Qi, G. J., Aggarwal, C. C., & Huang, T. S. (2015). Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 119–128. ACM.

Charnay, C., Lachiche, N., & Braud, A. (2015). CARAF: Complex aggregates within random forests. In *Inductive logic programming—25th international conference, ILP 2015*, Kyoto, Japan, August 20–22, 2015, Revised Selected Papers, pp. 15–29. Springer.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*(4), 261–283.

Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). In *International conference on representation learning, ICLR*. arXiv:1511.07289.

Corani, G., Benavoli, A., Demšar, J., Mangili, F., & Zaffalon, M. (2017). Statistical comparison of classifiers through Bayesian hierarchical modelling. *Machine Learning*, *106*(11), 1817–1837.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Cumby, C. M., & Roth, D. (2003). On kernel methods for relational learning. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 107–114.

Dash, T., Srinivasan, A., Vig, L., Orhobor, O. I., & King, R. D. (2018). Large-scale assessment of deep relational machines. In *Proceedings of the international conference on inductive logic programming*, pp. 22–37. Springer, Berlin.

Dash, T., Srinivasan, A., Joshi, R. S., & Baskar, A. (2019). Discrete stochastic search and its application to feature-selection for deep relational machines. In I. V. Tetko, V. Kůrková, P. Karpov, & F. Theis (Eds.), *Artificial neural networks and machine learning: ICANN 2019–deep Learning* (pp. 29–45). Berlin: Springer.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, *34*(2), 786–797.

Demeester, T., Rocktäschel, T., & Riedel, S. (2016). Lifted rule injection for relation embeddings. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1389–1399.

Dumančić, S., Guns, T., Meert, W., & Blockleel, H. (2018). Auto-encoding logic programs. In *Proceedings of the international conference on machine learning*, Stockholm, Sweden.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. In *International conference on inductive logic programming*, pp. 92–103. Berlin: Springer.

Flach, P., & Lachiche, N. (2001). Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, *42*(1/2), 61–95.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Friedman, J. H., & Fisher, N. I. (1999). Bump hunting in high-dimensional data. *Statistics and Computing*, *9*(2), 123–143.

Gärdenfors, P. (2000). *Conceptual spaces: The geometry of thought*. Cambridge, MA: MIT Press.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge: MIT Press.

Grčar, M., Trdin, N., & Lavrač, N. (2013). A methodology for mining document-enriched heterogeneous information networks. *The Computer Journal*, *56*(3), 321–335.

Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 855–864.

Guo, S., Wang, Q., Wang, L., Wang, B., & Guo, L. (2016). Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 192–202.

Haussler, D. (1999). Convolution kernels on discrete structures. Tech. rep., Department of Computer Science, University of California.

He, S., Liu, K., Ji, G., & Zhao, J. (2015). Learning to represent knowledge graphs with Gaussian embedding. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pp. 623–632. ACM.

Kralj, J., Robnik-Šikonja, M., & Lavrač, N. (2018). HINMINE: Heterogeneous information network mining with information retrieval heuristics. *Journal of Intelligent Information Systems*, *50*(1), 29–61.

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Džeroski & N. Lavrač (Eds.), *Relational data mining* (pp. 262–291). Berlin: Springer.

Krogel, M. A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In *Proceedings of international conference on inductive logic programming*, pp. 142–155. Berlin: Springer.

Krogel, M. A., Rawles, S., Železný, F., Flach, P., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In T. Horvath & A. Yamamoto (Eds.), *Proceedings of the 13th international conference on inductive logic programming (ILP-2003* (pp. 197–214). Berlin: Springer.

Kuželka, O., & Železný, F. (2008). HiFi: Tractable propositionalization through hierarchical feature construction. In Železný, F., Lavrač, N. (Eds.) *Late breaking papers, the 18th international conference on inductive logic programming*, pp. 69–74.

Kuželka, O., & Železný, F. (2011). Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, *83*(2), 163–192.

Lachiche, N., & Flach, P. A. (2003). 1BC2: A true first-order Bayesian classifier. *Proceedings of inductive logic programming*, pp. 133–148.

Lavrač, N., Džeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the 5th European working session on learning (EWSL-91)*, pp. 265–281. Springer, Porto, Portugal.

Lavrač, N., Kralj Novak, P., Mozetič, I., Podpečan, V., Motaln, H., Petek, M., & Gruden, K. (2009). Semantic subgroup discovery: Using ontologies in microarray data analysis. In *Proceedings of the 31st annual international conference of the IEEE EMBS*, pp. 5613–5616.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. New York: Ellis Horwood.

Lavrač, N., & Flach, P. (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, *2*(4), 458–494.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of international conference on machine learning*, pp. 1188–1196.

Lewis, D. D. (1992). An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval*, pp. 37–50 .

Lodhi, H. (2013). Deep relational machines. In *Proceedings of the international conference on neural information processing*, pp. 212–219. Berlin: Springer.

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.) *Advances in neural information processing systems*, pp. 4765–4774.

McInnes, L., Healy, J., Saul, N., & Grossberger, L. (2018). UMAP: Uniform manifold approximation and projection. *The Journal of Open Source Software*, *3*(29), 861.

Mease, D., & Wyner, A. (2008). Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, *9*, 131–156.

Michalski, R. S., Mozetič, I., Hong, J., & Lavrač, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. In *Proceedings of the 5th national conference on artificial intelligence*, pp. 1041–1045. Philadelphia, PA.

Michie, D., Muggleton, S., Page, D., & Srinivasan, A. (1994). To the international computing community: A new East-West challenge. Tech. rep., Oxford University Computing laboratory, Oxford, UK.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z.

Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 3111–3119). New York, USA: Curran Associates Inc.
Motl, J., & Schulte, O. (2015). The CTU Prague relational learning repository. arXiv:1511.03086.
Muggleton, S. H. (Ed.). (1992). *Inductive logic programming*. London: Academic Press Ltd.
Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*(3–4), 245–286.
Nickel, M., & Kiela, D. (2017). Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pp. 6338–6347.
Nickel, M., Tresp, V., & Kriegel, H. P. (2011). A three-way model for collective learning on multi-relational data. *Proceedings of International Conference on Machine Learning*, *11*, 809–816.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.
Perovšek, M., Vavpetič, A., Cestnik, B., & Lavrač, N. (2013). A wordification approach to relational data mining. In *Proceedings of the international conference on discovery science*, pp. 141–154. Berlin: Springer.
Perovšek, M., Vavpetič, A., Kranjc, J., Cestnik, B., & Lavrač, N. (2015). Wordification: Propositionalization by unfolding relational data into bags of words. *Expert Systems with Applications*, *42*(17–18), 6442–6456.
Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 701–710. ACM.
Plantié, M., & Crampes, M. (2013). Survey on social community detection. In N. Ramzan, R. Zwol, J. S. Lee, K. Clüver, & X. S. Hua (Eds.), *Social media retrieval* (pp. 65–85). London: Springer.
Podpečan, V., Lavrač, N., Mozetič, I., Kralj Novak, P., Trajkovski, I., Langohr, L., et al. (2011). SegMine workflows for semantic microarray data analysis in Orange4WS. *BMC Bioinformatics*, *12*, 416.
Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., & Tang, J. (2018). Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and Node2Vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, WSDM '18, pp. 459–467. ACM.
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.
Ribeiro, L. F., Saverese, P. H., & Figueiredo, D. R. (2017). Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, KDD '17*, pp. 385–394. New York: ACM.
Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM.
Ristoski, P., & Paulheim, H. (2016). Rdf2vec: Rdf graph embeddings for data mining. In P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, & Y. Gil (Eds.), *The semantic web: ISWC 2016* (pp. 498–514). Cham: Springer.
Robnik-Šikonja, M., & Kononenko, I. (2008). Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, *20*(5), 589–600.
Rocktäschel, T., Singh, S., & Riedel, S. (2015). Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 conference of the north American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1119–1129.
Rumelhart, D. E., & McClelland, J. L. (Eds.) (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1: Foundations. MIT Press, Cambridge, MA.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533.
Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, *26*(5), 1651–1686.
Schölkopf, B., & Smola, A. J. (2001). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge: The MIT Press.
Škrlj, B., Kralj, J., Konc, J., Robnik-Šikonja, M., & Lavrač, N. (2019). Deep node ranking: An algorithm for structural network embedding and end-to-end classification. arXiv:1902.03964.
Srinivasan, A. (2007). Aleph manual. http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/.
Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. (1997). Carcinogenesis predictions using ILP. In *Proceedings of the international conference on inductive logic programming*, pp. 273–287. Berlin: Springer.
Srinivasan, A., Vig, L., & Bain, M. (2019). Logical explanations for deep relational machines using relevance information. *Journal of Machine Learning Research*, *20*(130), 1–47.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Štrumbelj, E., & Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, *41*(3), 647–665.

Tang, J., Qu, M., & Mei, Q. (2015a). PTE: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1165–1174. ACM.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015b). LINE: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077.

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22.

Vapnik, V. (1995). *The nature of statististical learning theory*. New York: Springer.

Vavpetič, A., & Lavrač, N. (2011). Semantic data mining system g-SEGS. In *Proceedings of the workshop on planning to learn and service-oriented knowledge discovery (PlanSoKD-11), ECML PKDD conference*, pp. 17–29.

Wang, Q., Wang, B., & Guo, L. (2015). Knowledge base completion using embeddings and rules. In *Proceedings of the 24th international joint conference on artificial intelligence*, pp. 1859–1865.

Wang, Z., Zhang, J., Feng, J., & Chen, Z. (2014). Knowledge graph and text jointly embedding. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1591–1601.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*(12), 2724–2743.

Wu, L. Y., Fisch, A., Chopra, S., Adams, K., Bordes, A., & Weston, J. (2018). Starspace: Embed all the things! In *Proceedings of the 32nd AAAI conference on artificial intelligence*, pp. 5569–5577.

Železný, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*, 33–63.

Zhu, S., Bing, J., Min, X., Lin, C., & Zeng, X. (2018). Prediction of drug–gene interaction by using metapath2vec. *Frontiers in Genetics, 9*.

Žitnik, M., & Leskovec, J. (2017). Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, *33*(14), i190–i198.

Affiliations

Nada Lavrač[1,2] · Blaž Škrlj[3] · Marko Robnik-Šikonja[4]

Nada Lavrač
nada.lavrac@ijs.si

Marko Robnik-Šikonja
marko.robnik@fri.uni-lj.si

[1] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

[2] University of Nova Gorica, Glavni trg 8, 5271 Vipava, Slovenia

[3] International Postgraduate School Jožef Stefan, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

[4] University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana, Slovenia

3.3 Neuro-symbolic Node Embedding

Learning from simpler relational structures, such as weighted graphs $G = (N, E, w)$, has been an ongoing research endeavour in the last years (Perozzi et al., 2014). A prominent technique for general relational learning from such structures revolves around the construction of *node embeddings*. These *real-valued*, low-dimensional representations of nodes, when used as input to a given learner, capture the rich *graph-topological properties* of a given node's neighbourhood, whilst maintaining low dimension and do not require special learning techniques for solving a given task. Currently, most representation learning algorithms that address the problem of *node embedding* operate as *black boxes* (Kipf & Welling, 2017; Veličković et al., 2018), offering relatively little insight into the learning process, but also the content of the learned representations. To address this issue, we hypothesized and implemented an approach that explored whether the *neuro-symbolic* paradigm could be of use to address some of these limitations. The paper relevant to this section is the following one:

> Škrlj, B., Kralj, J., Konc, J., Robnik-Šikonja, M., & Lavrač, N. (2021). Deep node ranking for neuro-symbolic structural node embedding and classification. *International Journal of Intelligent Systems*, 1–30. https://doi.org/https://doi.org/10.1002/int.22651

3.3.1 Key Contributions

We next present the key contributions of the conducted work.

1. A neuro-symbolic algorithm for node embedding achieving state-of-the-art performance.

2. Comparison of symbolic vs. sub-symbolic performance offering insights into suitable representation types for low-resource learning.

3. An improved version of Personalized PageRank with Shrinking which includes *node pivoting*, the process of ranking only with respect to a heuristically selected (ranked) node subspace, demonstrating substantial scalability improvements for larger networks.

4. A simple-to-use Python library, implementing highly optimized versions of the Deep Node Ranking algorithm.

3.3.2 Addressed Hypotheses and Discussion

The main hypotheses concerning the presented contribution address both the scalability aspect – we were interested whether neuro-symbolic representation learning scales better than symbolic for a given task (Hypothesis 4, Section 1.3), as well as the understandability of the relationship between symbolic and sub-symbolic representation learning (Hypothesis 1, Section 1.3). Further, the proposed DNR also represents a strong competitor to existing sub-symbolic approaches (Hypothesis 3, Section 1.3). In particular, we first demonstrated that by performing personalized node ranking with respect to a selected sub-network (and not the whole network), followed by embedding learning, the proposed procedure performs competitively to a symbolic variant which includes the stationary walk distributions with respect to all nodes, but need significantly (asymptotically) less space. To validate this claim, we performed extensive ablation studies on multiple synthetic networks, where consistent space performance improvements were observed for the neuro-symbolic node

embedding (DNR). Further, we demonstrated that the proposed neuro-symbolic node embedding algorithm performs well when the labels are scarce, even if not (on average) as well as the symbolic-only variant. This result indicates that, in terms of performance, the additional compression from the $|N|$ dimensions to d dimensions results in minimal performance loss. Such behaviour is expected, as the neural network-based representation compression is not a *lossless* process.

The second hypothesis concerns *better understanding* of the interplay between ranking-based symbolic node representations and the final, sub-symbolic ones. The proposed DNR is, to our knowledge, the first neuro-symbolic node embedding algorithm that offers a direct exploration of the two learned node spaces, their performance trade-offs and the obtained representations' properties. We demonstrated that the two spaces differ in their compactness; the symbolic (high-dimensional) space consists of very distant representations, with only a small percentage that is close to each other. On the contrary, the neuro-symbolic representations (low-dimensional) are more compact – more representations are close to each other. We also demonstrated that the two representation types differ mostly with respect to the node representations that are close to each other, with the distant node representations being similar for both representation types. We believe this type of study offers a direct exploration of both properties of different representations and, more importantly, their impact on one another; note that DNR first constructs symbolic representations, which are only subsequently distilled with a neural network. Compared to the existing state-of-the-art structural node representation learners, DNR offers controlled interventions into the initial representation learning step with predictable consequences. This property led us to include *node pivoting*, which enables scalable heuristic-based construction of the symbolic representations. As the heuristic addresses the question of selecting the appropriate node subspace, we could, during the development of this heuristic, test different options, for which we could, to some extent, anticipate their effect on the constructed symbolic representation. This type of debugging/exploration would have been harder if considering a black-box learner directly.

The next discussed hypothesis is the third one, focusing on predictive performance. The obtained results indicate that the proposed neuro-symbolic node representation learning algorithm performs *competitively* to the existing state-of-the-art. Statistical evaluations indicate, however, that the differences are not significant, indicating that no clear dominance was observed by any of the existing state-of-the-art approaches. We believe there are two possible reasons for this observation. First, it is possible that the algorithms (the best-performing ones) approached the limit of what is possible to learn from a given network's structure. As each network is most likely not representative of the ground-truth relations and potentially changes, a given network's snapshot was perhaps not sufficient for very good classification. The second reason could be that walk-based methods are not expressive enough in terms of graph-topological features they can capture. Examples of features that are possibly not directly captured via walks include convex subgraphs, graphlets and community-like structures, which can play prominent roles in a given network's behaviour/overall structure.

We finally discuss the explainability of the final set of learners. Note that we adopted the accepted benchmarking scheme that utilizes the logistic regression classifier (LR) for learning to map from the obtained node representations to the designated target space. Should LR be used to learn from the symbolic representation, the importance of individual nodes could be inferred directly based on the final set of coefficients (weights). In any other case, if more sophisticated learners are used, SHAP-based explanations are a viable alternative, as demonstrated elsewhere (Mežnar et al., 2020).

Received: 16 October 2020 | Revised: 22 July 2021 | Accepted: 21 August 2021

DOI: 10.1002/int.22651

RESEARCH ARTICLE

WILEY

Deep node ranking for neuro-symbolic structural node embedding and classification

Blaž Škrlj[1,2] | **Jan Kralj**[1,3] | **Janez Konc**[4] | **Marko Robnik-Šikonja**[5] | **Nada Lavrač**[1,6]

[1]Jožef Stefan Institute, Ljubljana, Slovenia

[2]Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

[3]Cosylab d.o.o., Ljubljana, Slovenia

[4]National Institute of Chemistry, Ljubljana, Slovenia

[5]Faculty of Computer and Information Science, Ljubljana, Slovenia

[6]University of Nova Gorica, Ajdovščina, Slovenia

Correspondence
Blaž Škrlj, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia.
Email: blaz.skrlj@ijs.si

Abstract

Network node embedding is an active research subfield of complex network analysis. This paper contributes a novel approach to learning network node embeddings and direct node classification using a node ranking scheme, coupled with an autoencoder-based neural network architecture. The main advantages of the proposed Deep Node Ranking (DNR) algorithm are competitive or better classification performance, significantly higher learning speed and lower space requirements when compared to state-of-the-art approaches on 15 real-life structural node classification benchmarks. It also enables exploration of the relationship between symbolic and the derived sub-symbolic node representations, offering insights into the learned node space structure. To avoid the space complexity bottleneck in a direct node classification setting, DNR, if needed, computes stationary distributions of personalized random walks from given nodes in mini-batches, scaling seamlessly to larger networks. The scaling laws associated with DNR were also investigated by considering 1,488 synthetic Erdős-Rényi networks, demonstrating its scalability to tens of millions of links.

KEYWORDS

complex networks, deep learning, network node embedding, node classification

1 | INTRODUCTION

Numerous real-world systems consisting of interconnected entities can be represented as complex networks. Analysis of such networks provides insights into the underlying patterns applicable in various practical scenarios, including the discovery of drug targets, modelling of disease outbreaks, author profiling, modelling of transportation and the study of social dynamics.[1]

Modern machine learning approaches applied to complex networks offer intriguing opportunities for developing fast and accurate algorithms that can learn based on the structural topology of a given network. Recently, approaches based on network node embedding[2–4] became prevalent for many common tasks, such as node classification, edge prediction and unsupervised node clustering (community detection). Node embedding refers to the process of learning node representations in a numeric vector format that captures the topological properties of network nodes.[5] Embeddings are useful, as vector representations are suitable for conventional machine learning algorithms capable of addressing numerous tasks, ranging from classification and regression to clustering.

In this study, we propose a new network node embedding and classification algorithm named *Deep Node Ranking (DNR)*, which combines efficient node ranking with the nonlinear approximation power of deep neural networks. The developed framework uses deep neural networks to obtain node embeddings directly from *stationary random walk distributions* produced by random walkers with a restart from individual nodes of interest. Compared to existing methods, DNR is one of the first *neuro-symbolic* node representation learning algorithms, which offers joint construction of low-dimensional latent representations via symbolic (inspectable) node features.

Even though there already exist embedding approaches based on higher-order random walks[2,4] (i.e. random walkers with memory), the stationary distribution of first-order random walkers has not yet been fully explored in a deep learning setting. Widely used methods such as node2vec and struc2vec perform well; however, they do not necessarily scale to larger networks and often require extensive hyperparameter tuning for good performance. Further, these methods mostly learn representations via rather shallow, single latent layer-like optimization schemes, potentially missing the abstraction learning power of deeper neural networks. Finally, in massive networks, not all nodes need to be accounted for during representation learning—information relevant to representing a given node can depend on its relation to a small number of key nodes. We demonstrate that the neuro-symbolic paradigm offers an elegant solution to this problem via ranking-based pivoting (selection of symbolic features before deep learning), scaling to networks comprised of tens of millions of links and tens of thousands of nodes on commodity hardware. This paper also offers ablation studies of DNR's scalability on more than 1,400 synthetic networks of different sizes—this type of analysis is seldom considered in related work.

We showcase the developed algorithm's capabilities on the challenging problems of node classification and network visualization, highlighting its ability to learn and accurately predict node labels *at scale*. Further, we compiled one of the largest collections of node classification data sets and used it for empirical evaluation of the methods. Key contributions of this paper are the following:

1. A fast network embedding algorithm named DNR based on global personalized node ranks. It performs competitively and can be used for a multitude of downstream learning tasks, including node classification, network visualization and similar. The proposed

neuro-symbolic algorithm is also faster than many state-of-the-art embedding algorithms, and scales better.
2. To our knowledge, the proposed node embedding algorithms are for the first time benchmarked against contemporary approaches on such a scale (15 real data sets), as commonly, the algorithms are tested only on a handful of data sets.
3. We conducted an extensive empirical evaluation on 1,488 synthetic networks to study the effects of node pivoting, for which we hypothesized substantially improves scalability.
4. The DNR algorithm performs better than the competing algorithms when the labelled data is scarce (small percentage of labelled nodes).

The remainder of this study is structured as follows. In Section 2, we shortly review the related work on neuro-symbolic representation learning, network node classification and network node ranking. Section 3 presents the proposed DNR network node embedding algorithm that combines deep neural networks with network node ranking. In Section 4, we describe the experimental setting and different non-synthetic complex networks from different domains used in the evaluation, including the newly composed data sets. The experimental results are presented in Section 5. In Section 6 we conclude the work and present the plans for further work.

2 | BACKGROUND AND RELATED WORK

This section presents deep and neuro-symbolic learning preliminaries that describe how algorithms learn from complex networks and what is learned, followed by an overview of node ranking algorithms relevant to this study.

2.1 | Neuro-symbolic representation learning

We first discuss the branch of methods from the fields of deep learning and symbolic learning, referred to as *neuro-symbolic representation learning*. This paradigm of learning has been actively studied for the past 20 years (see d'Avila Garcez and Lamb[6]); however, it resurged recently with many works that demonstrated this paradigm's utility when compared to symbolic/sub-symbolic-only learning. The interest in neuro-symbolic learning, for example, spiked recently[7] by the development of a neuro-symbolic system that partially operates via symbolic and partially via a sub-symbolic space, used to distil human-understandable concepts from images. The recent work on closing the loop between recognition (neural) and reasoning (symbolic)[8] introduced a grammar model as a symbolic bridge between neural perception and symbolic reasoning, alongside a top-down, human-like induction procedure. This study demonstrated that such a combined approach significantly outperforms the conventional reinforcement learning-based baselines. The Microsoft research division (MSR) recently explored the interplay between visual recognition and reasoning.[9] They introduced a framework to isolate and evaluate the reasoning aspect of visual question answering separately from its perception, followed by a calibration procedure that offers an exploration of the relation between reasoning and perception. Further, a neuro-symbolic approach to logical deduction was proposed as *Neural Logic Machines*.[10] This architecture was shown to have inductive logic learning capabilities, which was demonstrated on simple tasks such as sorting. Finally, the two recent approaches from the field of inductive logic programming (ILP) explored the interplay between the logical input structures and how they perform when associated with neural

network-based learning. The Deep Relational Machines[11] were one of the first approaches to showcase the utility of combining the two paradigms. Further, the recent work of Srinivasan et al.[12] explored how Deep Relational Machines can be *explained*, emphasizing that being able to explain what a given pattern discovery/recognition system does is highly relevant in, for example, the field of biomedicine.

In the last years, links between sub-symbolic learning and inductive logic programming were also established. For example, the DeepProbLog system[13] demonstrates how neural predicates could be useful for constructing expressive (and short) programs for complex tasks such as image-based enumeration/computing. Furthermore, links between statistical learning and the neuro-symbolic paradigm were also studied.[14] Finally, recent endeavours in this direction also introduce the notion of stochasticity as a programming component.[15]

Albeit being actively explored, the notion of neuro-symbolic representation learning was, to our knowledge, not yet considered in the context of node representation learning, which is the key focus of this study.

2.2 | Network node classification

Complex networks, representing real-world phenomena such as financial markets, transportation, biological interactions, or social dynamics[1,16,17] often possess interesting properties such as scale invariance, nontrivial partitioning, presence of hub nodes, weakly connected components, heavy-tailed node degree distributions, occurrence of communities, significant motif pattern counts, and so on.[18,19] *Learning from complex networks* considers different aspects of complex networks, for example, network structure and node labels, which are used as inputs to machine learning algorithms to address learning tasks such as link prediction, node classification, and similar.

In this paper we focus on node classification, that is, the problem of classifying nodes into two or more distinct classes. This task is considered as semi-supervised learning, given that the whole network is used to obtain the representations of individual nodes, from which the network node classification model is learned. Information propagation algorithms[20] propagate label information via nodes' neighbours until all nodes are labeled. These algorithms learn in an *end-to-end* manner, meaning that no intermediary representation of a network is first obtained and subsequently used for training a classifier.

Another class of node classification algorithms learns node labels from node embeddings, that is, node representations in vector form.[21] Here, the whole network is first transformed into an information-rich, compact low-dimensional representation (a dense matrix). This representation serves as an input to a plethora of more general machine learning approaches that can be used for node classification.

We distinguish between two main branches of embedding-based learning algorithms, discussed next: graph neural networks and random walk-based learners. Graph neural networks (GNNs), introduced in the recent years, attempt to incorporate a given network's adjacency structure as a part of new neural network layers. Among first such approaches were the graph convolutional networks (GCNs),[22] their generalization with the attention mechanism,[23] and the more recent isomorphism-based variants with provable properties.[24] Treating the adjacency structure as a neural network has also shown promising results.[25] The key characteristic of this branch of methods is their capability to account for *node features* by multiplication of the normalized adjacency matrix as part of a special layer during learning from features. In contrast, if node features are not available, which is the case with the majority of freely available

public data sets, more optimized methods focused on *structure-based learning* are preferred. For example, the LINE algorithm[26] uses the network's *eigendecomposition* to learn a low dimensional network representation, for example, a representation of the network's nodes in 128 dimensions instead of the dimension that matches the number of nodes. Approaches that use random walks to sample the network include DeepWalk[5] and its generalization node2vec.[2] It was recently proven that DeepWalk, node2vec, and LINE can be reformulated as implicit matrix factorization.[27] Furthermore, approaches such as struc2vec[4] demonstrated how more complex, multilayer structure can be compressed into node representations for better performance. Despite many promising approaches developed, a recent extensive evaluation of network embedding techniques[28] suggests that node2vec[2] remains one of the best embedding approaches for the task of structural node classification.

2.3 | Network node ranking

Node ranking algorithms assess the relevance of a node in a network either globally (relative to the whole network) or locally (relative to a subnetwork) by assigning a *score* (i.e. *rank*) to each node in the network. In this study we only consider node ranking algorithms that compute a local relevance score of a node based on its direct neighbourhood. The key such node ranking algorithm is the Personalized PageRank (P-PR) algorithm,[29] sometimes referred to as random walk with restart.[30] Personalized PageRank uses random walks to calculate the relevance of nodes in a network. It obtains the stationary distribution of a random walk that starts at a given node. The P-PR-based approaches were used successfully to study cellular networks, social phenomena,[31] and many other real-world networks.[32] Efficient implementation of P-PR algorithms remains an active research field, for example, the recent bidirectional variation of the P-PR was introduced to speed up the node ranking process.[33] The obtained stationary distribution of a random walk can be used directly for network-based learning tasks, as demonstrated by the HINMINE methodology.[34]

2.4 | Combining node ranking and node representation learning

Exploring the ideas of augmenting learning with ranking was in the recent years explored in the context of graph neural networks. For example, ranking was used to prioritize propagation[35] and to scale graph neural networks.[36] A similar idea was exploited by Xu et al.,[37] where a more efficient propagation scheme was proposed by using node ranking. The proposed DNR algorithm is novel with respect to these works, as it exploits both the fast, parallel personalized node rank computation and the *representation learning* power of deep neural networks.

3 | DEEP NODE RANKING

This section presents the DNR algorithm for neuro-symbolic structural network node embedding and end-to-end node classification (overview shown in Figure 1). The name of the algorithm, DNR, reflects the two main ideas considered: network node ranking step (*symbolic*) and the subsequent deep neural network learning step (*neural*/*sub-symbolic*). In the first step of DNR, personalized node ranks are computed for each node, resulting in Personalized

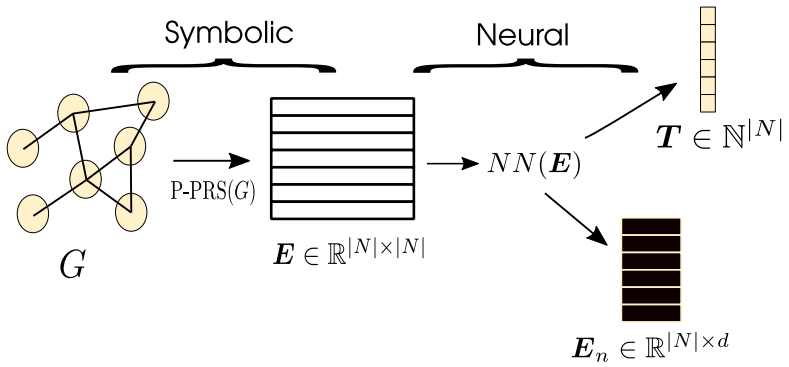


FIGURE 1 Deep Node Ranking algorithm. The symbolic part of the algorithm computes personalized PageRank vectors ($\boldsymbol{E}$), which are subsequently compressed with a neural network (NN) into either a lower dimensional representation ($\boldsymbol{E}_n$), or used for end-to-end classification ($\boldsymbol{T}$). Note that the intermediary symbolic P-PRS based representation remains interpretable (features are nodes) [Color figure can be viewed at wileyonlinelibrary.com]

PageRank with shrinking (P-PRS) vectors. These vectors are symbolic, as each dimension corresponds to a given node. In the second step, the P-PRS vectors are considered by a deep neural network consisting of at least a single dense embedding layer of size equal to the predefined embedding dimension. This embedding is sub-symbolic, as one can no longer interpret the meaning of individual (latent) dimensions. The third, output step, consists either of an output layer with the number of its neurons equal to the number of target classes (top) enabling direct classification of nodes or embeddings (bottom), which correspond to the embedding layer from Step 2. The obtained embeddings can be used for downstream machine learning tasks, such as classification, network visualization, and comparison.

The DNR algorithm, which takes as input a partially labeled complex network, consists of three steps outlined below.

1. Network node ranking results in learned node representations, obtained by using the P-PRS algorithm. This step results in a matrix of P-PRS vectors of dimension $|N|$.
2. Representation distillation. A neural network architecture compresses the prepared personalized PageRank vectors into compact representations (of dimension d).
3. Output phase. The output of the network can be either node classification, that is, direct learning of node labels or a collection of low-dimensional node representations.

3.1 | Node ranking with the personalized pagerank with shrinking algorithm

We first build and upgrade the representation learning idea, introduced in previous work,[34] where node representations are obtained via personalized node ranking. The following description represents a substantial theoretical extension of the original idea, which was further parallelized for the first time in this study. Furthermore, this study introduces node pivoting, which substantially speeds up the personalized ranking time. We consider a version of the PPR[38] algorithm to which we refer to as P-PRS (algorithm 1). This variant of the widely known PPR algorithm produces node representations (or P-PRS vectors) by simulating random walks for each node of the input network. Compared to the network adjacency matrix, P-PRS vectors contain traversal information for each node, reflecting its ranking based on a node's position with respect to a given network's topology.

Algorithm 1: P-PRS: Personalized PageRank with Shrinking

Data: A complex network's adjacency matrix A, with nodes N and edges E, starting node $u \in N$

Parameters : damping factor δ, spread step σ, spread percent τ (default 50%), stopping criterion ϵ

Result: P-PRS$_u$ vector describing stationary distribution of random walker visits with respect to $u \in N$

$A \leftarrow$ toRightStochasticMatrix(A); ▷ Transpose and normalize rows of A

core_vector $\leftarrow [0, \ldots, 0]$; ▷ Initialize zero vector of size |N|

core_vector$[u] \leftarrow 1$; rank_vector $\leftarrow$ core_vector; $v \leftarrow$ core_vector;

steps $\leftarrow 0$; ▷ Shrinking part

nz $\leftarrow 1$; ▷ Number of non-zero P-PRS values

while $nz < |N| \cdot \tau \wedge steps < \sigma$ **do**

  steps $\leftarrow$ steps $+ 1$;

  $v = v + A \cdot v$; ▷ Update transition vector

  nzn $\leftarrow$ nonZero(v); ▷ Identify non-zero values

  if $nzn = nz$ **then**

    shrink $\leftarrow$ True;

    end while;

  end

  nz $\leftarrow$ nzn;

end

if *shrink* **then**

  toReduce $\leftarrow \{i; v[i] \neq 0\}$; ▷ Indices of non-zero entries in vector v

  core_rank $\leftarrow$ core_rank[toReduce]; rank_vector $\leftarrow$ rank_vector[toReduce];

  $A \leftarrow A$[toReduce, toReduce]; ▷ Shrink a sparse adjacency matrix

end

$diff \leftarrow \infty$;

$steps \leftarrow 0$; ▷ Node ranking - power iteration

while $diff > \epsilon \wedge steps < max\_steps$ **do**

  steps $\leftarrow$ steps $+ 1$;

  new_rank $\leftarrow A \cdot$ rank_vector; rank_sum $\leftarrow \sum_i$ rank_vector$[i]$;

  if $rank\_sum < 1$ **then**

    new_rank $\leftarrow$ new_rank $+$ start_rank $\cdot (1 -$ rank_sum$)$;

  end

  new_rank $\leftarrow \delta \cdot$ new_rank $+ (1 - \delta) \cdot$ start_rank;

  diff $\leftarrow \|$rank_vec $-$ new_rank$\|$; ▷ Norm computation

  rank_vec $\leftarrow$ new_rank;

end

if *shrink* **then**

  P-PRS$_u \leftarrow [0, \ldots, 0]$; ▷ Zero vector of dimension |N|

  P-PRS$_u$[toReduce] $\leftarrow$ rank_vec;

else

  P-PRS$_u \leftarrow$ rank_vec

return P-PRS$_u$;

The P-PRS algorithm consists of two main parts:

1. In the first part named the *shrinking step* (Lines 5–20 of Algorithm 1), in each iteration, the walker spreads from nodes with nonzero PageRank values to their neighbours.
2. In the second part of the algorithm, named the *P-PRS computation step* (Lines 23–38 of Algorithm 1), P-PRS vectors corresponding to individual network nodes are computed using the power iteration method (Equation 1).

Shrinking step. In the shrinking step we take into account the following:

- If no path exists between node u (the starting node) and node i, the P-PRS value assigned to node i will be zero.
- The P-PRS values for nodes reachable from u will be equal to P-PRS values calculated for a reduced network G_u, obtained from the original network by only accounting for the subset of nodes reachable from u and connections between them (Lines 6–15 in Algorithm 1).

If the network is strongly connected, G_u will be equal to the original network, yielding no change in performance compared to the original P-PRS algorithm. However, if the resulting network G_u is smaller, the calculation of P-PRS values will be faster as they are calculated on G_u instead of on the whole network. In our implementation, we first estimate if network G_u contains less than 50% (i.e., spread percentage) of nodes of the whole network (Lines 6–14 in Algorithm 1). This is achieved by expanding all possible paths from node i and checking the number of visited nodes in each step. If the number of visited nodes stops increasing after a maximum of 15 steps, we know we have found a network G_u, and we count its nodes. If the number of nodes is still increasing, we abort the calculation of G_u. We limit the maximum number of steps because each step of computing G_u is computationally comparable to one step of the power iteration used in the PageRank algorithm[38] which converges in about 50 steps. Therefore we can considerably reduce the computational load by limiting the number of steps in the search for G_u. Next, in Lines 16–20, the P-PRS algorithm shrinks the personalized rank vectors based on nonzero values obtained as the result of the shrinking step.

P-PRS computation step. In the second part of the algorithm (Lines 23–38), node ranks are computed using the power iteration (Equation 1), whose output consists of P-PRS vectors. An example stationary distribution is shown in Figure 2 for the *Cora* network.

For each node $u \in V$, a feature vector γ_u (with components $\gamma_u(i)$, $1 \leq i \leq |N|$) is computed by calculating the stationary distribution of a random walk, starting at node u. The stationary distribution is approximated using power iteration, where the ith component $\gamma_u(i)^{(k)}$ of approximation $\gamma_u^{(k)}$ is computed in the $k+1$th iteration as follows:

$$\gamma_u(i)^{(k+1)} = \alpha \cdot \sum_{j \to i} \frac{\gamma_u(j)^{(k)}}{d_j^{out}} + (1-\alpha) \cdot v_u(i); \quad k = 1, 2, \ldots \tag{1}$$

The number of iterations k is increased until the visit distribution converges to the final stationary distribution vector (P-PRS value for node i). In the above equation, α is the damping factor that corresponds to the probability that a random walk follows a randomly chosen outgoing edge from the current node rather than restarting its walk. The summation index j runs over all nodes of the network that have an outgoing connection towards i (denoted as

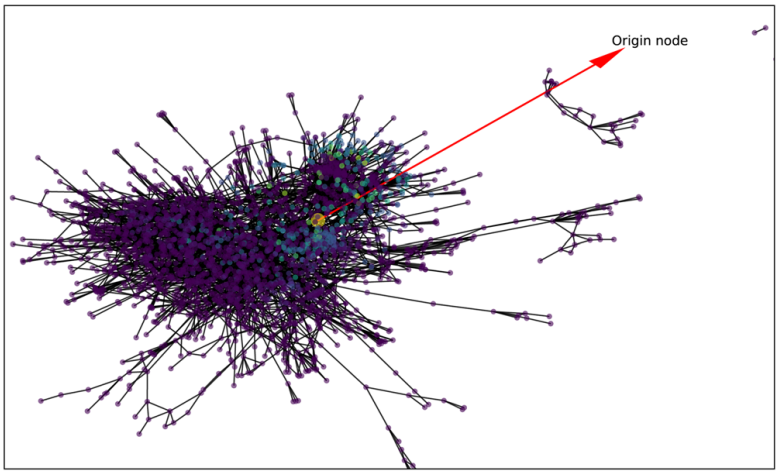


FIGURE 2 Stationary distribution, visualized (*Cora*). The origin node is the large yellow node pointed from. The upper right part of the network contains nodes that a simulated walker (from the origin node) likely ends up at (green-colored nodes) [Color figure can be viewed at wileyonlinelibrary.com]

$j \rightarrow i$ in the sum), and d_j^{out} is the out-degree of node d_j. Term $v_u(i)$ is the restart distribution that corresponds to a vector of probabilities for a walker's return to the starting node u, that is, $v_u(u) = 1$ and $v_u(i) = 0$ for $i \neq u$. This vector guarantees that the walker will jump back to the starting node u in case of restart.*

In a single iteration $(k \rightarrow k+1)$, all stationary distribution vector components $\gamma_u(i)$, $1 \leq i \leq |N|$, are updated which result in the P-PRS vector $\gamma_u^{(k+1)}$. Increasing k thus leads to the $\gamma_u^{(k)}$ eventually converging to the PageRank γ_u of a random walk starting from node u (see Algorithm 1). Equation (1) is optimized by using the *power iteration*, which is especially suitable for large sparse matrices, since it does not rely on spatially expensive matrix factorization to obtain the eigenvalue estimates.†

The P-PRS algorithm simulates a first-order random walk in which no past information is incorporated during obtaining the final stationary distribution. The time complexity of the described P-PRS algorithm with shrinking for k iterations is $\mathcal{O}(|N|(|E| + |N|) \cdot k)$ for the whole network, and $\mathcal{O}((|E| + |N|) \cdot k)$ for a single node.

3.2 | Additional shrinking by rank-based pivoting

We next present an additional step of shrinking explored as part of this study that offers scaling to very large networks. Recall (Algorithm 1) that the PageRank iteration, if the network is reduced, operates on the smaller adjacency matrix indexed via the set of nodes toReduce. This step, as offered in the Algorithm 1, prunes out the nodes unreachable via traversal from the current node. This step preserves the computed vectors' properties, however, it does not guarantee asymptotically faster computation and is largely dependent on a given network's structure. For DNR to scale to very large networks, a more lossy selection scheme can be adopted. Recall the toReduce, the set of nodes that define the final set of iterations that yield a given node's P-PRS-based representation. The idea discussed next defines the set toReduce upfront; the size of this set is parametrized with an integer value p (number of *pivot nodes*). Members of this set are obtained as follows. We hypothesize that two main types of *pivot nodes* need to be preserved in toReduce; namely, the nodes local to the node of interest, but also global nodes (via their relation to the node of interest). To address both concerns, we first

define a given target node u's neighbours as $\mathrm{Ne}(u)$. Next, we define with $\mathrm{argSortDes}(\mathrm{PR}(\mathrm{G}))$ the set of initial nodes, sorted by their PageRank values in descending order. Note that this step takes only $\mathcal{O}(|N|\log|N| + |E|)$ steps, and as such, scales to very large networks. The final ordered set of toReduce is constructed by first including all nodes from the neighbourhood, followed by the global nodes which are not already in the neighbourhood until the set is of cardinality p. We can formally define the set of $\mathrm{toReduce}_u$ pivot nodes as the first p nodes of the ordered union of the node's neighbourhood and the top-ranked nodes, that is,

$$T^u = \mathrm{Ne}(u) \,||\, \mathrm{argSortDes}(\mathrm{PR}(\mathrm{G}))\setminus \mathrm{Ne}(u)$$
$$\mathrm{toReduce}_u = \left\{T_1^u, T_2^u, ..., T_p^u\right\}.$$

Here, T^u is the final ordered set ("||" denotes concatenation), and T_i^u the ith element of that set (obtained with respect to node u). This heuristic selection of pivot nodes implies local neighbourhood for low values of p, and mixed neighbourhood for larger p values. The computed (symbolic) node representations are used as inputs to the subsequent step of (neural) representation compression.

The advantage of the deep neural network architecture discussed in the following section is that it can learn incrementally, from small batches of calculated P-PRS vectors. In contrast, the related HINMINE approach[34] requires that all P-PRS vectors are calculated before learning, which is due to HINMINE using the k-nearest neighbours and Support Vector Machine-based classifiers. This incurs substantial space requirements as the P-PRS vectors for the entire network require $\mathcal{O}(|N|^2)$ space. The DNR algorithm presented here uses a deep neural network instead, which can take as small input batches of P-PRS vectors. Therefore, only a small percentage of vectors need to be computed before the second step of the algorithm (*neural network training*) can begin. This results in improved space and time complexities of the learning process.

3.3 | Node representation learning

In this section, we address the second step of DNR algorithm (outlined in Figure 1)—the incremental compression of batches of personalized PageRank vectors via neural network learning. We next discuss the key formalisms used to describe the two types of learning implemented as part of DNR. We can formalize the key idea underlying DNR as the following mapping:

$$\mathrm{DNR}: \underbrace{\mathbb{R}^{|N|\times|N|}}_{\text{Adjacency matrix}} \xrightarrow{\text{P-PRS}} \underbrace{[0,1]^{|N|\times|N|}}_{\text{P-PRS vectors}} \xrightarrow{\mathrm{NN}_f} \underbrace{\mathbb{R}^{|N|\times d}}_{\text{Node embeddings}},$$

where d represents a *latent dimension*, N is the set of nodes and DNR the mapping approximated by the proposed approach. Note how the second space consists of visit *probabilities* with respect to individual nodes. We will next focus on the two mapping methods displayed in the scheme above; P-PRS and NN_f.

The first mapping (P-PRS) takes as input the network adjacency matrix and, if executed for each node, outputs the same dimensional matrix which contains richer, walk convergence-based information describing individual nodes (instead of only their neighbours). The initial adjacency can be stored as a sparse data structure, requiring only $\mathcal{O}(|E|)$ space. However, the

probability matrix is commonly dense (with the exception of nodes in different components, e.g.), $\mathcal{O}(|N|^2)$ space can already pose a problem to the method's utility. To address this concern, we can consider *batches* of nodes (b) from the first mapping onwards, potentially at no point requiring the full dense $\mathcal{O}(|N|^2)$ matrix. The first mapping adheres to this implementation due to the fact that with respect to individual nodes, P-PRS vectors can be obtained *independently*. We denote with $\mathrm{P-PRS}_b$ a produced *batch* of such vectors. The union of all such batches (forming the set B) can be used to construct the whole probability matrix, that is,

$$[0,1]^{|N|\times|N|} = |_{b\in B}\ \mathrm{P-PRS}_b(A), \tag{2}$$

where A represents the adjacency matrix and $|_b$ the concatenation alongside the first axis. Here, we assume the batches preserve the order of input nodes. The same property holds for transforming the $b \times |N|$-dimensional vectors into $b \times d$ dimensional ones with a *trained* neural network NN_f^b. Similarly to the Equation (2), the final matrix can be written as follows:

$$\mathbb{R}^{|N|\times d} = |_{b\in B}\,\mathrm{NN}_f^b(\mathrm{P-PRS}_b(A)).$$

The two equations above assume projection-ready mapping methods (NN_f and P-PRS). The probability vector computation P-PRS indeed requires no additional training. However, this is not the case for the neural network NN. Note that we denoted with NN_f only the forward pass up to the hidden layer with d outputs—the embedding. To describe the whole process, the missing point remains the *neural network training*. Denoted with NN, we represent a single epoch (forward and backward pass) of training the neural network (for all nodes). If we denote with ω the number of epochs required to train either an autoencoder-like, or an end-to-end architecture (d is in this case the number of classes), DNR requires $\mathcal{O}(\omega \cdot (\mathrm{NN} + |N|(|E| + |N|))$ operations. Furthermore, if the whole probability matrix fits into memory, the product is decoupled, making the full probability matrix computed only once, resulting in complexity $\mathcal{O}(\omega \cdot \mathrm{NN} + |N|(|E| + |N|))$. The initial complexity which recomputes the rank vectors for each batch (b) has, compared to the pre-computed rank matrix version lower space complexity, that is, $\mathcal{O}(b \cdot |N|) << \mathcal{O}(|N|^2)$. This analysis demonstrates that for larger networks, additional computation needs to be performed to maintain the DNR's low space complexity. Finally, the node pivoting scheme similarly reduces the space complexity of the rank matrix computation from $\mathcal{O}(|N|^2)$ to $\mathcal{O}(|N| \cdot p)$ (p is the number of pivot nodes). Similarly, the time complexity reduces linearly ($|N| \to p$) for the ranking step. Hence, the pivoting scheme was hypothesized to improve both space and time-related performances substantially. Having discussed the coupled and the decoupled (memoization) variants of DNR, it is apparent that the low space version will take much longer to compute compared to the memory-intensive version. To fine-tune this to a given hardware setting, DNR is able to *estimate* the approximate RAM utilization by assuming 32-bit floating point precision and takes as hyperparameter an integer number denoting the upper RAM bound. Should this bound be exceeded, the memory-efficient version is considered, and the faster one otherwise. In this study, we set this bound to 16 GB.

3.4 | DNRNet: A neural network architecture

In the previous section, a description of the core feature construction process based on personalized node ranking was described alongside its time and space complexities. We next

discuss in more detail the considered neural network architecture and the training regime, which is also a contribution of this study.

We are interested in compressing the P-PRS-based representation (Equation 2) we henceforth refer to as $\boldsymbol{P}$. The goal of the designed neural network is to compress this representation from dimension $|N|$ to d, in *unsupervised* manner. To achieve this compression, we implemented an autoencoder-like architecture with a forward pass defined as follows (note the indexing):

$$
\begin{aligned}
l_i &= \text{Dropout}\Big(\text{ELU}\Big(\boldsymbol{W_d^T} \cdot \boldsymbol{P} + b_i\Big)\Big) \\
h_1 &= \text{ELU}\Big(\boldsymbol{W_{d1}^T} \cdot l_i + b_{h1}\Big) \\
&\cdots \qquad\qquad\qquad\qquad\qquad\qquad \text{Initial embedding order} \\
h_k &= \text{ELU}\Big(\boldsymbol{W_{dk}^T} \cdot h_{k-1} + b_{hk}\Big) \\
r_1 &= \text{ELU}\Big(\boldsymbol{W_{dk}^T} \cdot l_i + b_{r1}\Big) \\
&\cdots \qquad\qquad\qquad\qquad\qquad\qquad \text{Reversed embedding order} \\
r_k &= \text{ELU}\Big(\boldsymbol{W_{d1}^T} \cdot r_{k-1} + b_{rk}\Big) \\
l_o &= \boldsymbol{W_o^T}\left(\frac{1}{2} \cdot r_k \oplus \ h_k\right).
\end{aligned}
$$

The first part of the architecture projects and activates the input probabilities ($\boldsymbol{P}$) to a lower dimension (d). The ELU activation is defined as follows:

$$
\text{ELU}(x) = \begin{cases} x; & x > 0, \\ \alpha \cdot (e^x - 1); & x \leq 0. \end{cases}
$$

The parameter α was set to 1 throughout this study. The inner part of the architecture consists of multiple same-dimensional (d) layers, which refine the representation. The final layer projects the refined representation back to the initial dimension ($|N|$). The key component is the regularization (dropout) before the embedding layers, as it notably improved the architecture's stability during design. The loss function used is the Smooth L1 Loss defined as follows:

$$
\mathcal{L}_n = \begin{cases} \frac{1}{2} \cdot (x_n - y_n)^2/\beta; & |x_n - y_n| < \beta, \\ |x_n - y_n| - \frac{1}{2} \cdot \beta; & \text{otherwise.} \end{cases}
$$

Here, x_n represents the prediction, y_n the actual value and β a parameter (set to 1.0 in this study). The loss is averaged on the batch level. The key novelty of the proposed neural network-based compression is not the architecture but the way *forward passes are conducted*. We impose an additional constraint on the intermediary representations by implementing the forward pass so that it includes multiplication *in both ways* across the hidden layers (note the shared parameters—there is no weight duplication). This means that each forward pass incorporates a scenario where the first hidden layer is first, but also last in the forward pass (inverted latent space); with this, we enforce reverse consistency, as *all* intermediary representations are used to obtain the final embedding. This is possible due to the symmetric nature of the activation-dense

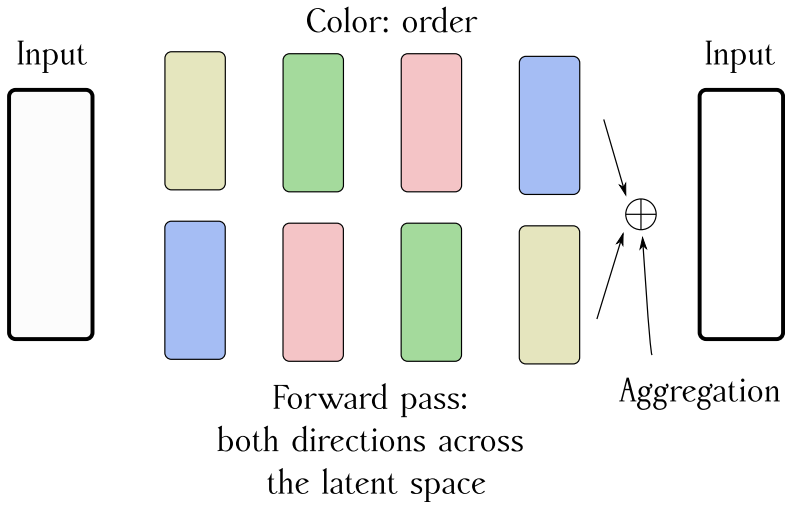


FIGURE 3 The forward pass with the inclusion of reversed latent spaces [Color figure can be viewed at wileyonlinelibrary.com]

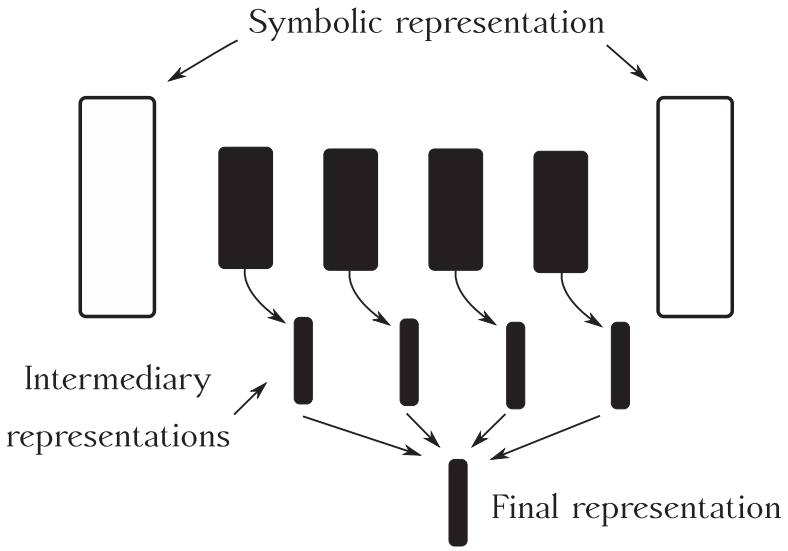


FIGURE 4 DNRNet's representation construction process. The final representation ($\boldsymbol{E}$) is obtained as an aggregate of *all* relevant intermediary layers

layers—inverting the order during the forward pass amplifies the effect of different hidden layers with the same type of output. When inspecting the following scheme note that $\oplus$ denotes the Hadamard summation (elementwise). Note also that such inverse projections during the same forward pass are possible because the dimensionalities of the intermediary representations are all the same (d). A schematic overview of this idea is shown in Figure 3.

Lastly, we discuss how the final representations (node embeddings) are obtained. Recall that $h_1, ..., h_k$ represent the outputs of the intermediary embedding layers. The final node representations ($\boldsymbol{E}$) are obtained by performing Hadamard summation across these intermediary representations and dividing with the number of intermediary hidden outputs, that is,

$$\boldsymbol{E} = k^{-1} \cdot \bigoplus_i \boldsymbol{h}_i.$$

Schematic overview of this process is shown in Figure 4. The main reason such multispace aggregation is conducted is that the information used for reconstructing the origin rank space is

likely distributed across all hidden layers, implying that by considering, for example, only the last layer, valuable parts of the final representation could be lost. This idea was inspired by how representations are obtained from contextual language models.[39]

4 | DATA SETS AND EXPERIMENTAL SETTING

This section first describes the data sets used, the experimental setting and the DNR implementations tested together with their hyperparameters, followed by a description of the compared baseline approaches.

4.1 | Data sets

We evaluated the proposed approach on 15 real-world complex networks, three of them introduced in this study, which is one of the largest collections of complex networks for the task of node classification. The *Homo Sapiens* (proteome),[40] POS tags,[41] and Blogspot data sets[42] are used in the same form as in Grover and Leskovec.[2]

The *Homo sapiens* data set represents a subset of the human proteome, that is, a set of interacting proteins. The subnetwork consists of all proteins for which biological states are known.[43] The goal is to predict protein function annotations. The POS data set represents part-of-speech tags obtained from Wikipedia—a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump.[41] Thus, different POS tags are predicted. The Blogspot data set represents a social network of bloggers (Blogspot website).[42] The labels represent bloggers' interests inferred through the metadata provided by the bloggers. The CiteSeer citation network consists of scientific publications classified into one of the six classes (categories).[44] The Cora citation network consists of scientific publications classified into one of seven classes (categories).[44] The E-commerce network is a heterogeneous network connecting buyers with different products. As DNR and the compared baseline algorithms operate on homogeneous networks, the E-commerce network was transformed to a homogeneous network before learning using a term frequency weighting scheme.[34] The created edges represent mutual purchases of two persons, that is, two customers are connected if they purchased an item from the same item category. We refer the interested reader to Kralj et al.,[34] for a detailed description of the data set and its transformation to a homogeneous network. The two-class values being predicted correspond to the buyers' gender. The *film*, *squirrel*, *chameleon*, *wisconsin*, *texas,* and *cornell* data sets are based on a recent study about geometric deep learning.[45] Given that some of the data sets have features, and the purpose of this paper is structure-only learning, instead of neglecting the feature spaces, we converted them into weights between nodes as follows. If the cardinality of the feature spaces of a given node was the same for all nodes, we computed the weights as inverse Euclidean distances with one added to the denominator (similarities). If the features were sets, we computed the weights as cardinalities of the intersection sets between pairs of nodes.

One of the contributions of this study are also three novel node classification data sets, which we constructed as follows. Two data sets are related to Bitcoin trades.[46] The two networks correspond to transactions within two different platforms, namely Bitcoin OTC and Bitcoin Alpha. Each edge in this network represents a transaction along with an integer score

TABLE 1 Networks used in this study and their basic statistics

Name	#Classes	#Nodes	#Edges	Mean deg	CC	CCoef	Density
cornell	4	183	280	3.06	1	0.17	0.0168
texas	4	183	295	3.22	1	0.20	0.0177
wisconsin	4	251	466	3.71	1	0.21	0.0149
ions	12	1969	16,092	16.35	326	0.53	0.0083
chameleon	4	2277	31,421	27.60	1	0.48	0.0121
cora	7	2708	5278	3.90	78	0.24	0.0014
citeseer	6	3327	4676	2.81	438	0.14	0.0008
Bitcoin_alpha	20	3783	14,124	7.47	5	0.18	0.0020
Homo_sapiens	50	3890	38,739	19.92	35	0.15	0.0051
POS	40	4777	92,517	38.73	1	0.54	0.0081
squirrel	4	5201	198,493	76.33	1	0.42	0.0147
Bitcoin	20	5881	21,492	7.31	4	0.18	0.0012
film	4	7600	14,056	3.70	1975	0.04	0.0005
Blogspot	39	10,312	333,983	64.78	1	0.46	0.0063
ecommerce_tf	2	29,999	178,608	11.91	8304	0.48	0.0004

denoting trust in the range $[-10, 10]$ (zero-valued entries are not possible). We reformulate this as a classification problem by collecting the trust values associated with individual nodes and considering them as target classes. The resulting integer values can thus belong to one of the 20 possible classes. Note that more than a single class is possible for an individual node, as we did not attempt to aggregate trust scores for individual nodes.

The *ions* data set is based on the recently introduced protein-ion binding site similarity network.[47] The network was constructed by structural alignment using the ProBiS family of algorithms[48–50] where all known protein-ion binding sites were considered. The obtained network was pruned for structural redundancy as described in Škrlj et al.[47] Each node corresponds to one of 12 possible ions, and each weighted connection corresponds to the ion-binding site similarity between the two considered binding sites. Overall, this is to date one of the largest collections of structure-only node classification benchmark data sets.

The considered data sets are summarized in Table 1. In the table, CC denotes the number of connected components. The clustering coefficient measures how nodes in a graph tend to cluster together and is computed as the ratio between the number of closed triplets and the number of all triplets. The network density is computed as the number of actual connections divided by all possible connections. The mean degree corresponds to the average number of connections of a node. Links to data sets, along with other material presented in this paper are discussed in Section 7.

Furthermore, to test the DNR's scalability, we created 1488 Erdős-Rényi networks in node number range from 2500 to 35,000 in the increments of 1000 with different seeds and the probability parameter set to 0.05 (sparser networks).

4.2 | Experimental setting

In this section, we describe the experimental setting used to evaluate the proposed method against the existing baselines.

There are two main evaluation aspects relevant to this paper; investigating the quantitative performance of embeddings on a given downstream task and computation time. To assess the classification performance, we use the same evaluation scheme as in related work on node classification.[2,5,26,51] Here, as all methods for node embedding construction are unsupervised, an embedding is first constructed and used as input to a logistic regression-based classification scheme suitable for multiclass and multilabel classification tasks.

We repeated the classification experiments five times and averaged the results to obtain stable performance estimates with corresponding variabilities. The performance of trained classifiers was evaluated by using micro and macro F_1 scores, as these two measures are used in the majority of related node classification studies.[2,5,26,51]

Due to many classifier comparisons, we utilize the Friedman test with Nemenyi *post hoc* correction to compute the statistical significance of the differences. The results are visualized as critical difference diagrams, where *average ranks* of individual algorithms according to scores across all data set splits are presented.[52] The selected algorithms are also compared via the Bayesian hierarchical *t*-test[53] with a prior value of $\rho = 0.8$ and rope region value set to 2%. All experiment repetitions were used for posterior sampling.

All experiments were conducted on a machine with 64 GB RAM, 6 core Intel(R) Core(TM) i7-6800K CPU @ 3.40GH with a Nvidia 1080 GTX GPU. As the maximum amount of RAM available for all approaches was 64 GB, the run is marked as unsuccessful, should this amount be exceeded. Further, we gave each algorithm at most 5 h for learning the embeddings and subsequent classification. We selected these constraints as the networks used are of medium size, and if a given method cannot work on these networks, it will not scale to larger networks; for example, social networks with millions of nodes and tens of millions, or even billions of edges, without substantial adaptation of the method. The unsuccessful runs were replaced with a random embedding. Node ranking was implemented by using sparse matrices from the SciPy module[54] and the PyTorch library.[55]

4.3 | DNR implementations

We implemented the following variants of DNR, each emphasizing a different aspect of the algorithm.

The DNR represents a default DNR implementation with no node pivoting, two hidden layers, trained for at most 100 epochs with the stopping criterion of five epochs. The learning rate was set to 0.01 and adaptively decreased throughout the training. The upper memory bound was set to 16 GB, meaning that networks that would require more space would be computed incrementally, on the fly, reducing the space requirements but increasing the computation time. The final embedding dimension was for this and all other embedding-based methods set to 128 (as also seen in related work). The DNR4 architecture includes four hidden layers instead of two, and DNR8 eight hidden layers. The DNRPH is a DNR variant with the pivoting node number set to $|N|/2$. The DNRPQ to $|N| \cdot 0.75$ and DNRPM to $\sqrt{|N|}$. All pivot number estimates were rounded to the nearest integer. Finally, we implemented the symbolic-only learner we refer to as DNR-symbolic, which outputs the matrix of personalized rank vectors (symbolic part of the full DNR).

The P-PRS algorithm parameters (constant throughout all experiments) were set as follows. ϵ, the error bound, which specifies the end of an iteration, was set to 10^{-6}. *Max steps* is the number of maximum steps allowed during one iteration, was set to 100,000 steps. *Damping factor* is the probability that a random walker continues at a given step and was set to 0.5. *Spread step* and is the number of iteration steps allowed for the shrinking part and was set to 10. *Spread percent* is the maximum percentage of the network to be explored during shrinking and was set to 50%.

4.4 | The baseline approaches

We tested the proposed approach against different baselines outlined below. The baselines were selected as they are currently considered as either very weak (random) or strong (node2vec, struc2vec). All approaches apart from label propagation are node embedding algorithms. For label propagation, the same data splits were used for classification evaluation as for the logistic regression when considering embedding-based learning.

- *node2vec*.[2] This algorithm maximizes the likelihood of preserving network neighbourhoods of nodes. This is achieved via biased random walk sampling. This algorithm is considered a strong baseline for structure-only learning.
- *struc2vec*.[4] This algorithm uses a hierarchy-like structure to measure node similarity at different scales and constructs a multilayer graph to encode structural similarities and generate structural context for nodes. It remains one of the key approaches capable of including information on structural similarity.
- *Label propagation (LP)*.[56] Label propagation is a well-known algorithm for node classification. It operates by incrementally sending information from the neighbouring nodes to the unlabeled nodes, eventually reaching an equilibrium and yielding the final set of predictions for the masked part of the network.
- *GraphWave*[57] is a method that represents each node's local network neighbourhood via a low-dimensional embedding by leveraging spectral graph wavelet diffusion patterns. This is one of the more scalable methods considered in this study.[‡]
- *Graph neural networks* (GAT and GCN).[23] We trained the models with the stopping criterion of 100 epochs for up to 1000 epochs. Due to unstable performance, we report the best performance (epoch scoring best). Further, as GATs were not initially implemented for multilabel classification, we extended them, so they minimize binary cross-entropy and output a sigmoid-activated space with the same dimension as the number of targets (the multiclass version does not work for such problems). As this branch of models operates with additional features assigned to nodes, and the considered benchmark data sets do not possess such features, we used the identity matrix of the adjacency matrix as the feature space, thus expecting suboptimal performance. This Algorithm was shown to outperform other variants of graph neural networks such as the GCNs[22] which were also considered under the same training regime. The PyTorch-Geometric library was used for the two GNN-based baselines.[58]
- *Random baseline* which is a random float matrix $\in[0, 1]^{|N|\times d}$.

For all baselines, suggested default hyperparameter settings were used (either taken from papers or from the codebases). Similarly, default configurations of DNR variants were used to ensure fair comparisons (no additional hyperparameter optimization was conducted across data sets). Thus, we evaluated out-of-the-box performance – additional hyperparameter

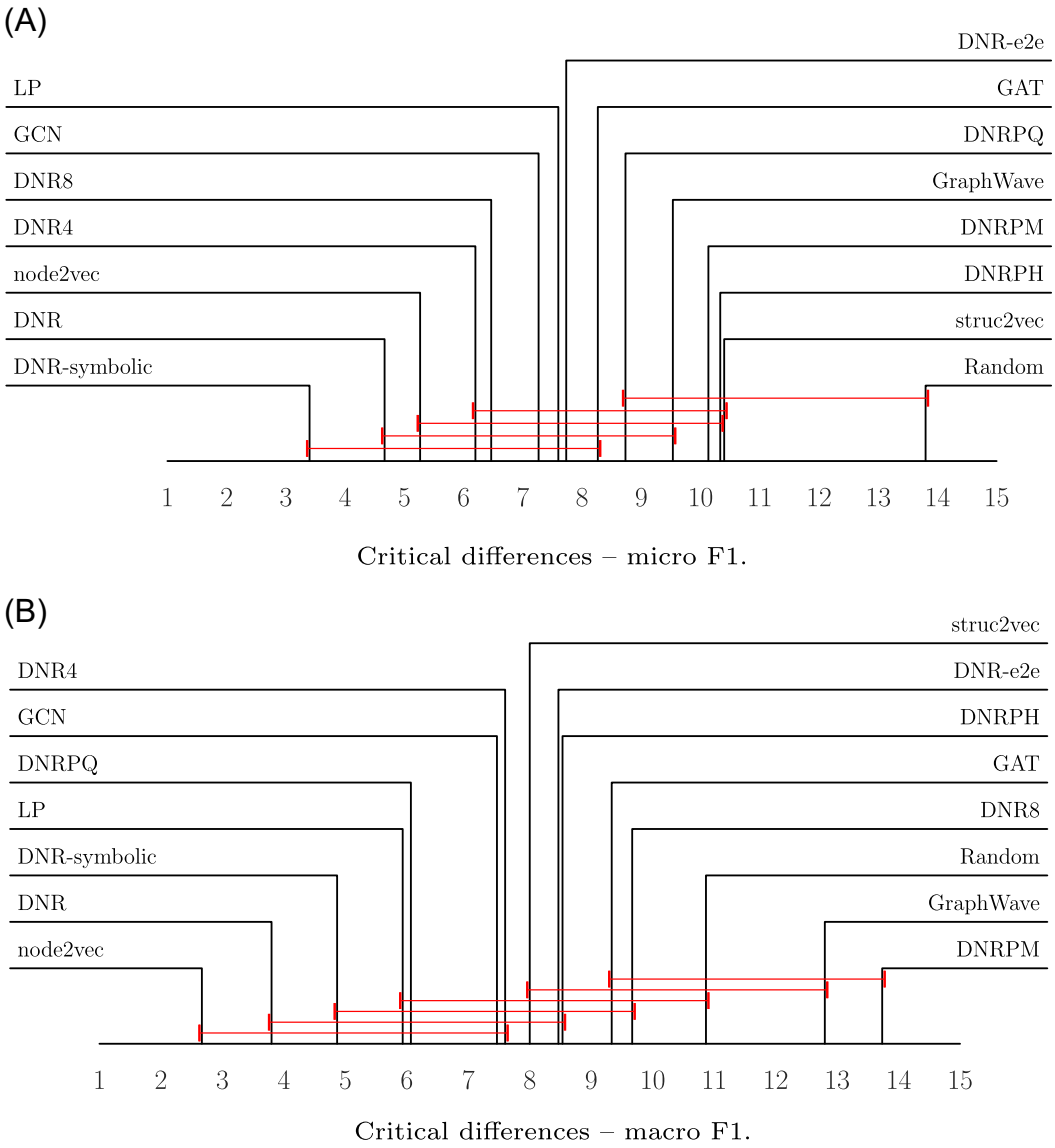


FIGURE 5 Overview of classification performance—critical difference diagrams. (A) Critical differences—micro-F1 and (B) critical differences—macro-F1 [Color figure can be viewed at wileyonlinelibrary.com]

tunning could significantly increase the training time and render some of the methods inapplicable even at the mid-scale networks considered in this study.

5 | RESULTS

In this section, we present the empirical results and discuss their qualitative as well as quantitative aspects. We first present the results for the node classification task, followed by a qualitative evaluation of the proposed DNR algorithm for the task of node visualization.

5.1 | Classification performance

We first present the results of classification experiments. In Figure 5 the reader can observe the critical difference plots of micro and macro F1 scores aggregated across all data sets. It can be observed that similar algorithms dominate with respect to both scores; node2vec, DNR,

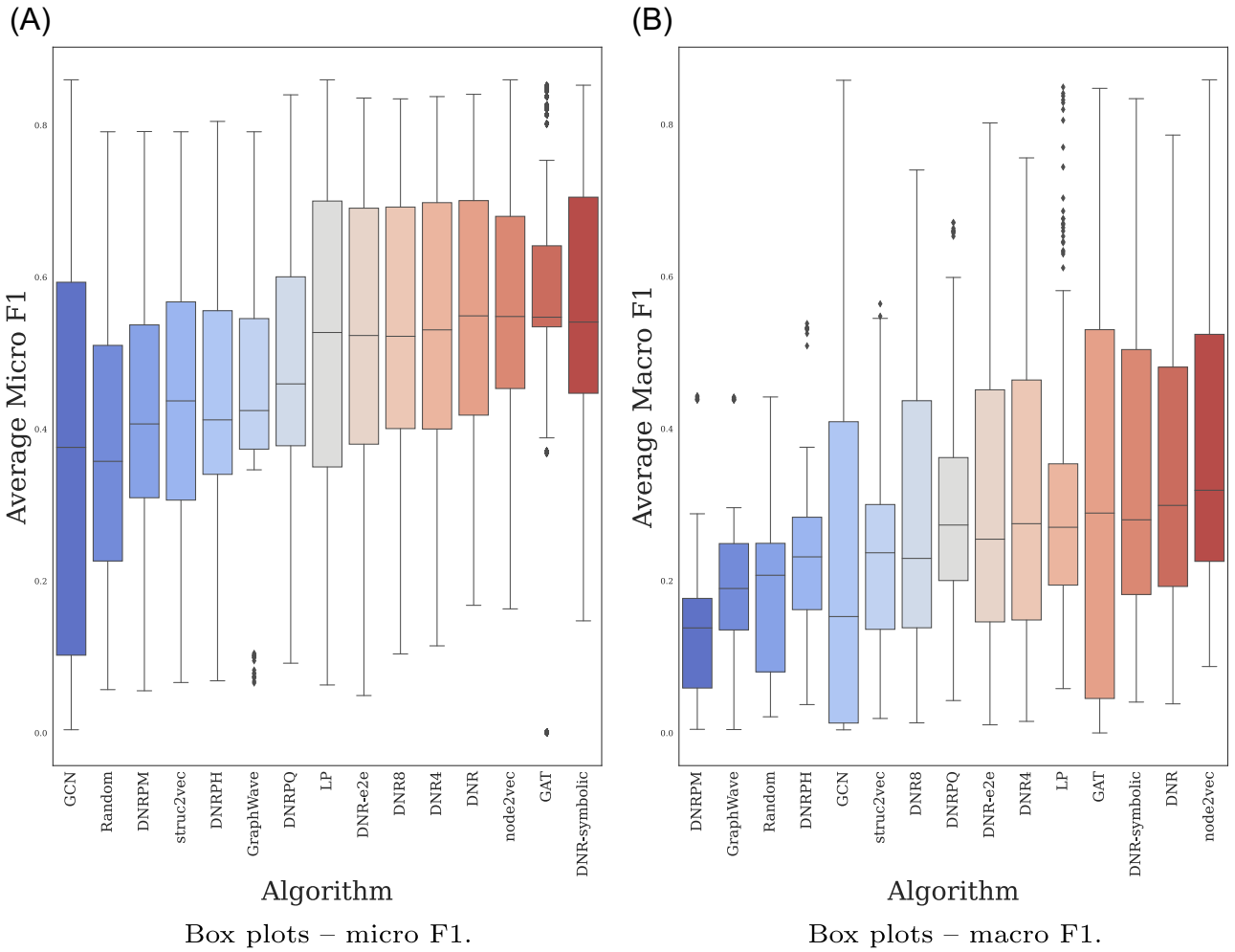


FIGURE 6 Micro- and macro-F1 performance distributions for considered algorithms. (A) Box plots—micro-F1 and (B) box plots—macro-F1 [Color figure can be viewed at wileyonlinelibrary.com]

DNR-symbolic, and label propagation are amongst the best-performing ones. The differences between the best performers are insignificant, as demonstrated via statistical analysis (CD diagrams).[52] Next, GraphWave and DNRPM underperform w.r.t. macro-F1 (Figure 6). This observation could be due to multiple factors, ranging from GraphWave's hyperparameter sensitivity, poor performance on small networks (too much information is lost) or similar. Among the best performing algorithms are either the default DNR variant with two hidden layers, DNR-symbolic or node2vec. The DNR variant implementing a deeper neural network (DNR8) performed worse than the more shallow versions, indicating overfitting (highly likely especially for smaller networks). The DNRPM variant, which uses a substantially reduced version of the adjacency matrix for rank computation, performed better than random when considering micro-F1. However, it was overall among the worst performing variants of DNR. The DNRPQ variant performed better, indicating that node pruning can have a substantial impact on the final representation—too low values of p indicate detrimental effects on the final performance. The end-to-end variant of DNR performed competitively w.r.t. micro-F1; however, it performed worse when considering macro-F1. This result indicates over-fitting, but also the method's potential sensitivity to classification of nodes in smaller networks (see the appendix materials for detailed scores on smaller networks). Note that the proposed end-to-end DNR out-performed the two GNN-based baselines. Current results indicate that structure-only learning is harder for GCN and GAT-based models—either due to higher possibility of overfitting or due to space complexity which arises if considering the attention-based

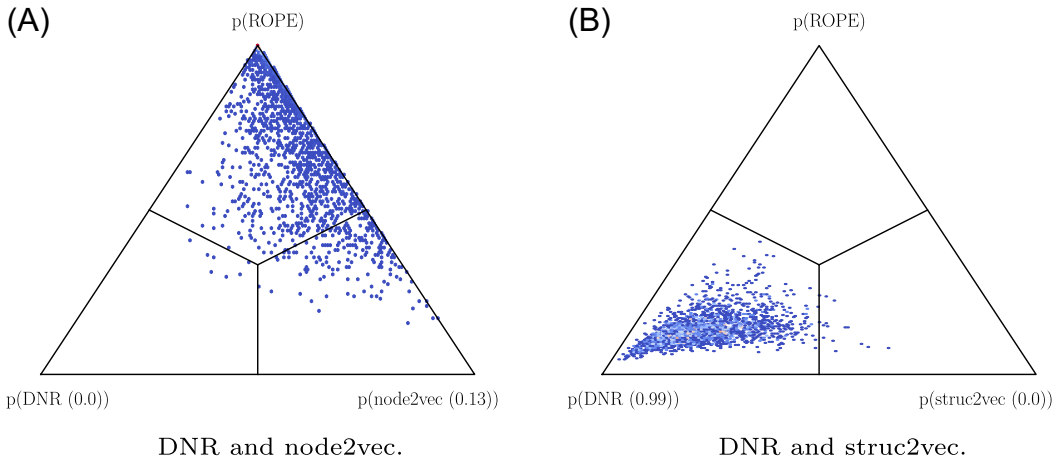


FIGURE 7 Bayesian comparison of selected algorithm pairs. (A) DNR and node2vec and (B) DNR and struc2vec [Color figure can be viewed at wileyonlinelibrary.com]

architecture. Current results indicate that the neural network in neuro-symbolic DNR variants, as expected, acts as a compression layer, losing some of the expressive power of the origin rank space at the cost of being more efficient space-wise. Bayesian comparison of default DNR with node2vec and struc2vec confirms the results obtained via frequentist analysis and is shown in Figure 7. The numbers denote the posterior probability estimates (higher is better). Note the insignificant difference between DNR and node2vec (most of the density is in the rope region), but the significant difference (as also confirmed via frequentist analysis) between DNR and struc2vec. Overall, the Bayesian analysis confirms the findings supported by the classical analysis.

5.2 | Execution time analysis

Overview of the execution times is shown in Figure 8. We present overall execution times followed by the per-data set execution times.

It can be observed that the fastest DNR variants perform up to two orders of magnitude faster than, for example, struc2vec and node2vec. Given that, for example, DNR offers very similar performance, this result serves as a strong case for using DNR-based embeddings, especially on larger networks. Further, note that the execution time includes both embedding construction and classification, rendering the random baseline slower than label propagation (logistic regression is the bottleneck in this case).

5.3 | Number of pivot nodes and scalability

We finally present the results on synthetic Erdős-Rényi networks, where the effect of the number of pivot nodes on the execution time was studied. The main result is shown in Figure 9. The result indicates that the number of pivot nodes can reduce the execution time by more than an order of magnitude—with no pivoting nodes, DNR's execution time increases observably faster. More detailed results displaying the dependence with the node and link numbers are shown in Figure 9A,B. The complexity, if considering pivoting, remains linear with respect to the number of nodes (constant $d = p$ instead of $d = |N|$ in the symbolic step of DNR). Without pivoting, the complexity increases substantially, which is problematic for larger networks. Consistent improvement with

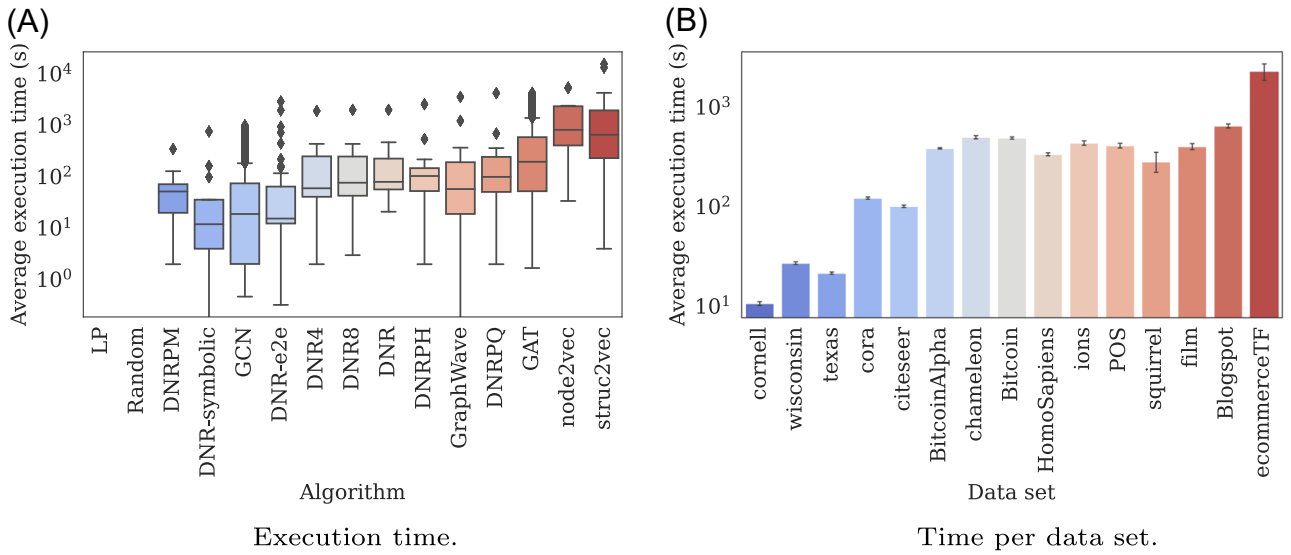


FIGURE 8 Execution time analysis. The proposed DNR algorithm performs substantially faster than struc2vec and node2vec. (A) Execution time and (B) time per data set [Color figure can be viewed at wileyonlinelibrary.com]

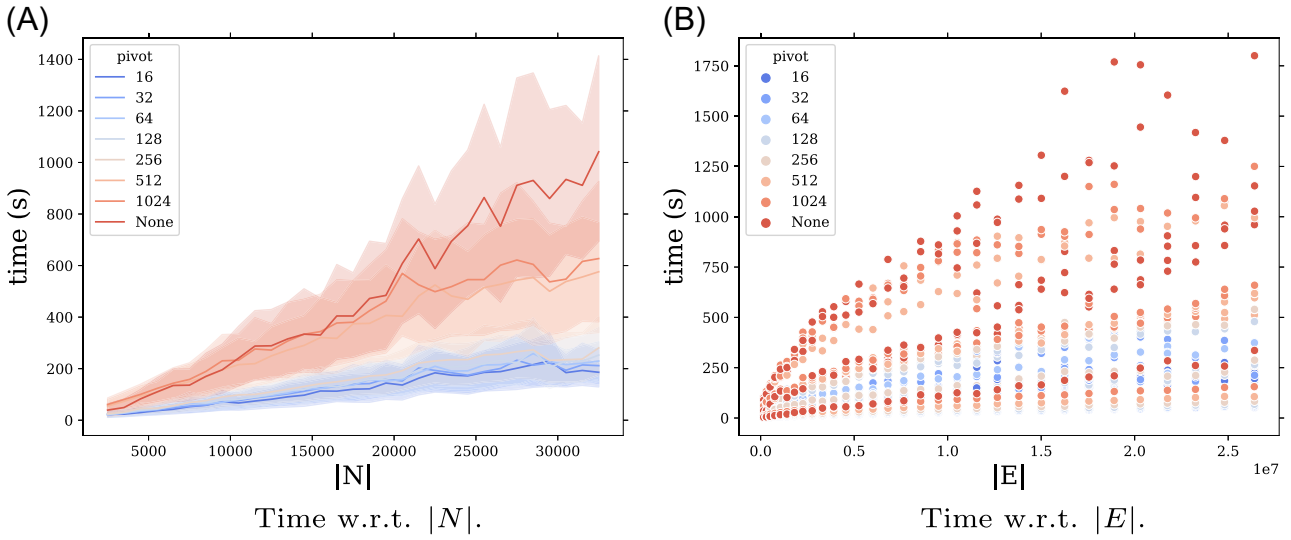


FIGURE 9 Execution time with respect to the number of pivot nodes. Smaller number of pivot nodes induces substantially faster execution times. (A) Time w.r.t. |N| and (B) Time w.r.t. |E| [Color figure can be viewed at wileyonlinelibrary.com]

respect to the number of pivot nodes was observed—the lower the pivot number, the faster the overall process. This observation confirms our theoretical analysis which indicated that substantial improvements could be observed, especially for larger networks. The results indicate that for larger networks comprised of tens of millions of edges, the pivoting-based solutions could offer more than an *order of magnitude faster* embedding construction. For completion, tabular results summarised in this section are available as Appendix A.

5.4 | Performance in a low-data regime

One of the main limitations of many existing node classification algorithms is their performance when only a small portion of a given network is labelled. We next present the

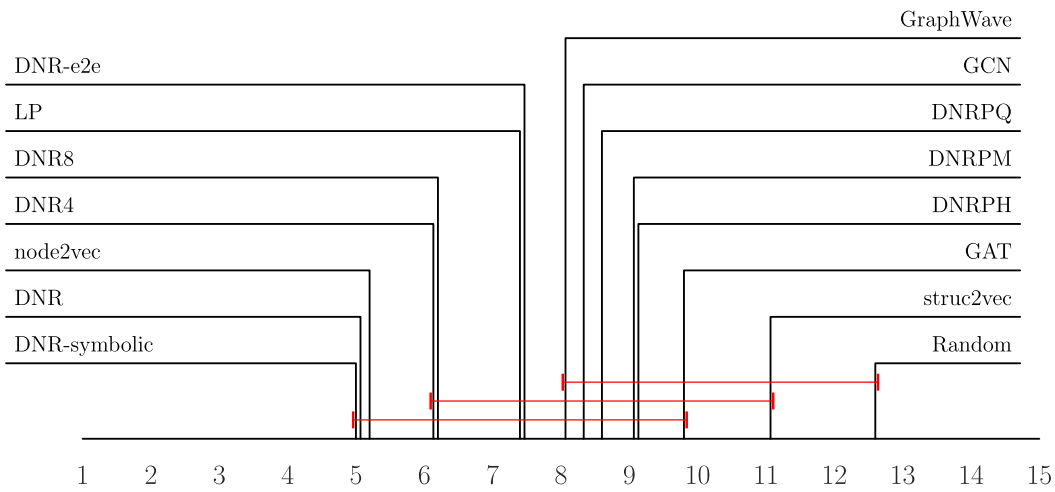


FIGURE 10 Micro-F1 performance when the labelled data is scarce (10%) [Color figure can be viewed at wileyonlinelibrary.com]

DNR's behaviour when considering only 10% of the labelled data in Figure 10. The experiment indicates that two of the DNR variants perform well when only a relatively few labelled data are available. This result potentially indicates the link between using a symbolic (global) rank space instead of using the more local, sampled walks, indicating that neuro-symbolic node representation learning has exciting potential for low-resource learning. Note that for larger data sets, this amount of labelled nodes can already be at the limit of what can be learned on a given commodity hardware setup, rendering DNR potentially relevant for larger data sets.

5.5 | Network visualization

We next demonstrate how DNR's results can be compressed to two dimensions with UMAP[59] to visualize a given network. By considering 10 nearest neighbours with the minimum distance parameter set to 0.5, we obtained the visualization (colored by node labels) as shown in Figure 11.

The visualization shows distinct clustering patterns which to some extent also correspond to the label space (colors). Note that this representation was obtained in unsupervised manner, hence some variability with respect to label-position assignment is expected. A prominent use for this type of visualizations is when inspection of potentially interesting, structurally similar units is investigated via an overlay of additional information. This visualization was, alongside the embedding to 2D, computed in a matter of seconds.

5.6 | Comparing symbolic and sub-symbolic representations

The proposed DNR's neuro-symbolic capabilities render it open for exploration of all intermediary representations and the relations between them. For the Cora network, we first computed node representations with DNR-symbolic ($d = |N|$) and DNR ($d = 128$) and investigated the distances between representations of individual nodes. For each node representation, we computed the cosine distance and normalized all values in the matrix by subtracting the

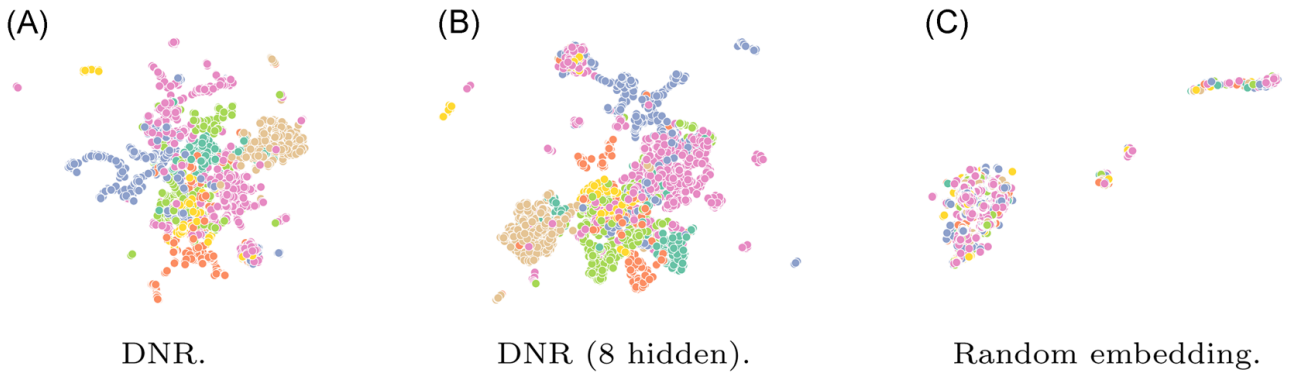


FIGURE 11 Embedding visualization with overlaid labels (Cora). The random embedding is added as a reference of a non-structure-preserving projection visualization. (A) DNR; (B) DNR (8 hidden); and (C) Random embedding [Color figure can be viewed at wileyonlinelibrary.com]

minimum and dividing with the difference between the maximum and minimum to ensure a more fair comparison with respect to the distance bias for individual representations. The result is shown in Figure 12. We observe the following. The symbolic representation comparison matrix (a) mostly consists of entries indicating very high distances between a given embedding pair (intense red color $\Rightarrow$ higher distance). This observation indicates that representations in high-dimensional spaces (in this case $d = |N|$) are far apart. The exceptions (similar nodes) are in the upper left part (blue). On the contrary, many more nodes are closer if we consider the more compact node representations obtained via the DNR algorithm (b). This result indicates that the neural network compresses the space (as expected), yielding fewer node representations that are distant from the others (red strips in the matrix). The final representation (c) represents the difference between the representations—blue color in this case represents similar representations. The reader can observe that distant node representations obtained by DNR are relatively close to the ones obtained by DNR-symbolic (blue horizontal and vertical strips). The red squares in the upper left part, however, indicate node similarities that were not amplified by DNR-symbolic, but with DNR. This type of ablation is possible only for neuro-symbolic representation learners, and is to our knowledge one of the first of its kind for the considered task.

6 | DISCUSSION AND CONCLUSIONS

In this study, we presented DNR, a methodology for scalable neuro-symbolic node embedding and direct end-to-end classification based on a given network's structure. In extensive empirical evaluation, we demonstrated DNR's competitive performance and superior scalability on multiple real-life and synthetic benchmark problems.

The proposed methodology offers one of the first neuro-symbolic node representation learners—the initial node features that are interpretable are compressed with a novel neural network architecture (DNRNet). Albeit the resulting representations are latent and non-interpretable to a human, the input to obtaining such a representation can be manipulated in a symbolic manner (e.g., effects of node removal), offering a simple-to-use testbed for investigating the effects of different *structural interventions* on a given network. Extensive empirical evaluation indicates that symbolic features are highly competitive. However, they could be impractical to compute, rendering the proposed neuro-symbolic variant of DNR highly useful for many contemporary network-based learning tasks. Furthermore, DNRNet performs

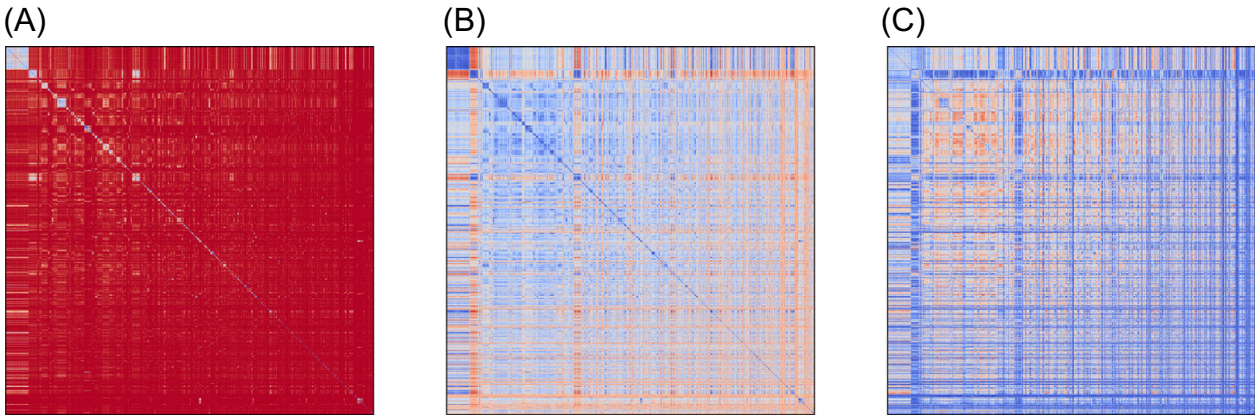


FIGURE 12 Representation space comparison. Each cell in (A) and (B) represents cosine distance between a given embedding pair. Cells in (C) represent the absolute difference values. The rows and columns correspond to the same nodes for all three sub-figures. Red represents high values and blue low ones. (A) DNR-symbolic ($d = |N|$); (B) DNR ($d = 128$); (C) abs(DNR - symbolic − DNR) [Color figure can be viewed at wileyonlinelibrary.com]

well in low-data regimes, which was an interesting finding—we expected that the symbolic-only variant would dominate in such settings.

We demonstrated that neuro-symbolic approaches could scale better than purely sub-symbolic ones (e.g., node2vec or struc2vec), indicating that not all interpretability is necessarily sacrificed for good performance. We demonstrated that out-of-the-box DNR implementation performs competitively and in terms of micro-F1, better than state-of-the-art, and further, it offers at least an order of magnitude speedup. By introducing the concept of *node pivoting*, we demonstrate that DNR can scale to very large networks with tens of millions of links—a scale where many other considered methods do not operate well without specialized hardware. We confirmed the findings related to algorithms' performance with frequentist and Bayesian analysis. As Bayesian analysis was previously not conducted in such evaluation settings, we believe further work which will investigate the suitability (and scalability) of this branch of tests for network-related tasks is an interesting research direction.

In terms of further work related to the proposed algorithm, we see the following main directions. First, the effects of studying different pivoting schemes could offer better trade-offs between efficiency and performance. Next, by considering GPU-based implementations of the power iteration considered for computing the stationary random walk distributions, we believe additional speedups could be observed. Neuro-symbolic node ranking offers the direct study of the effects of *perturbing* specific, for example, nodes, and observing the properties of the resulting low-dimensional representations. Such *structural interventions* potentially offer a more native explanation mechanism compared to *post-hoc* approximation schemes considered in contemporary machine learning. Finally, we plan to explore the scalability of DNR across multiple machines by sharing the input network and performing the rankings only locally. Such implementation could scale to much larger networks than considered by current state-of-the-art approaches.

Finally, this study demonstrates that deeper neural networks are suitable models for structure-only learning, albeit, as shown, in a neuro-symbolic setting. Current results indicate that deeper neural networks are possible representation learners for the considered task, and potentially offer superior performance (unless overfitting takes place). A promising direction that would offer additional improvements is also the automatic development of neural network architectures via neuroevolution.

ACKNOWLEDGMENTS

The work of the first author was funded by the Slovenian Research Agency through a young researcher grant (BŠ). The work of other authors was supported by the Slovenian Research Agency (ARRS) core research programs P2-0103 and P6-0411, and research projects J7-7303, L7-8269, and N2-0078 (financed under the ERC Complementary Scheme). The work was also supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media).

DATA AVAILABILITY STATEMENT

The DNR and the data sets allowed to be shared w.r.t. their licenses will be freely accessible at https://github.com/SkBlaz/DNR.

ENDNOTES

*If the binary vector was composed exclusively of ones, the iteration would compute the global PageRank vector, and Equation (1) would reduce to the standard PageRank iteration.

†The power iteration (Equation 1) converges exponentially, that is, the error is proportional to α^k, where α is the damping factor and k is the iteration number.

‡Python 3 implementation used: https://github.com/benedekrozemberczki/GraphWaveMachine

ORCID

Blaž Škrlj https://orcid.org/0000-0002-9916-8756
Jan Kralj https://orcid.org/0000-0002-5770-9139
Janez Konc https://orcid.org/0000-0003-0160-3375
Marko Robnik-Šikonja https://orcid.org/0000-0002-1232-3320
Nada Lavrač https://orcid.org/0000-0002-9995-7093

REFERENCES

1. Benson AR, Gleich DF, Leskovec J. Higher-order organization of complex networks. *Science*. 2016; 353(6295):163-166.
2. Grover A, Leskovec J. node2vec: Scalable feature learning for networks. In: Krishnapuram B, Shah M, Smola AJ, Aggarwal CC, Shen D, Rastogi R, eds. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 13–17, 2016. ACM; 2016:855-864. https://doi.org/10.1145/2939672.2939754
3. Žitnik M, Leskovec J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*. 2017;33(14):i190-i198.
4. Ribeiro LF, Saverese PH, Figueiredo DR. *struc2vec*. Learning node representations from structural identity. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, Association for Computing Machinery; 2017:385-394. https://doi.org/10.1145/3097983.3098061
5. Perozzi B, Al-Rfou R, Skiena S. Deepwalk: online learning of social representations. In: Macskassy SA, Perlich C, Leskovec J, Wang W, Ghani R, eds. The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'14, August 24–27, 2014. ACM; 2014:701-710. https://doi.org/10.1145/2623330.2623732
6. d'Avila Garcez A, Lamb LC. Neurosymbolic AI: the 3rd wave; 2020. https://arxiv.org/abs/2012.05876
7. Mao J, Gan C, Kohli P, Tenenbaum JB, Wu J. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In: *7th International Conference on Learning*

Representations, ICLR 2019, New Orleans, LA, May 6–9, 2019, OpenReview.net; 2019. https://openreview.net/forum?id=rJgMlhRctm

8. Li Q, Huang S, Hong Y, Chen Y, Wu YN, Zhu S. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In: *Proceedings of the 37th International Conference on Machine Learning*, ICML 2020, July 13–18, 2020, Virtual Event, Vol. 119 of Proceedings of Machine Learning Research, PMLR; 2020:5884-5894. http://proceedings.mlr.press/v119/li20f.html
9. Amizadeh S, Palangi H, Polozov A, Huang Y, Koishida K. Neuro-symbolic visual reasoning: Disentangling "visual" from "reasoning". In: *Proceedings of the 37th International Conference on Machine Learning*, ICML 2020, July 13-18, 2020, Virtual Event, Vol. 119 of Proceedings of Machine Learning Research, PMLR; 2020: 279-290. http://proceedings.mlr.press/v119/amizadeh20a.html
10. Dong H, Mao J, Lin T, Wang C, Li L, Zhou D. Neural logic machines. In: *7th International Conference on Learning Representations*, ICLR 2019, New Orleans, LA, May 6–9, 2019, OpenReview.net; 2019. https://openreview.net/forum?id=B1xY-hRctX
11. Lodhi H. Deep relational machines. In: *Proceedings, Part II, of the 20th International Conference on Neural Information Processing—Volume 8227*, ICONIP 2013, Springer-Verlag, Berlin, Heidelberg; 2013:212-219. https://doi.org/10.1007/978-3-642-42042-9_27
12. Srinivasan A, Vig L, Bain M. Logical explanations for deep relational machines using relevance information. *J Mach Learn Res*. 2019;20(130):1-47.
13. Manhaeve R, Dumancic S, Kimmig A, Demeester T, Raedt LD. Deepproblog: neural probabilistic logic programming. In: Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, NeurIPS 2018, December 3–8, 2018, Montréal,Canada; 2018:3753-3763. https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html
14. Raedt LD, Dumancic S, Manhaeve R, Marra G. From statistical relational to neuro-symbolic artificial intelligence. In: Bessiere C, ed. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI 2020; 2020:4943-4950. https://doi.org/10.24963/ijcai.2020/688
15. Winters T, Marra G, Manhaeve R, Raedt LD. Deepstochlog: neural stochastic logic programming. 2021. https://arxiv.org/abs/2106.12574
16. Nowzari C, Preciado VM, Pappas GJ. Analysis and control of epidemics: a survey of spreading processes on complex networks. *IEEE Control Syst*. 2016;36(1):26-46.
17. Le D-H. A novel method for identifying disease associated protein complexes based on functional similarity protein complex networks. *Algorith Mol Biol*. 2015;10(1):14.
18. Costa LdF, Rodrigues FA, Travieso G, VillasBoas PR. Characterization of complex networks: a survey of measurements. *Adv Phys*. 2007;56(1):167-242.
19. Van Der Hofstad R. Random graphs and complex networks; 2016. https://www.cambridge.org/core/books/random-graphs-and-complex-networks/AA6578462E56868A874B083E082C9FE7
20. Zhu X, Ghahramani Z. Learning from labeled and unlabeled data with label propagation. Technical Report; 2002.
21. Cui P, Wang X, Pei J, Zhu W. A survey on network embedding. *IEEE Trans Knowl Data Engineering*.
22. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: *5th International Conference on Learning Representations*, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net; 2017. https://openreview.net/forum?id=SJU4ayYgl
23. Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: *6th International Conference on Learning Representations*, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings, OpenReview.net; 2018. https://openreview.net/forum?id=rJXMpikCZ
24. Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? In: *7th International Conference on Learning Representations*, ICLR 2019, New Orleans, LA, May 6–9, 2019, OpenReview.net; 2019. https://openreview.net/forum?id=ryGs6iA5Km
25. Hamilton WL, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R, eds. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, December 4–9,

2017; 2017:1024-1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html
26. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q. LINE: large-scale information network embedding. In: Gangemi A, Leonardi S, Panconesi A, eds. *Proceedings of the 24th International Conference on World Wide Web*, WWW 2015, Florence, Italy, May 18–22, 2015. ACM; 2015:1067-1077. https://doi.org/10.1145/2736277.2741093
27. Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: Chang Y, Zhai C, Liu Y, Maarek Y, eds. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM 2018, Marina Del Rey, CA, February 5-9, 2018. ACM; 2018:459-467. https://doi.org/10.1145/3159652.3159706
28. Goyal P, Ferrara E. Graph embedding techniques, applications, and performance: a survey, arXiv preprint. https://www.sciencedirect.com/science/article/abs/pii/S0950705118301540
29. Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the web. Technical Report, Stanford InfoLab; 1999.
30. Tong H, Faloutsos C, Pan J-Y. Fast random walk with restart and its applications. In: *Proceedings of the Sixth International Conference on Data Mining*; 2006: 613-622.
31. Halu A, Mondragón RJ, Panzarasa P, Bianconi G. Multiplex pagerank. *PLOS One*. 2013;8(10):e78293.
32. Yu X, Lilburn TG, Cai H, Gu J, Korkmaz T, Wang Y. Pagerank influence analysis of protein-protein association networks in the malaria parasite plasmodium falciparum. *Int J Computat Biol Drug Design*. 2017;10(2):137-156.
33. Lofgren P, Banerjee S, Goel A. Personalized pagerank estimation and search: A bidirectional approach. In: Bennett PN, Josifovski V, Neville J, Radlinski F, eds. *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, San Francisco, CA, February 22–25, 2016. ACM; 2016:163-172. https://doi.org/10.1145/2835776.2835823
34. Kralj J, Robnik-Šikonja M, Lavrač N. HINMINE: heterogeneous information network mining with information retrieval heuristics. *J Intell Inform Syst*. 2017:1-33.
35. Klicpera J, Bojchevski A, Günnemann S. Predict then propagate: Graph neural networks meet personalized pagerank. In: *7th International Conference on Learning Representations*, ICLR 2019, New Orleans, LA, May 6–9, 2019, OpenRevew.net; https://openreview.net/forum?id=H1gL-2A9Ym
36. Bojchevski A, Klicpera J, Perozzi B, Blais M, Kapoor A, Lukasik M, Günnemann S. Is pagerank all you need for scalable graph neural networks? In: *ACM KDD, MLG Workshop*; 2019.
37. Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K, Jegelka S. Representation learning on graphs with jumping knowledge networks. In: Dy JG, Krause A, eds. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan*, Stockholm, Sweden, July 10–15, 2018, Vol. 80 of Proceedings of Machine Learning Research. PMLR; 2018:5449-5458. https://proceedings.mlr.press/v80/xu18c.html
38. Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the web, Technical Report, Stanford InfoLab; 1999.
39. Reimers N, Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics*; 2019. https://arxiv.org/abs/1908.10084
40. Stark C, Breitkreutz B-J, Chatr-Aryamontri A. The biogrid interaction database: 2011 update. *Nucleic Acids Res*. 2010;39(suppl_1):D698-D704.
41. Mahoney M. Large text compression benchmark. http://mattmahoney.net/dc/text.html
42. Zafarani R, Liu H. Social computing data repository at ASU; 2019. http://socialcomputing.asu.edu
43. Stark C, Breitkreutz B-J, Reguly T, Boucher L, Breitkreutz A, Tyers M. BioGRID: A general repository for interaction datasets. *Nucleic Acids Res*. 2006;34(suppl_1):D535-D539.
44. Lu Q, Getoor L. Link-based classification. In: Fawcett T, Mishra N, eds. Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003. AAAI Press: Washington, DC; 2003:496-503. http://www.aaai.org/Library/ICML/2003/icml03-066.php
45. Pei H, Wei B, Chang KC, Lei Y, Yang B. Geom-gcn: Geometric graph convolutional networks. In: *8th International Conference on Learning Representations*, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020, OpenReview.net; 2020. https://openreview.net/forum?id=S1e2agrFvS

46. Kumar S, Spezzano F, Subrahmanian V, Faloutsos C. Edge weight prediction in weighted signed networks. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE; 2016:221-230.
47. Škrlj B, Kunej T, Konc J. Insights from ion binding site network analysis into evolution and functions of proteins. *Mol Inform*. https://onlinelibrary.wiley.com/doi/10.1002/minf.201700144
48. Konc J, Janežič D. Probis-ligands: a web server for prediction of ligands by examination of protein binding sites. *Nucleic Acids Res*. 2014;42(W1):W215-W220.
49. Konc J, Depolli M, Trobec R, Rozman K, Janežič D. Parallel-ProBiS: fast parallel algorithm for local structural comparison of protein structures and binding sites. *J Computat Chem*. 2012;33(27):2199-2203.
50. Konc J, Skrlj B, Erzen N, Kunej T, Janezic D. Genprobis: web server for mapping of sequence variants to protein binding sites. *Nucleic Acids Res*. 2017; 45(W1): W253-W259.
51. Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: Chang Y, Zhai C, Liu Y, Maarek Y, eds. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM 2018, Marina Del Rey, CA, February 5–9, 2018. ACM; 2018:459-467. https://doi.org/10.1145/3159652.3159706
52. Demšar J. Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res*. 2006;7:1-30.
53. Benavoli A, Corani G, Demšar J, Zaffalon M. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *J Mach Learn Res*. 2017;18(1):2653-2688.
54. Virtanen P, Gommers R, Oliphant TE, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nat Methods*. 2020;17(3):261-272. https://doi.org/10.1038/s41592-019-0686-2
55. Paszke A, Gross S, Massa F, et al. Pytorch: an imperative style, high-performance deep learning library. In: Wallach HM, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox EB, Garnett R, eds. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, NeurIPS 2019, December 8–14, 2019. Vancouver, BC, Canada; 2019: 8024-8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
56. Zhu X, Ghahramani Z. Learning from labeled and unlabeled data with label propagation. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.8280
57. Donnat C, Zitnik M, Hallac D, Leskovec J. Learning structural node embeddings via diffusion wavelets. In: Guo Y, Farooq F, eds. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD 2018, London, UK, August 19–23, 2018. ACM; 2018: 1320-1329. https://doi.org/10.1145/3219819.3220025
58. Fey M, Lenssen JE. Fast graph representation learning with PyTorch geometric. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*; 2019. https://arxiv.org/abs/1903.02428
59. McInnes L, Healy J, Melville J. Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint. https://joss.theoj.org/papers/10.21105/joss.00861

How to cite this article: Škrlj B, Kralj J, Konc J, Robnik-Šikonja M, Lavrač N. Deep node ranking for neuro-symbolic structural node embedding and classification. *Int J Intell Syst*. 2021;1-30. https://doi.org/10.1002/int.22651

APPENDIX A: TABULAR RESULTS WITH DEVIATIONS

In this section, we present the performance results for micro- and macro-F1 scores. The first table represents the micro-F1 scores, followed by a table showing macro-F1 scores. The runs marked with NaN reached the time-out point (did not finish).

TABLE A1 Micro-F1

Data set setting	Bitcoin	Bitcoin Alpha	Blogspot	Homo Sapiens	POS	Cha-meleon	Citeseer	Cora	Cornell	Ecom-merceTF	Film	Ions	Squirrel	Texas	Wisconsin
DNR	0.71 (0.01)	0.71 (0.01)	0.2 (0.01)	0.2 (0.01)	0.45 (0.02)	0.56 (0.04)	0.55 (0.02)	0.78 (0.02)	0.52 (0.06)	0.83 (0.0)	0.34 (0.01)	0.64 (0.03)	0.41 (0.03)	0.59 (0.07)	0.53 (0.03)
DNR-e2e	0.69 (0.01)	0.7 (0.01)	0.09 (0.01)	0.07 (0.01)	0.41 (0.01)	0.45 (0.04)	0.57 (0.03)	0.77 (0.04)	0.49 (0.12)	0.83 (0.01)	0.35 (0.02)	0.66 (0.03)	0.39 (0.02)	0.61 (0.06)	0.49 (0.08)
DNR-symbolic	0.72 (0.01)	0.71 (0.01)	0.28 (0.04)	0.22 (0.03)	0.45 (0.02)	0.59 (0.04)	0.62 (0.04)	0.8 (0.04)	0.54 (0.04)	0.83 (0.01)	0.36 (0.01)	0.67 (0.04)	0.5 (0.05)	0.58 (0.07)	0.49 (0.04)
DNR4	0.71 (0.01)	0.71 (0.01)	0.15 (0.02)	0.15 (0.02)	0.4 (0.0)	0.52 (0.03)	0.55 (0.01)	0.74 (0.02)	0.48 (0.05)	0.83 (0.01)	0.37 (0.01)	0.63 (0.04)	0.42 (0.02)	0.58 (0.05)	0.51 (0.03)
DNR8	0.7 (0.01)	0.71 (0.0)	0.14 (0.02)	0.12 (0.01)	0.4 (0.0)	0.5 (0.02)	0.54 (0.02)	0.71 (0.03)	0.53 (0.02)	0.82 (0.01)	0.37 (0.01)	0.58 (0.05)	0.41 (0.01)	0.55 (0.04)	0.52 (0.05)
DNRPH	0.68 (0.01)	0.68 (0.01)	0.15 (0.01)	0.08 (0.01)	0.4 (0.01)	0.34 (0.01)	0.26 (0.01)	0.41 (0.01)	0.54 (0.03)	0.8 (0.0)	0.34 (0.01)	0.48 (0.01)	0.39 (0.01)	0.53 (0.03)	0.5 (0.03)
DNRPM	0.69 (0.0)	0.7 (0.0)	0.1 (0.0)	0.06 (0.01)	0.4 (0.01)	0.37 (0.01)	0.2 (0.01)	0.31 (0.0)	0.52 (0.02)	0.78 (0.0)	0.37 (0.01)	0.43 (0.01)	0.41 (0.01)	0.53 (0.02)	0.51 (0.07)
DNRPQ	0.69 (0.01)	0.69 (0.01)	0.2 (0.01)	0.11 (0.01)	0.41 (0.01)	0.42 (0.02)	0.4 (0.02)	0.6 (0.02)	0.51 (0.03)	0.83 (0.01)	0.34 (0.02)	0.58 (0.03)	0.37 (0.01)	0.54 (0.03)	0.44 (0.06)
GAT	0.54 (0.0)	0.55 (0.02)	NaN	0.0 (0.0)	0.35 (0.12)	0.6 (0.04)	0.66 (0.07)	0.82 (0.03)	0.58 (0.05)	NaN	0.37 (0.0)	0.51 (0.01)	NaN	0.65 (0.05)	0.53 (0.03)
GCN	0.58 (0.02)	0.57 (0.01)	0.11 (0.01)	0.03 (0.01)	0.05 (0.05)	0.61 (0.04)	0.69 (0.05)	0.83 (0.03)	0.56 (0.04)	0.82 (0.0)	0.37 (0.01)	0.19 (0.12)	0.44 (0.01)	0.63 (0.06)	0.51 (0.02)
GraphWave	0.69 (0.0)	0.7 (0.0)	0.1 (0.0)	0.07 (0.01)	0.4 (0.01)	0.37 (0.02)	0.37 (0.01)	0.42 (0.02)	0.5 (0.03)	0.78 (0.0)	0.37 (0.01)	0.53 (0.03)	0.41 (0.01)	0.54 (0.04)	0.47 (0.04)
LP	0.7 (0.0)	0.71 (0.01)	0.22 (0.02)	0.06 (0.0)	0.07 (0.0)	0.4 (0.02)	0.64 (0.06)	0.82 (0.04)	0.54 (0.05)	0.72 (0.02)	0.35 (0.01)	0.69 (0.06)	0.41 (0.01)	0.56 (0.02)	0.42 (0.06)
Random	0.65 (0.02)	0.66 (0.03)	0.07 (0.01)	0.06 (0.0)	0.37 (0.03)	0.3 (0.03)	0.18 (0.0)	0.21 (0.02)	0.47 (0.06)	0.78 (0.01)	0.33 (0.02)	0.36 (0.02)	0.34 (0.04)	0.5 (0.05)	0.42 (0.03)
node2vec	0.69 (0.01)	0.69 (0.02)	0.36 (0.03)	0.21 (0.02)	0.52 (0.02)	0.58 (0.02)	0.58 (0.02)	0.83 (0.03)	0.51 (0.04)	0.83 (0.01)	0.34 (0.02)	0.65 (0.03)	0.43 (0.03)	0.57 (0.05)	0.51 (0.02)
struc2vec	0.67 (0.02)	0.68 (0.01)	0.08 (0.01)	0.08 (0.0)	0.38 (0.02)	0.54 (0.03)	0.27 (0.01)	0.3 (0.02)	0.45 (0.04)	0.78 (0.01)	0.34 (0.01)	0.46 (0.02)	0.41 (0.02)	0.56 (0.04)	0.53 (0.1)

TABLE A2 Macro-F1

Data set setting	Bitcoin	BitcoinAlpha	Blogspot	HomoSapiens	POS	Chameleon	Citeseer	Cora	Cornell	EcommerceTF	Film	Ions	Squirrel	Texas	Wisconsin
DNR-e2e	0.27 (0.01)	0.27 (0.01)	0.01 (0.0)	0.03 (0.01)	0.04 (0.0)	0.39 (0.07)	0.5 (0.03)	0.74 (0.06)	0.22 (0.04)	0.66 (0.02)	0.17 (0.02)	0.2 (0.02)	0.18 (0.03)	0.31 (0.04)	0.32 (0.04)
DNR-symbolic	0.31 (0.02)	0.29 (0.01)	0.11 (0.03)	0.16 (0.04)	0.06 (0.01)	0.57 (0.04)	0.56 (0.04)	0.79 (0.05)	0.21 (0.05)	0.67 (0.04)	0.19 (0.02)	0.22 (0.05)	0.39 (0.08)	0.23 (0.06)	0.27 (0.04)
DNR4	0.31 (0.02)	0.28 (0.01)	0.03 (0.01)	0.09 (0.02)	0.04 (0.0)	0.48 (0.03)	0.48 (0.01)	0.72 (0.03)	0.19 (0.02)	0.65 (0.01)	0.15 (0.01)	0.2 (0.03)	0.25 (0.03)	0.26 (0.05)	0.34 (0.04)
DNR8	0.3 (0.02)	0.28 (0.01)	0.03 (0.01)	0.06 (0.01)	0.04 (0.0)	0.46 (0.02)	0.47 (0.02)	0.68 (0.05)	0.19 (0.03)	0.63 (0.01)	0.14 (0.0)	0.17 (0.02)	0.2 (0.03)	0.22 (0.04)	0.26 (0.03)
DNRPH	0.29 (0.01)	0.27 (0.01)	0.06 (0.01)	0.05 (0.01)	0.04 (0.0)	0.22 (0.01)	0.21 (0.01)	0.33 (0.02)	0.26 (0.03)	0.53 (0.01)	0.2 (0.01)	0.14 (0.02)	0.24 (0.01)	0.26 (0.04)	0.33 (0.03)
DNRPM	0.27 (0.01)	0.26 (0.01)	0.01 (0.0)	0.02 (0.0)	0.03 (0.0)	0.14 (0.0)	0.06 (0.0)	0.07 (0.0)	0.17 (0.0)	0.44 (0.0)	0.14 (0.0)	0.06 (0.0)	0.15 (0.0)	0.18 (0.02)	0.17 (0.01)
DNRPQ	0.3 (0.01)	0.28 (0.01)	0.08 (0.01)	0.09 (0.01)	0.05 (0.0)	0.37 (0.02)	0.36 (0.02)	0.58 (0.02)	0.24 (0.03)	0.66 (0.01)	0.21 (0.01)	0.22 (0.05)	0.25 (0.0)	0.28 (0.05)	0.35 (0.08)
GAT	0.04 (0.0)	0.05 (0.01)	NaN	0.0 (0.0)	0.01 (0.0)	0.59 (0.04)	0.62 (0.07)	0.81 (0.04)	0.35 (0.16)	NaN	0.15 (0.01)	0.08 (0.02)	NaN	0.36 (0.08)	0.36 (0.05)
GCN	0.08 (0.01)	0.08 (0.01)	0.01 (0.0)	0.01 (0.0)	0.02 (0.01)	0.6 (0.04)	0.64 (0.05)	0.82 (0.03)	0.32 (0.13)	0.64 (0.0)	0.15 (0.01)	0.08 (0.03)	0.34 (0.01)	0.35 (0.07)	0.38 (0.05)
GraphWave	0.27 (0.01)	0.26 (0.01)	0.0 (0.0)	0.03 (0.0)	0.04 (0.0)	0.21 (0.03)	0.25 (0.01)	0.21 (0.01)	0.21 (0.02)	0.44 (0.0)	0.14 (0.0)	0.14 (0.01)	0.14 (0.0)	0.18 (0.02)	0.22 (0.03)
LP	0.28 (0.01)	0.29 (0.01)	0.1 (0.02)	0.06 (0.0)	0.06 (0.0)	0.33 (0.03)	0.61 (0.06)	0.81 (0.04)	0.24 (0.05)	0.67 (0.02)	0.2 (0.01)	0.32 (0.04)	0.24 (0.02)	0.26 (0.02)	0.28 (0.09)
Random	0.27 (0.01)	0.26 (0.01)	0.02 (0.0)	0.05 (0.01)	0.04 (0.0)	0.25 (0.02)	0.15 (0.0)	0.13 (0.01)	0.21 (0.03)	0.44 (0.0)	0.22 (0.01)	0.08 (0.0)	0.22 (0.01)	0.23 (0.05)	0.23 (0.02)
node2vec	0.33 (0.01)	0.3 (0.01)	0.23 (0.03)	0.18 (0.02)	0.11 (0.01)	0.57 (0.02)	0.54 (0.02)	0.82 (0.03)	0.23 (0.03)	0.67 (0.0)	0.21 (0.01)	0.32 (0.05)	0.36 (0.03)	0.27 (0.04)	0.37 (0.06)
struc2vec	0.29 (0.01)	0.28 (0.01)	0.03 (0.0)	0.06 (0.01)	0.06 (0.0)	0.52 (0.03)	0.24 (0.01)	0.17 (0.01)	0.21 (0.04)	0.45 (0.0)	0.21 (0.02)	0.13 (0.01)	0.34 (0.01)	0.28 (0.07)	0.3 (0.03)

Chapter 4

Neuro-symbolic Representation Evolution from Text Data

> Follow that will and that way
> which experience confirms to be your own.
>
> Carl Gustav Jung

In this chapter, we discuss the design and development of a neuro-symbolic method aimed at automating humans' participation at competitions related to the domain of *text classification*. The key novelty introduced in this chapter concerns a developed approach to automatic *representation evolution* and subsequent learning. The developed approach is one of the first approaches offering autoML capabilities on an off-the-shelf laptop, while performing better than many existing baseline approaches that are commonly adopted in practice.

4.1 Learning from Texts

The central problem addressed with the developed approach concerns *representation learning* from texts. As this type of input cannot be directly considered by fast learners, such as regularized logistic regression, an initial step is required, which produces a collection of representations of a given document. This set of representations can be re-weighted and combined to address a particular task, which is many times manually conducted by a human developer. The goal of this work was to automate this part, using an evolution strategy-based approach which learns to emphasize particular parts of the feature space in an interactive manner.

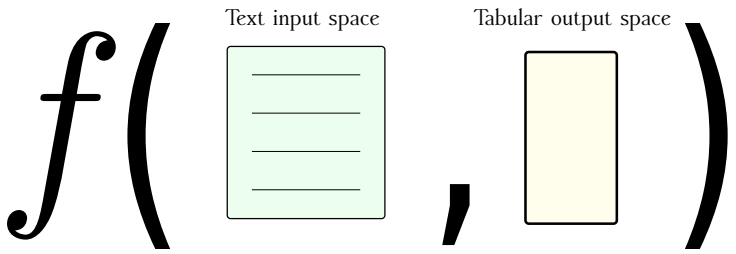


Figure 4.1: Schematic overview of text classification.

4.2 Automating Neuro-symbolic Representation Learning

The key contribution discussed next addresses the problem of automated neuro-symbolic representation evolution. The rationale for developing the presented autoBOT method revolves around the assumption that non-structured data sources, such as documents (texts), can be represented differently, entailing a given instance's properties at different semantic/lexical levels. As identifying the suitable level can be a cumbersome (manually solved) task, we aimed to *automate* this step entirely by performing an evolution-based search through the space of both representations and learners for a given classification task. The paper relevant to this section is the following one:

> Škrlj, B., Martinc, M., Lavrač, N., & Pollak, S. (2021). autoBOT: evolving neuro-symbolic representations for explainable low resource text classification. *Machine Learning*, 989–1028. https://doi.org/10.1007/s10994-021-05968-x

4.2.1 Key Contributions

We next present the key contributions of the conducted work.

1. We developed autoBOT, an automated text classifier. Its performance was demonstrated on multiple real data sets against strong baselines such as neural language models and other autoML approaches.

2. By exploring how sparse text representations can be integrated into the final joint representation efficiently, the developed method scales to hundreds of thousands of features on an off-the-shelf laptop.

3. We demonstrated that the representations relevant for a given task highly depend on the task type itself, indicating that an adaptive approach such as autoBOT captures task-specific intricacies potentially ignored by conventional approaches, which do not perform task-specific representation tuning.

4. The proposed autoBOT does not reach neural language model-level performance for all considered tasks, however remains *interpretable* both at the level of individual features and at the level of feature types (indicating what feature type is the most relevant). Furthermore, the final models include orders of magnitude fewer parameters than state-of-the-art, offering solutions that do not require specialized hardware for competitive performance.

5. The developed autoBOT is presented as a simple-to-use Python library, requiring only a few lines of code to test the default version, making it one of the simplest-to-use autoML systems for text.

4.2.2 Addressed Hypotheses and Discussion

We next discuss the two main hypotheses addressed with the work presented in this chapter. The first hypothesis concerns neuro-symbolic *model interpretability* (Hypothesis 1, Section 1.3), and the second *model scalability* (Hypothesis 4, Section 1.3). As demonstrated in the presented paper, maintaining high-performant and *interpretable* classifiers is a non-trivial task. Current experiments indicate that automatically obtained models can perform similarly to contemporary large neural language models; however, the obtained models remain interpretable as their coefficients correspond directly to separate features. Furthermore, feature type-level importances are obtained as a direct result of *evolution*.

By being able to pinpoint what are the key feature types/features, even though parts of the feature space are sub-symbolic, autoBOT demonstrates that the neuro-symbolic paradigm is a potentially promising solution to better understand the key patterns identified for solving a given task. It can happen, however, that the sub-symbolic features are the most important ones. In this case, autoBOT is as interpretable as any sub-symbolic model – it can be hard to pinpoint what impacts such models without *post-hoc* analysis.

One of the key contributions of this work is the autoBOT's capability to *adapt* to a given task automatically. This part of the evaluation partially also addresses the hypothesis concerning the better understanding of the two paradigms when jointly used for learning. We observed that for different tasks, different feature subspaces emerged as the most relevant. Furthermore, as the feature types remain the same across all considered text classification tasks, we could directly compare these solutions (i.e. type-related importances), discovering that sets of tasks share the same parts of the feature space in the final solution.

In terms of predictive performance (Hypothesis 3, Section 1.3) we observed the following. We hypothesized that the neuro-symbolic system would perform better than its symbolic counterparts, however, worse when compared to the contemporary language models such as RoBERTa and BERT. Based on current results, we can confirm the hypothesis as, indeed, autoBOT (neuro-symbolic) performed better than its symbolic-only counterparts. Further, the large language models performed consistently better in terms of classification performance. We attribute this performance difference to the differences in model sizes – the neuro-symbolic version of autoBOT includes at most tens of thousands of parameters ($\approx 10^4$), whereas the language models' parameter count exceeds 10^8. These multiple orders of magnitude larger models appear to capture more potentially relevant/more sophisticated, sub-word patterns, which can offer superior performance. Interestingly, however, we observed that variants of autoBOT with, e.g., 20,000 features performed better than word-only TF-IDF-based models with more features. This observation indicates that different feature types, when correctly emphasized, potentially offer task-specific, efficient solutions for a given text classification problem. Another observation related to classification performance is that when a larger number of classes are present, contextual language models do not necessarily offer competitive performance out-of-the-box. This observation indicates that additional fine tuning of these models could have been appropriate, however, it was out of the scope of this study.

Finally, one of the key features of autoBOT is its low-resource mode of operation. The tool was designed to not require any specialized hardware, require as little human input as possible, and perform better than *ad-hoc* baselines even on an off-the-shelf laptop. One of the key design patterns which led us to the current implementation is the adoption of sparse matrix algebra, both for representing the feature space and for training the classifiers. By being able to maintain only the sparse structure of the space (and weights) through the evolution, autoBOT requires relatively low memory. Furthermore, the final models are also low-resource: they commonly consist of thousands (or at most tens of thousands of parameters).

Machine Learning
https://doi.org/10.1007/s10994-021-05968-x

autoBOT: evolving neuro-symbolic representations for explainable low resource text classification

Blaž Škrlj[1,2] · **Matej Martinc**[1,2] · **Nada Lavrač**[1,3] · **Senja Pollak**[1]

Received: 20 April 2020 / Revised: 9 February 2021 / Accepted: 4 March 2021

Abstract

Learning from texts has been widely adopted throughout industry and science. While state-of-the-art neural language models have shown very promising results for text classification, they are expensive to (pre-)train, require large amounts of data and tuning of hundreds of millions or more parameters. This paper explores how automatically evolved text representations can serve as a basis for explainable, low-resource branch of models with competitive performance that are subject to automated hyperparameter tuning. We present autoBOT (automatic Bags-Of-Tokens), an autoML approach suitable for low resource learning scenarios, where both the hardware and the amount of data required for training are limited. The proposed approach consists of an evolutionary algorithm that jointly optimizes various sparse representations of a given text (including word, subword, POS tag, keyword-based, knowledge graph-based and relational features) and two types of document embeddings (non-sparse representations). The key idea of autoBOT is that, instead of evolving at the learner level, evolution is conducted at the representation level. The proposed method offers competitive classification performance on fourteen real-world classification tasks when compared against a competitive autoML approach that evolves ensemble models, as well as state-of-the-art neural language models such as BERT and RoBERTa. Moreover, the approach is explainable, as the importance of the parts of the input space is part of the final solution yielded by the proposed optimization procedure, offering potential for meta-transfer learning.

Keywords Representation learning · Natural language processing · AutoML · Neuro-symbolic computing

Editors: Nikos Katzouris, Alexander Artikis, Luc De Raedt, Artur d'Avila Garcez, Sebastijan Dumančić, Ute Schmid, Jay Pujara.

✉ Blaž Škrlj
blaz.skrlj@ijs.si

Extended author information available on the last page of the article

1 Introduction

Contemporary machine learning approaches successfully solve many natural language processing tasks, spanning from question answering, disambiguation, duplicate detection to classification. The emerging paradigm that successfully solves these tasks are transformer-based language models, i.e. deep neural networks that are first pre-trained on large corpora and only fine-tuned for a specific task (Devlin et al. 2019; Jing and Xu 2019).

Even though such (black-box) models offer state-of-the-art performance, the models are not directly explainable (Rudin 2019). Further, specialized hardware, such as Tensor Processing Units (TPUs) or GPGPUs (General Purpose Graphical Processing Units) are needed for their training and evaluation. Neural language models (such as the transformer architectures) inherently operate with dense vector spaces (embeddings), leveraging the multiparallelism of the modern hardware (Jouppi et al. 2017). This work focuses on the other part of the model spectrum: we investigated whether different sparse representations of text could be *evolved* in a low-resource manner, offering similar performance as dense representations, especially in settings where the available data is scarce. The main contributions of this work are summarized below.

- We propose autoBOT (automatic Bags-Of-Tokens), a system capable of efficient, simultaneous learning from multiple representations of a given document set.
- The system's hyperparameters are optimized by using an evolutionary algorithm, adopted for exploration of high-dimensional sparse vector spaces—evolution governs the representation used for learning by a collection of linear models trained with stochastic gradient descent.
- The dimension of the evolved space is estimated based on the expected sparsity of the representation.
- The performance of autoBOT can be competitive to pre-trained transformer models and other state-of-the-art learners, as demonstrated on fourteen text classification data sets, while using less computational resources and requiring zero manual hyperparameter tuning for achieving reasonable out-of-the-box performance (given enough time).
- autoBOT offers visualization of the similarity of parts of the feature space across multiple data sets. Such visualizations offer fast overview into key parts of the feature space relevant for a given data set.
- We explore three novel feature types, namely features derived from document keywords, relational features that represent pairs of tokens at a given distance and first-order features constructed based on a collection of 34,074,917 grounded relations from the ConceptNet (Speer et al. 2017) *knowledge graph*.
- The proposed system is especially suited for settings, where hardware as well as the amount of data are limited.

The remainder of this work is structured as follows. In Section 2 we discuss the related work that influenced the development of autoBOT. Section 3 presents the proposed autoBOT system for learning from evolvable text representations, including the issue of representing texts, the formulation of the autoBOT learning task, as well as the issue of its explainability. Section 4 presents the conducted experiments, and in Section 5 we discuss the obtained results. Section 6 presents the conclusions and plans for further work.

2 Related work

In this section we discuss the related approaches that inspired the development of the proposed autoBOT system. We begin by discussing the notion of text representation learning (Section 2.1), followed by text classification (Section 2.2) and evolutionary computation (Section 2.3). Finally, we discuss the state-of-the-art autoML systems in Section 2.4.

2.1 Text representation learning

Machine learning approaches that learn from text usually consist of two main steps: preprocessing the text into a suitable representation, e.g., the Bag-of-words (BoW) format, followed by subsequent learning. The main drawback of such approaches is the requirement of the user's specification of how the text should be represented, at what granularity etc. Such semi-automated feature construction can be time-demanding and requires large amounts of development time, however, the subsequent *learning* can be very efficient (Mirończuk and Protasiewicz 2018).

Recent developments in the field of representation learning offer many insights into the importance of having a suitable representation for the given problem. Transformer-based language models, such as BERT (Devlin et al. 2019), RoBERTa (Liu et al. 2019), XLNet (Yang et al. 2019), learn multi-faceted representations of the provided input sequences, where multiple computational layers are used to distill the obtained representation into a form used for more general problem solving. Similar insights also emerged in the fields of graph (Kipf and Welling 2017) and image (Szegedy et al. 2017) representation learning. The state-of-the-art transformer language models also use subword information due to byte-pair encoded inputs (Sennrich et al. 2016), offering even better performance, albeit at the cost of explainability.

Representations learnt by deep neural network models are dense; for example, vectors of dimension < 1000 are used to capture relations between input tokens. On the other hand, many shared tasks, especially the ones where the number of input instances is in the order of hundreds, yield themselves to more conventional, even linear models that operate on sparse input spaces (Martinc et al. 2017). The main caveat of such approaches is the inclusion of the human factor: humans need to *carefully* fine-tune many parameters without well defined properties or predictable behavior. For example, it is not clear how the word-based features should be weighted when compared to character-based ones, how the classifier should be regularized etc.

Further, the collections of *features* are also arbitrary as there is no general theoretical background as to when to apply what type of e.g., n-grams or other features (e.g., emoji counts etc.). Hence, such systems are commonly fine-tuned for a particular domain, yet need non-negligible human effort to perform adequately well for the same task in a different domain. For example, a system can perform well when classifying sentiment, however it fails at the prediction of side effects based on the patient reports. Finally, exhaustive search of the hyperparameter space is in most cases computationally intractable.

2.2 Text classification

We continue the discussion by considering different machine learning approaches employed for the task of text classification, how they relate to this paper and what are their potential limitations. Text classification explores how representations of a given collection

of documents can be *associated* with a given target space, such as for example a collection of genres. Broadly, text classification approaches can be split into two main groups, namely symbolic and sub-symbolic classifiers. The canonical example of symbolic learners are linear classifiers such as the logistic regression or linear Support Vector Machines, which learn to classify e.g., TF-IDF encoded documents (Manning et al. 2008; Kowsari et al. 2019; Agarwal and Mittal 2014). In recent years, however, the paradigm of neural language models has also offered state-of-the-art classifiers across multiple domains (Jing and Xu 2019). Some of the currently best-performing classifiers are commonly fine-tuned language models, pre-trained on large textual corpora (Belinkov and Glass 2019). Albeit extensive pre-training is currently inaccessible to majority of researchers, fine-tuning can be conducted with adequate off-the-shelf GPUs, and is actively employed on many e.g., shared tasks, ranging from classification of social media-related texts to classification of biomedical documents (Moradi et al. 2020). Compared to discussed approaches, which derive a representation from raw text, approaches that are able to exploit background knowledge alongside raw text are also of increasing interest and serve as one of the motivations for the proposed autoBOT. Background knowledge can be considered in many forms. Ontologies and taxonomies represent formally defined, hierarchical structures with human-defined concepts and relations between them. Some canonical examples of such knowledge sources are for example the WordNet (Fellbaum 2012) and similar taxonomies. On the other hand, *knowledge graphs* are the structures that can be defined semi-automatically, and are commonly comprised of millions of subject-predicate-object triplets. Examples of freely available knowledge graphs include the ConceptNet (Speer et al. 2017) used in this work.

2.3 Evolutionary computation and learning

We discuss in more detail the applications and the underpinnings of evolutionary computation, and more specifically *genetic algorithms*, as this metaheuristic optimization idea was also used to guide representation learning conducted by autoBOT. Genetic algorithms have been considered for both combinatorial and continuous optimization problems in the second part of the 20th century (Mitchell 1998). Inspired by (a very basic) notion of biological evolution, these optimization algorithms often gradually *evolve* a solution via the process of intermediary evaluation, crossover, mutation and selection.

More recently, genetic algorithms (GA) evidence widespread use throughout industrial and academic projects, where GAs were successfully applied to tackle otherwise analytically intractable problems (Chambers 2000). Even though genetic and other algorithms for hard optimization problems were applied to many real-life problems, their use for improving machine learning approaches has only recently become mainstream (see Stanley et al. (2019) for an exhaustive overview); neuroevolution was already considered in 1960s, however it was computationally infeasible at the time. Neuroevolution performs well for traditional benchmark tasks, such as the knapsack problems (Denysiuk et al. 2019), but also real-life robotics problems (Zimmer and Doncieux 2017). Evolution-based approaches were also successfully adopted for the task of scientific workflow discovery (Pilat et al. 2016), offering symbolic descriptions of data mining workflows, directly applicable in practice. Neuroevolution Stanley et al. (2019) approaches have shown promising results in the domain of computer vision, where more efficient neural networks were evolved with minimal performance trade-offs (Zoph et al. 2018).

One of the early approaches on how genetic algorithms can be adopted for the feature selection purposes was proposed in Vafaie and De Jong (1998). The authors developed a system that employs a genetic algorithm to select feature subspaces useful for a decision tree classifier. They successfully showcased the performance of their approach on an eye-detection problem. The proposed autoBOT builds on a similar idea, i.e. that feature subspaces can be evolved prior to learning, however, extends the idea to multiple different instance (documents instead of images) representations, from symbolic to non-symbolic. Further, autoBOT also explores novel representation types such as e.g., knowledge-graph based features, capable of exploiting the knowledge beyond the textual training data considered.

More recent works explore how task scheduling can be tackled by employing a combination of evolution and learning (Dorronsoro and Pinel 2017). Similarly convincing results were also recently demonstrated for the task of material discovery (Jennings et al. 2019), where machine learning algorithms were used to guide the evolution, offering up to 50x speedup compared to naïve exhaustive search.

2.4 Advancements in autoML systems

Automatic learning of machine learning pipelines has been thoroughly explored for tabular data in tools such as AutoWEKA (Thornton et al. 2013) and auto-sklearn (Feurer et al. 2019). The key idea is that parts of the learning procedure are *modularized* and automatically explored. For example, AutoWEKA and auto-sklearn employ Bayesian optimization (Snoek et al. 2012) for scalable and efficient exploration of such hyperparameter spaces. These approaches assume a tabular input, and consequently explore both the preprocessing, as well as heterogeneous ensemble construction methods that yield the best performing configuration. Another example of automated (tree-based) learning is conducted within TPOT (Olson et al. 2019), a tool for automatic construction of scikit-learn workflows specializing in tree-based learners. The main advantage of TPOT is *simplicity*—competitive results on tabular data sets can be obtained by merely running the default optimization setting for a dedicated amount of time. Development of approaches for automatic learning renders possible fast *prototyping*—instead of spending days in deciding to what extent the current data is suitable for learning—autoML systems offer quick and effortless answers to such questions, greatly speeding up the machine learning development and deployment process.

Another prominent example of the machine learning algorithm design are the automatically constructed deep neural architectures, for example, used for solving image recognition tasks (He et al. 2018). In this field of *neuroevolution* (Stanley et al. 2019) , genetic algorithms and their variations are commonly used, and were recently shown to perform better than many alternative optimization approaches. Even though evolved neural networks were shown to perform well for image data, and the majority of the remaining autoML systems focus on tabular data, we believe that research on how automatic machine learning can aid the development of algorithms that learn from texts is still scarce and worth exploring. The idea of autoML was adapted also to text domains (Madrid 2019). Similarly, Google also

offers proprietary cloud-based solutions that address also the domain of natural language[1]. Learning from texts automatically is an interesting research question, especially if the hardware is not specialized for learning, and the data are scarce.

Apart from the machine learning-based approaches, explored by the evolutionary computation community, the machine learning papers that exploit evolution (or similar optimization) were developed in parallel to the aforementioned studies. For example, the implications of using evolutionary computation for the meta learning purposes on tabular data was also explored (Reif et al. 2012). They explored the performance of SVMs and random forest-based classifiers on over 100 data sets from the UCI (Dua and Graff 2017). The authors have shown that a standard genetic algorithm already offers performance improvements. Note that the methods such as the auto-sklearn (Feurer et al. 2019), TPOT (Olson et al. 2019) and AutoWEKA (Kotthoff et al. 2017) also show consistent improvements of using stochastic optimization on tabular data. Further, autoML frameworks such as GAMA (Gijsbers and Vanschoren 2019), hyperopt-sklearn (Komer et al. 2014), ML-Plan (Mohr et al. 2018) and OBOE (Yang et al. 2019) all offer an optimization layer on top of an existing e.g., learning pipeline which requires hyperparameter tuning. The proposed autoBOT, albeit being conceptually similar to the work of (Dua and Graff 2017) at the optimization level, explores how the evolution can be conducted at the representation level, which is a rather novel endeavour. Further, evolution on unstructured data such as texts is also a novelty compared to e.g., optimization for tabular classifiers.

2.5 The rationale behind autoBOT

This work presents autoBOT, an approach for scalable, low-resource text classification that requires as little human input as possible, but nevertheless offers a decent classification performance. To our knowledge, similar approaches were explored mostly for tabular data, where the representation is already given, or for evolution of neural network architectures, where the models many times require custom hardware and are not (at all) explainable. We believe that evolution—when operating with less structured inputs such as texts—should simultaneously consider both the suitable representation and the subsequent learning, which was to our knowledge not yet explored at the scale done in this work. Further, the optimized feature space is inherently sparse, requiring an end-to-end implementation that operates with sparse matrix-algebraic operations (including learning), otherwise resulting in high dimensional dense vector spaces that require lots of computational resources. For example, considering a dense matrix of a hundred thousand features is computationally infeasible, unless sparse representation is considered.

3 Learning from evolving text representations with autoBOT

In this section, we present the proposed autoBOT approach. First, we discuss the representations of text considered, followed by the overall formulation of the approach. A schematic overview of autoBOT is shown in Figure 1.

[1] https://cloud.google.com/natural-language/automl/docs/beginners-guide, however this software is not open-source.

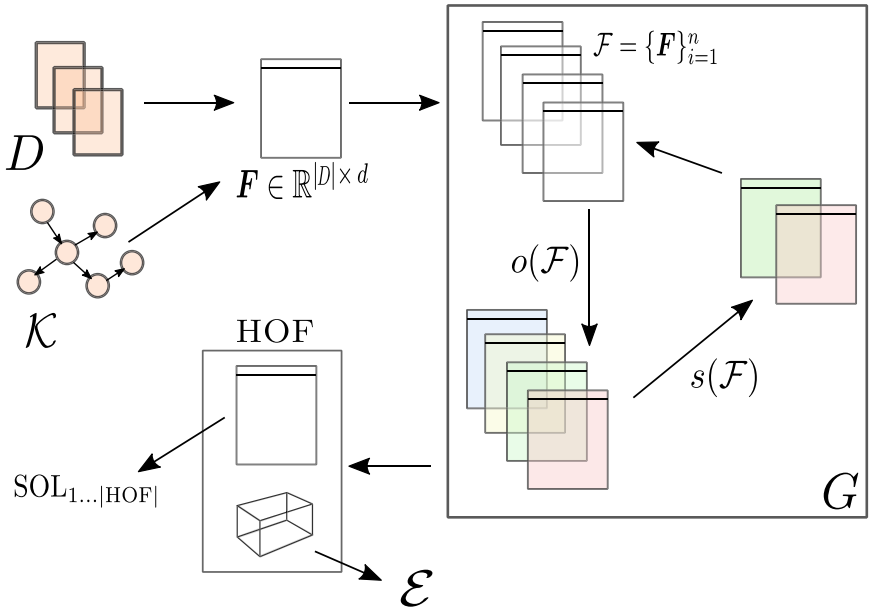


Fig. 1 Schematic overview of autoBOT. The input is a collection of documents D alongside a knowledge graph $\mathcal{K}$. The feature space $\boldsymbol{F}$ is constructed based on the information from both sources. Next, G generations of representation evolution are conducted. Here, the $o(\mathcal{F})$ represents the application of different operators to solution vectors representing weights of feature subspaces (e.g., word, character etc.), followed by selection, $s(\mathcal{F})$, where the next generation of solutions is chosen. Once the optimization finishes, the best solutions (HOF - Hall Of Fame) are used for the final set of predictions. The $\mathrm{SOL}_1 \cdots {}_{|\mathrm{HOF}|}$ denotes the individual solutions, used for construction of final classifiers, and ε represents the set of explanations – feature-value associations. As the solutions encode both the weights at the feature subspace level, as well as weights of individual features, autoBOT offers two distinct views of feature importances

Here, the training set of documents is first represented at different granularities ($\boldsymbol{F}$); Sparse bag-of-words type of vectors on the level of characters, words, part-of-speech (POS) tags as well as keywords and relations spanning multiple tokens, to dense document embeddings and knowledge graph-based features ($\mathcal{K}$). This is followed by the process of representation evolution (G field). The obtained initial set of representations is considered as the base for evolutionary optimization. Here, weights (individuals), multiplied with the feature values corresponding to the parts of this space are evolved so that a given performance score is maximized. The final set of solutions is used to obtain a set of individual classifiers, each trained on a different part of the space. However, for obtaining final predictions, a majority vote scheme is considered. Hence, evolution effectively emits an ensemble of classifiers. More details follow below.

3.1 Multi-level representation of text

Let FT represent the set of all *feature types* that are considered during evolution. Let D denote the set of considered document instances. Examples of feature types include single word features, their n-grams, character n-grams etc. Assuming f represents a given feature type. Let d_f denote the number of features of this type. The number of all features is defined as $d = \sum_f d_f$. Hence, the final d-dimensional document space consists of concatenated $\boldsymbol{F}_f \in \mathbb{R}^{|D| \times d_f}$-dimensional matrices, i.e.

$$\boldsymbol{F} = \Big|\Big|_i \boldsymbol{F}_i,$$

Table 1 Different feature types considered by autoBOT

Feature generator type	Description	Data type	Feature type	Sparse
Word n-grams	words	raw text	symbolic	yes
Character n-grams	tuples of sequential characters	raw text	symbolic	yes
Keyword features	one or multi-term keyphrases	graph-based token paths	symbolic	yes
Relational features	globally close characters	distance relation	symbolic	yes
POS n-grams	part-of-speech tags	grammatical	symbolic	yes
Knowledge graph features	grounded relations	semantic	symbolic	yes
Document embeddings	document embeddings (distributed memory - DM)	embedding	sub-symbolic	no
Document embeddings	document embeddings (distributed bag of words - DBOW)	embedding	sub-symbolic	no

where i denotes the i-th feature type, and $||$ denotes concatenation along the separate columns. The matrix is next normalized (L2, row-wise), as is common practice in text mining. Types of features considered by autoBOT are summarized in Table 1.

The considered features, apart from the relational ones and document embeddings, are subject to TF-IDF weighting, i.e.,

$$\text{TF-IDF}(t, m) = \sum_{j \in m} \mathbb{1}[j = t] \cdot \log\left(\frac{|D|}{\sum_{k \in D} \mathbb{1}[t \in k] + 1}\right), \tag{1}$$

where t is a token of interest and m the document of interest. The D is the set of all documents. While word and character n-grams, POS tags as well as document embeddings[2] are commonly used, the relational, knowledge graph-based and keyword-based features are a novelty of autoBOT discussed below.

Relational features. One of the key novelties introduced in this paper is the relational feature construction method, summarized as follows. Consider two tokens, t_1 and t_2. autoBOT already considers n-grams of length 2, which would account for patterns of the form (t_1,t_2). However, longer-range relations between tokens are not captured this way. As part of autoBOT, we implemented an efficient *relation extractor*, capable of producing symbolic features described by the following (i-th) first-order rule: $\mathcal{R}_i := \text{presentAtDistance}(t_1, t_2, \overline{\delta}(t_1, t_2))$, where $\overline{\delta}$ represents the average distance between a given token pair across the *training documents*. Thus, the features represent pairs of tokens, characterized by binary feature values, derived from the top $d_{t=\text{relational}}$ distances (number of considered features) between token pairs. An example is given next.

[2] See Le and Mikolov (2014) for an overview of the two embedding models used. The two namings, i.e., DBOW and DM are used in the state-of-the-art implementation in Khosrovian et al. (2008).

Example. Consider the sentence "We like peanut butter". Assuming the "like" and the "butter" tokens are, when co-occurring, on average two tokens away, i.e. $\overline{\delta} = 2$. The new feature that is considered is thus a tuple ("like","butter") with corresponding value 2 , assuming the distance of 2 falls under the top $d_{t=\text{relational}}$ considered features if sorted in ascending order. The feature values are thus binary, corresponding directly to a given distance predicate. Current implementation, however, uses characters instead of words, as they tend to offer better results. For example, ("t","r") tuple is a possible feature if considering the sentence at the start of this paragraph.

Keyword-based features.

The second type of features introduced in this work are the features based on *keywords*. Given a document, *keywords* represent a subset of tokens that are representative of the document. There exist many approaches for keyword detection. For example, statistical methods, such as KP-MINER (El-Beltagy and Rafea 2009), RAKE (Rose et al. 2010) and YAKE (Campos et al. 2018), use statistical characteristics of texts to capture keywords. On the other hand, graph-based methods, such as TextRank (Mihalcea et al. 2004), Single Rank (Wan and Xiao 2008), TopicRank (Bougouin et al. 2013), Topical PageRank (Sterckx et al. 2015) and RaKUn (Škrlj et al. 2019) build graphs to rank words based on their position in the graph. The latter is also the method adopted as a part of autoBOT for the feature construction process, which proceeds in the following steps:

1. **Keyword detection**. First, for each class, the set of documents from the training corpus corresponding to this class are gathered. Next, keywords are detected by using the RaKUn algorithm for each set of documents separately. In this way, a set of keywords is obtained for each target class.
2. **Vectorization**. The set of unique keywords is next obtained, and serves as the basis for novel features that are obtained as follows. For each document in the training corpus, only the keywords from the subset of all keywords corresponding to the class with which the document is annotated are recorded (in the order of appearance in the original document), and used as a token representation of a given document. This way, the keywords specific for a given class are used to construct novel, simpler "documents". Finally, a TF-IDF scheme is adopted as for e.g., character or word n-grams, yielding n most frequent keywords as the final features [3].

The rationale behind incorporating keyword-based features is that more local information, specific to documents of a particular class is considered, potentially uncovering more subtle token sets that are relevant for the differentiation between the classes.

Knowledge graph-based features. A key novelty introduced as part of autoBOT is the incorporation of *knowledge-graph-based features*. Knowledge graphs are large, mostly automatically constructed relational sources of knowledge. In this work we explored how

[3] The features, identified on the training set of data as relevant are also used to construct the test set's instances.

Table 2 Considered relations. from ConcepNet considered by PropFOL

/r/Antonym	/r/AtLocation	/r/CapableOf
/r/Causes	/r/CausesDesire	/r/CreatedBy
/r/dbpedia/capital	/r/dbpedia/field	/r/dbpedia/genre
/r/dbpedia/genus	/r/dbpedia/influencedBy	/r/dbpedia/knownFor
/r/dbpedia/language	/r/dbpedia/leader	/r/dbpedia/occupation
/r/dbpedia/product	/r/Desires	/r/DistinctFrom
/r/Entails	/r/EtymologicallyDerivedFrom	/r/EtymologicallyRelatedTo
/r/ExternalURL	/r/FormOf	/r/HasA
/r/HasContext	/r/HasFirstSubevent	/r/HasLastSubevent
/r/HasPrerequisite	/r/HasProperty	/r/HasSubevent
/r/InstanceOf	/r/IsA	/r/LocatedNear
/r/MadeOf	/r/MannerOf	/r/NotDesires
/r/NotHasProperty	/r/NotUsedFor	/r/ObstructedBy
/r/PartOf	/r/ReceivesAction	/r/RelatedTo
/r/SimilarTo	/r/SymbolOf	/r/Synonym
/r/UsedFor	/r/MotivatedByGoal	/r/NotCapableOf
/r/DefinedAs	/r/DerivedFrom	

ConceptNet (Speer et al. 2017), one of the currently largest freely available multilingual knowledge graphs could be used to construct novel features of which scope extends the considered data set[4]. We propose an algorithm for *propositionalization* of *grounded relations*, discussed next.

Assuming a collection of documents D, the proposed propositionalization procedure identifies which relations, present in the knowledge graph, are also present in a given $k \in D$. Let $\mathcal{K} = (N, E)$ represent the knowledge graph used, where N is the set of terms and E the set of subject-predicate-object triplets, so that the subject and the object are two terms. We are interested in finding a collection of features F_{KG} (i.e. knowledge graph-based features). We build on the late propositionalization ideas of Lavrač et al. (2020), where zero-order logical structures are effectively used as features, that are *automatically* identified. We refer to the algorithm capable of such scalable extraction of first-order features as PropFOL, summarised next. The key idea of PropFOL is related to grounding the triplets, appearing in a given knowledge graph while traversing the document space. More specifically, each document k is traversed, and the relations present in each document are stored. The relations considered by PropFOL are shown in Table 2. The PropFOL operates by memorizing the collections of grounded relations in each k (document). Once the document corpus is traversed, the *bags* of grounded relations are *vectorized* in TF-IDF manner. Finally, for each new document, two operations need to be conducted. First, the grounded relations need to be identified. Second, the collection of relations is vectorized by using the stored weights of the individual relations occurring based on the training data. The feature construction algorithm is given as the Algorithm 1.

[4] September 2020 version, found at https://github.com/commonsense/conceptnet5

Algorithm 1: PropFOL

Data: Set of labeled training documents D, a knowledge graph $\mathcal{K}$, number of features μ

1 relationBags $\leftarrow$ [] ‡ Empty container for bags.
2 tokenSpace $\leftarrow$ getAllTokens(D) ‡ Space of all tokens.
3 groundedTripletGraph $\leftarrow$ {}
4 **for** $(head, relation, tail) \in \mathcal{K}$ **do**
5 **if** $head \in tokenspace \wedge tail \in tokenspace$ **then**
6 groundedTripletGraph.add(((head, relation, tail))) ‡ Grounded subgraph.
7 **for** $k \in D$ **do**
8 documentTokens $\leftarrow$ getTokens(k) ‡ Extract tokens.
9 tokenSubgraph $\leftarrow$ getSubgraph(groundedTripletGraph, documentTokens)
10 rBAG $\leftarrow$ []
11 **for** $e \in E(tokenSubgraph)$ **do**
12 decodedTriplet $\leftarrow$ decodeToTriplet(e) ‡ Get the whole triplet.
13 rBAG.append(decodedTriplet) ‡ Construct bags.
14 relationBags.append(rBAG)
15 vectorizer $\leftarrow$ frequencyWeighting(relationBags) ‡ See E.q. 1.
16 $\boldsymbol{F}_{\text{KG}} \leftarrow$ vectorizer.transform(relationBags,μ) ‡ Final representation.
17 **return** $\boldsymbol{F}_{\text{KG}}$

The algorithm consists of two main steps. First, the document corpus (D) is traversed (line 4), whilst the relations are being recorded for each document (k). Once memorized (for training data, line 7), a vectorizer is constructed, which in this work conducts TF-IDF re-weighting (line 16) of first order features, and based on their overall frequency selects the top n such features that shall be used during evolution. Note that this simple *propositionalization* scheme is adopted due to a large knowledge graph considered in this work, as one of the key purposes of autoBOT is to maintain scalability (such graph can be processed on an off-the-shelf laptop). Note that in practice, even though millions of entities and tens of millions of possible relations are inspected, the final collection of grounded relations, particular to a considered data set, remains relatively small. In more detail, the *getAllTokens* (line 2) method maps a given document corpus D to a finite set of possible tokens (e.g. words). The obtained token base is retrieved for each document (k, line 7) via *getTokens* method. The subset of tokens corresponding to a given document is next used to extract *a subgraph* of the input knowledge graph $\mathcal{K}$, corresponding to a given document. This step is mandatory as the subgraph effectively corresponds to the set of triplets that are used as features. The missing component at this point are the relations, which are retrieved via the *decodeToTriplet* method (line 12). Such triplets represent potentially interesting, background knowledge ($\mathcal{K}$)-based features. In the final part of the algorithm, triplet sets are processed as standard bags-of-items to obtain the real valued feature space suitable for learning ($\boldsymbol{F}_{\text{KG}}$).

The following example demonstrates how the constructed features are obtained, and what are the potentially interesting relations entailed by performing such feature construction.

Example. Consider the sentence "Both dogs and cats are animals, but so are bobcats". The proposed PropFOL would for example extract first order features IsA(dog, animal), IsA(bobcat, animal) and IsA(cat, animal). However, relations beyond IsA, such as SimilarTo(cat, bobcat) could also be considered.

This type of feature construction is thus able to extract relations, otherwise inaccessible by conventional learners that operate solely based on e.g., word-based representations. Even though current implementation of autoBOT exploits the ConceptNet knowledge graph due to its generality, the implementation permits utilization of *any* triplet knowledge base that can be mapped to parts of texts, and as such offers many potentially interesting domain-specific applications.

3.2 Solution specification and weight updates

The key part of every genetic algorithm is the notion of *solution* (an individual). The solution is commonly represented as a (real-valued) vector, with each element corresponding to the part of the overall solution. Let FT represent the set of feature types. The solution vector employed by the autoBOT is denoted with $\text{SOL} \in [0, 1]^{|\text{FT}|}$ ($|FT|$ is the number of feature types).

Note that the number of parameters a given solution consists of is exactly equal to the number of unique feature types (as seen in Table 1). The solution is denoted as:

$$\text{SOL} = \big[\underbrace{w_1, w_2, \ldots, w_{|\text{FT}|}}_{\text{Subspace weights}}\big].$$

Thus, the solution vector of the current implementation of autoBOT consists of 8 (hyper) parameters (for eight different feature types as seen in Table 1). Next, *solution evaluation*, the process of obtaining a numeric score from a given solution vector is discussed.

Each solution vector SOL consists of a set of weights, applicable to particular parts of the feature space. Note that the initial feature space, as discussed in Section 3.1, consists of d features. Given the weight-part of SOL, i.e. $[w_1, w_2, \ldots, w_{|\text{FT}|}]$, we define with I_i^{from} and I_i^{to} the two column indices, which define the set of columns of the i-th feature type. The original feature space $\boldsymbol{F}$ is updated as follows:

$$\boldsymbol{F}_s^{I_i^{\text{from}} \text{ to } I_i^{\text{to}}} = w_i^s \odot \boldsymbol{F}^{I_i^{\text{from}} \text{ to } I_i^{\text{to}}}. \tag{2}$$

where $\odot$ refers to matrix-scalar product and s to a particular individual (updated feature space). Note also that the superscript in the weight vector corresponds to the considered individual. The union of the obtained subspaces represents the final *representation* used for learning.

The key idea of autoBOT is that instead of evolving on the learner level, evolution is conducted at the *representation* level. The potential drawback of such setting is that if only a single learner was used to evaluate the quality of a given solution (representation),

the fitness score (that in this work equals to the mean score obtained during a five-fold cross validation on the training set) would be skewed. To overcome this issue, autoBOT—instead of a single classifier—considers a wide spectrum of linear models parameterized with different levels of elastic net regularization (trade-off between L1 and L2 norms) and losses (hinge and log loss are considered). Being trained by the stochastic gradient descent, hundreds of models can be evaluated in a matter of minutes, offering a more robust estimate of a given representation's quality. Note that each solution is considered by hundreds of learners, and there are multiple solutions in the overall population. More formally, we denote with

$$\mathcal{S}_c(\boldsymbol{F}) = \arg \max_h \left[\text{SGD}(\text{SOL}, h, \boldsymbol{F})\right] \tag{3}$$

the optimization process yielding the best performing classifier when considering feature space $\boldsymbol{F}$, where SGD represents a single, stochastic gradient descent-trained learner parameterized via h (a set of hyperparameters such as the loss function and regularization). Note that SGD considers the labeled feature space during learning.

A detailed specification of the family of linear models that are considered during fitness computation are given in Section 4.2. We next discuss the final component of autoBOT that can notably impact the evolution—the initialization. Let F_f represent a feature subspace (see Section 3.1 for details). The initial solution vector is specified as:

$$\text{SOL}_{\text{init}} = [\mathcal{S}_c(\boldsymbol{F}_f) \cdot \mathcal{U}(0.95, 1.05)]_{f \in FT}. \tag{4}$$

Note the link to Equation 3: the vector consists of feature type-specific performances. The $\mathcal{U}(a, b)$ represents a random number between a and b drawn from the uniform distribution. This serves as noise which we add to prevent initialization of too similar individuals. As in this work the F1 score is adopted for classifier performance evaluation, its range is known (0 to 1), thus the proposed initialization offers stable initial weight setting[5].

3.3 Dimension estimation

Commonly, dimension of a learned representation is considered as a hyperparameter. However, many recent works in the area of representation learning indicate that high-enough dimension is a robust solution across multiple domains, albeit at the cost of additional computational complexity. The proposed autoBOT exploits two main insights and adapts them for learning from *sparse* data. The dimension estimation is parametrized via the following relation:

$$d_f = \text{round}(d_d / s),$$

where d_f is the final dimension, d_d the *dense* dimension and s the estimated sparsity. The idea is that autoBOT attempts to estimate the size of the sparse vector space based on the assumption that models that operate with dense matrices require d_d dimensions for successful performance, and that s is the expected sparsity of the space produced by autoBOT. In this work, we consider $d_d = 128$ and $s = 0.1$, the dense dimension is based on the existing literature and s is low enough to yield a sparse space.

[5] However, should a different custom score be used, it is not necessarily a sensible approach.

3.4 Formulation of autoBOT

Having defined the key steps for evaluation of a single solution vector SOL, we continue by discussing how such evaluation represents a part of the *evolution* process undertaken by autoBOT. The reader can observe that the genetic algorithm adopted as part of autoBOT is one of the simplest ones, introduced already in the 1990s (Davis 1991).

Algorithm 2: autoBOT

Data: Set of labeled documents D, set of labels T, mutation rate η, crossover rate μ, maximum number of (dense) features per type d_t, sparsity s,tournament size ϕ

```
1  splits ← generateSplits(D,T)                          ‡ Splits of the data.
2  individuals ← generateInitial()                       ‡ Initialization.
3  HOF ← {}                                              ‡ Hall-of-fame initialization.
4  F ← initializeRepresentation(D,d_t,s)                 ‡ Initial sparse space.
5  while not converged or time limit not reached do
6      offspring ← copy(individuals)
7      for child pair in offspring do
8          if crossoverProb ≤ μ then
9              mate(child pair)                          ‡ Crossover phase.
10     for individual in offspring do
11         if mutationProb ≤ η then
12             individual ← mutate(individual)           ‡ Mutation phase.
13     fitnesses ← evaluateFitness(offspring,F,T)        ‡ See Equation 2
14     HOF ← updateHOF(offspring, fitnesses)             ‡ Hall-of-fame update.
15     individuals ← selectTournament(individuals ∪ offspring, φ)
16 ensembleLearner ← trainFinalLearners(HOF,D,T)         ‡ Final set of learners.
17 return ensembleLearner
```

The key steps of autoBOT, summarized in Algorithm 2, are outlined below. They involve initialization (line 2), followed by offspring creation (line 6). The two steps first initialize a population of a fixed size, followed by the main while loop, where each iteration generates a novel set of individuals (solutions), and finally (line 14) evaluates them against their parents in a tournament scheme. Note that prior to being evaluated, each population undergoes the processes of crossover and mutation (lines 7 and 10), where individuals are changed either pointwise (mutation), or piecewise (crossover). Once the evolution finishes, the HOF object (hall-of-fame) is inspected, and used to construct an *ensemble* learner that performs classifications via a voting scheme. In this work, we explore only time-bound evolution. Here, after a certain time period, the evolution is stopped. The more detailed description of the methods in Algorithm 2 is as follows. The *generateSplits* method offers the functionality to generate data splits used throughout the evolution. This step ensures that consequent steps of evolutions operate on the same feature spaces and are as such comparable. The *generateInitial* method generates a collection of real-valued vectors that serve as *the initial* population as discussed in Equation 4. Next, the *initializeRepresentation* method constructs the initial feature space, considered during evolution. Note that by initializing this space prior to evolution, the space needs to be constructed only once

compared to the naïve implementation where it is constructed for each individual. The *mate* and *mutate* methods correspond to standard crossover and mutation operators. The *evaluateFitness* method returns real valued performance assessment score of a given *representation*.[6] The *updateHOF* method serves as a storage of the best-performing individuals throughout all generations, and is effectively a priority queue with a fixed size. The *selectTournament* method is responsible for comparisons of individuals and the selection of the best-performing individuals that constitute the next generation of representations. Finally, the *trainFinalLearners* method considers the best-performing representations from the hall-of-fame, and trains the final classifier via extensive grid search.

We next discuss the family of linear models considered during evolution. Note that the following optimization is conducted both during evolution (line 13) and final model training (line 16). The error term considered by stochastic gradient descent is:

$$\text{Err}(\boldsymbol{w}, b) = \underbrace{\frac{1}{|D|}\sum_{i=1}^{|D|}\mathcal{L}(\boldsymbol{y}_i, \boldsymbol{w}^T\boldsymbol{x}_i + b))}_{\text{Loss term}} + \alpha\left[\underbrace{\frac{1-\beta}{2}\sum_{i=1}^{|D|}\boldsymbol{w}_i^2}_{\text{L2}} + \underbrace{\beta\sum_{i=1}^{|D|}|\boldsymbol{w}_i|}_{\text{L1}}\right],$$

where y is the target vector, x_i the i-th instance, $\boldsymbol{w}$ is a weight vector, $\mathcal{L}$ is the considered loss function, and α and β are two numeric hyperparameters: α represents the overall weight of the regularization term, and β the ratio between L1 and L2. The loss functions considered are the hinge and the log loss, discussed in detail for the interested reader in Friedman et al. (2001).

3.5 Theoretical considerations and explainability

We next discuss relevant theoretical aspects of autoBOT, with the focus on computational complexity and parallelism aspects, as the no-free-lunch nature of generic evolution as employed in this work has been previously studied in other works (Wolpert and Macready 1997; English 1996). In terms of computational complexity, the following aspects impact the evolution the most:

Feature construction. Let τ represent the number of unique tokens in the set of documents D. Currently, the most computationally expensive part is the computation of keywords, where the load centrality is computed (Škrlj et al. 2019). The worst case complexity of this step is $\mathcal{O}(\tau^3)$ – the number of nodes times the number of edges in the token graph, which is in the worst case τ^2. Note, however, that such scenario is unrealistic, as real-life corpora do not entail all possible token-token sequences (Zipf's law). The complexities of e.g., word, character, relational and embedding-based features are lower. Additionally, the features based on the knowledge graph information also contribute to the overall complexity, discussed next. Let $E(\mathcal{K})$ denote the set of all subject-predicate-object triplets considered. The propFOL (Algorithm 1) needs to traverse the space of triplets only once ($\mathcal{O}(|E(\mathcal{K})|)$). Finally, both of the mentioned steps take additional $|D|$ steps to read the corpus. We assume the remaining feature construction methods are less expensive.

Fitness function evaluation. As discussed in Section 3.2, evaluation of a single individual that encodes a particular representation is not conducted by training a single

[6] Note that each representation is evaluated by training a collection of linear classifiers in a cross-validation setting.

learner, but a family of linear classifiers. Let the number of models be denoted by ω, the number of individuals by ρ, and the number of generations by $|G|$ (G is a set of aggregated evaluations for each generation). The complexity of conducting evolution, guided by learning, is $\mathcal{O}(\rho \cdot \omega \cdot |G|)$.

Initial dimensionality estimation. The initial dimensionality is computed via a linear equation, and is $\mathcal{O}(1)$ w.r.t. the $|FT|$ (number of feature types).

Space complexity. When considering space complexity, we recognize the following aspects as relevant. Let $|I|$ denote the number of instances and $|FT|$ the number of distinct feature types. As discussed in Section 3.1 the number of all features is denoted with d_a, the space required by the evolution is $\mathcal{O}(|I| \cdot d_a \cdot \rho)$. In practice however, the feature space is mainly *sparse*, resulting in no significant spatial bottlenecks when tens of thousands of features are considered.

The individual computational steps considered above can be summarized as the following complexity:

$$\mathcal{O}(\underbrace{|D| + \tau^3 + |E(\mathcal{K})|}_{\text{Representation construction}} + \underbrace{\rho \cdot \omega \cdot |G|}_{\text{Evolution}}).$$

We next discuss how autoBOT computes solutions in parallel, offering significant speedups when multiple cores are used. There are two main options for adopting parallelism when considering simultaneously both the evolution and learning. The parallelism can be adopted either at the level of *individuals*, where each CPU core is occupied with a single individual, or at the learner level, where the grid search used to explore the space of linear classifiers is conducted in parallel. In autoBOT, we employ the second option, which we argument as follows. Adopting parallelism at the individual level implies that each worker considers a *different* representation, thus rendering sharing of the feature space amongst the learners problematic. However, this is not necessarily an issue when considering parallelism at the level of learners. Here, individuals are evaluated *sequentially*, however, the space of the learners is explored in parallel for a given solution (representation). This setting, ensuring *more memory efficient evolution*, is implemented in autoBOT. Formally, the space complexity, if performing parallelism at the individual's level rises to $\mathcal{O}(c \cdot |I| \cdot d_\alpha \cdot \rho)$, which albeit differing (linearly) only by the parameter c (the number of concurrent processes), could result in an order of magnitude higher memory footprint (when considering autoBOT on a e.g., 32 core machine). The option with sequential processing of the individuals but parallel evaluation of learners remains of favourable complexity $\mathcal{O}(|I| \cdot d_\alpha \cdot \rho)$ (assuming shared memory). An important aspect of autoBOT is also explainability, which is discussed next.

As individual features constructed by autoBOT already represent interpretable patterns (e.g., word n-grams), the normalized coefficients of the top performing classifiers obtained as a part of the final solution can be inspected directly. However, in practice, this can result in manual curation of tens of thousands of features, which is not necessarily feasible, and can be time consuming. To remedy this shortcoming, autoBOT's evolved weights, corresponding to semantically different *parts of the feature space* can be inspected *directly*. At this granularity, only up to e.g., eight different importances need to be considered, one per feature type, giving practical insights into whether the method, for example, benefits the most by considering word-level features, or it performs better when knowledge graph-based features are considered. In practice, we believe that combining both granularities can offer interesting insights into the model's inner workings, as considering only a handful of most important low-level (e.g., n-gram)

features can also be highly informative and indicative of the patterns recognized by the model as relevant.

Finally, autoBOT also offers direct insights into high-level overview of what *types of features* were the most relevant. We believe such information can serve for transfer learning purposes on the task level, which we explore as part of the qualitative evaluation.

3.6 How successful was evolution?

Quantification of a given evolution trace, i.e. fitness values w.r.t generations has been previously considered in Beyer et al. (2002), and even earlier in Rappl (1989), where the expected value of the fitness was considered alongside the optimum in order to assess how *efficient* is the evolution, given a fixed amount of resources. To our knowledge, however, the scores were not adapted specifically for a machine learning setting, which we address in the heuristic discussed next. We remind the reader that $G = (\text{perf}(i))_i$ represents a tuple denoting the evolution trace – the sequence of performances. Each element of G is in this work a real valued number between 0 and 1. Note that the tuple is ordered, meaning that when moving from left to right, the values correspond to the initial vs. late stages of the evolution's performance. Further, the perf(i) corresponds to the maximum performance in each generation. Let $\max_g(G)$ denote the maximum performance observed in a given evolution trace G. Let $\arg\max_g(G)$ represent the generation (i.e. evolution step) at which the maximum occurs. Finally, let $|G|$ denote the total number of evolution steps. Intuitively, both the maximum performance, as well as the time required to reach such performance (in generations) need to be taken into account. We propose the following score:

$$\text{GPERF}(G) = \underbrace{\max_g(G)}_{\text{Top score}} \cdot \underbrace{\left(1 - \frac{\arg\max_g(G)}{|G|}\right)}_{\text{How late it converged to the top score?}} .$$

Intuitively, the score should be high if the overall performance is good and evolution found the best performing solution quickly. On the other hand, if all the available time was spent, no matter how good the solution, the GPERF will be low. Note that the purpose of GPERF is to give insights into the evolution's efficiency, which should also take into account the time to reach a certain optimum. If the reader is interested solely in performance, such comparisons are also offered. Note that $\max_g(G)$ represents the best performing solution obtained during evolution. The heuristic, once computed for evolution runs across different data sets, offers also a potential insight into how suitable are particular classification problems for an evolution-based approach – this information is potentially correlated with the problem hardness.

4 Experiments

In this section we present the considered data sets, the adopted baselines with corresponding hyperparameter settings and the hardware environment used to conduct the experiments. The data sets are discussed in Section 4.1, followed by the discussion of the baselines in Section 4.2. Finally, the used hardware and software are presented in Section 4.3, followed by the evaluation in Section 4.4.

4.1 Data sets

This section presents the data sets used for quantitative evaluation of the autoBOT's performance. The data sets are summarized in Table 3. The selection of data sets spans from sentiment classification (*semeval* data sets), to news classification (*fox*, *bbc*), as well as personality classification (*mbti*). The data sets span various numbers of documents, from a few hundred to tens of thousands. The number of unique tokens represents the number of tokens obtained by doing document splitting directly by whitespace. Furthermore, multi-class and binary classification are considered.

4.2 Classifiers tested and hyperparameter settings

We next discuss the baseline approaches and configurations of autoBOT tested in this work. We divide baselines into the following main groups.

Manually tuned linear models. The first branch of models are linear classifiers, i.e. support vector machines (SVM) (Chang and Lin 2011) and logistic regression (LR), fine tuned across manually specified regularization ranges. The regularization of SVM and LR classifiers was in the range [0.1, 0.5, 1, 5, 10, 20, 50, 100, 500]. Each of the two learners was tested on word, character and word + character n-gram space. The feature space was normalized prior to learning.

Another autoML system. We considered TPOT, a state-of-the-art learner that adopts evolution on the level of learners (it evolves tree ensembles). We used the default settings on the word n-gram space, as this approach is not suitable for large sparse spaces.

Neural language models. Strong baselines, which operate with two orders of magnitude more parameters were also considered. More specifically, we fine-tuned BERT (base) and RoBERTa (base), two state-of-the-art language models for up to 20 epochs with early stopping, should the optimization converge faster. The hyperparameters for the two language models were left to defaults[7].

Representation-specific baselines. One of the key experiments needed to be conducted in order to assess the performance of the evolution was that of establishing baselines that learn directly from the constructed representation, however are not subject to iterative re-weighting of the feature space. To address this problem, we implemented a cartesian product of representation-learner baselines, that offer a solid estimation of how far can e.g., a SVM get by using only the initial autoBOT representation (but no evolution). The implemented classifiers are (as named in figures): autoBOT-svm-neural (only embeddings + SVM), autoBOT-svm-neurosymbolic (full feature space + SVM), autoBOT-svm-symbolic (symbolic features + SVM), and autoBOT-lr-neural (only embeddings + LR), autoBOT-lr-symbolic (symbolic features + LR) and autoBOT-lr-neurosymbolic (full feature space + LR).

Other baselines. We implemented a stratified majority classifier[8].

Having discussed the baseline approaches, we next discuss the considered variants of autoBOT. The main hyperparameters of evolution that we explored were the mutation rate and crossover rate. The mutation rates were varied in the range [0.3, 0.6, 0.9] and the crossover rates in the range [0.3, 0.4, 0.6, 0.9]. The tournament size was set to

[7] https://github.com/ThilinaRajapakse/simpletransformers

[8] https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html, default option

Table 3 Summary of the considered data sets

Data set	Documents	Unique tokens	Unique labels	Task	Source
kenyan	462	46189	2	News source prediction	Pollak et al. (2011)
semeval-2017-sentiment	1156	5144	4	Sentiment prediction	Nakov et al. (2013)[a]
bbc	2225	73491	4	News category prediction	Greene and Cunningham (2006)
subjects	1786	132996	4	Topic prediction	[b]
fox-news	2107	220063	7	News topic prediction	Qian and Zhai (2014)
insults	3946	36021	2	insult prediction	[c]
questions	5452	13279	6	Question types	Li and Roth (2002)
mbti	8675	572269	16	Personality type prediction	Myers (1962)[d]
yelp	10000	125446	5	Review prediction	[e]
hatespeech	10868	30555	4	Hate speech prediction	[f]
semeval2019	13240	53693	2	Offensive language prediction	Zampieri et al. (2019)[g]
sentimix	17000	89694	3	Sentiment prediction	[h]
articles	19990	285167	20	Objectivity prediction	Hajj et al. (2019)
sarcasm	28619	58779	2	Sarcasm prediction	Misra and Arora (2019)

[a]https://bitbucket.org/ssix-project/semeval-2017-task-5-subtask-2/src/master/

[b]https://www.kaggle.com/deepak711/4-subject-data-text-classification

[c]https://www.kaggle.com/c/detecting-insults-in-social-commentary/overview

[d]https://www.kaggle.com/datasnaek/mbti-type

[e]https://www.ics.uci.edu/ vpsaini/

[f] https://github.com/aitor-garcia-p/hate-speech-dataset

[g] https://sites.google.com/site/offensevalsharedtask/olid

[h]https://competitions.codalab.org/competitions/20654

be integer-rounded one third of the number of individuals. Three main variants of autoBOT are reported, i.e. autoBOT-neurosymbolic, a variant where document embeddings are evolved along with the symbolic part of the feature space and autoBOT-symbolic, a variant where the document embeddings are omitted (see Table 1). Further, autoBOT-neural evolves only the two neural representations. The time for evolution was set to 8h per data set. The time was selected from a practical viewpoint; leaving an autoML running during the night instead of having an idle machine is an option that does not require any additional time allocation at the user side. The population sizes were set to 8, the same number as the number of available cores for parallel evolution (with minimal overhead). The spectrum of linear models, evaluated during fitness evaluation was specified as follows[9]. The loss functions considered were the hinge and the log loss. The learning rate of stochastic gradient descent was set to a value from the set {0.01, 0.001, 0.0001}. The elasticnet penalty was adopted, where the ratio between L1 and L2 terms was varied in the range [0, 0.1, 0.5, 0.9, 1]. Here, if this ratio was 0, the penalty would be L2, however, if the ratio was 1, L1 penalty (lasso) would be adopted.

Finally, we discuss the data set splits considered used to evaluate the aforementioned approaches. Three different splits used for evaluation are discussed next. Each data set was split to 60% training, 20% validation and 20% testing, where the validation set was used to e.g., stop the training early on convergence when considering language models, however, as autoBOT employs cross-validation for determining the best learners, training and validation were merged—a similar scenario is computationally not feasible for language models.

4.3 Hardware and software used

The experiments were conducted using the SLING supercomputing architecture[10]. Each run was given at most 16GB of ram and 8CPU cores. autoBOT was implemented as a CPU-parallel procedure, and does not need GPU accelerators.

Additional information on the hardware used is accessible in Appendix 1. For language models benchmarks, however, specialized hardware Nvidia Tesla GPUs with 32GB of RAM (GPU) and 128GB of RAM (CPU) was used. Intentionally, we minimized the number of dependencies. Hence, Scikit-learn was used to fit linear classifiers (highly optimized) (Pedregosa et al. 2011), evolution primitives from the DEAP library (De Rainville et al. 2012) were used, and for matrix subsetting and similar linear-algebraic operations, Scipy library was adopted (Virtanen et al. 2020). The NLTK library was used for part-of-speech tagging and language parsing (Bird et al. 2009). The GENSIM library was used to obtain document embeddings (compiled versions of the algorithms) (Khosrovian et al. 2008). The language model baselines were implemented by using the PyTorch-transformers library (Wolf et al. 2020).

4.4 Evaluation of the results

Throughout the experiments we adopted the micro F1 score for multiclass classification and F1 score for binary classification. As critical distance diagrams (Demšar 2006) are currently one of the only alternatives for simultaneous comparison of multiple classifiers

[9] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[10] http://www.sling.si/

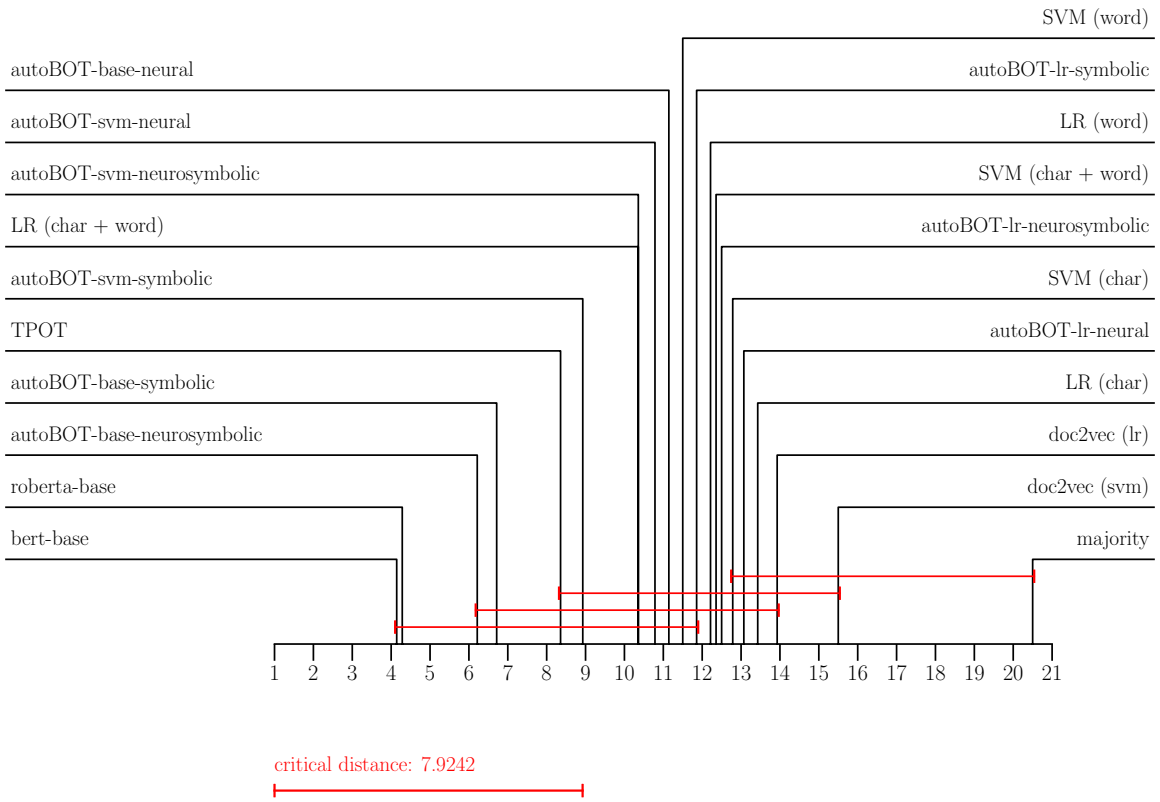


Fig. 2 Critical distance diagrams showing average ranks based on the F1 scores

across multiple data sets, we report the results by using these diagrams (for F1 and accuracy, separately) as they offer a more compact view compared to tabular results (which are reported in Appendix 2. The distance diagrams are interpreted as follows. The black lines denote the average ranks. The lower the average rank, the better the classifier. The red lines join all classifiers which are according to Friedman-Nemenyi testing part of the same significance class – there are no significant differences in their performance at ($p = 0.05$). We interpret the diagrams in alignment with the tabular results. In terms of GPERF, we visualize distributions for different data sets—such visualizations offered insights into which data sets are, given the same resources, easier or harder for the conducted evolution.

5 Results

In this section we discuss the results of empirical evaluation. We first report on classification performance in Section 5.1, followed by qualitative exploration of possible transfer learning properties of autoBOT in Section 5.2, an explainability case study in Section 5.3, and case studies of evolution's behavior in Section 5.4.

5.1 Classification performance

We summarize the F1 and accuracy-based performances in the form of critical distance diagrams, shown in Figures 2 and 3, and tabular results, shown in Tables 5 and 6 in Appendix 2. We report the results for the best performing evolution hyperparameter settings which were the mutation rate of 0.3 and the crossover rate of 0.9. It can be observed

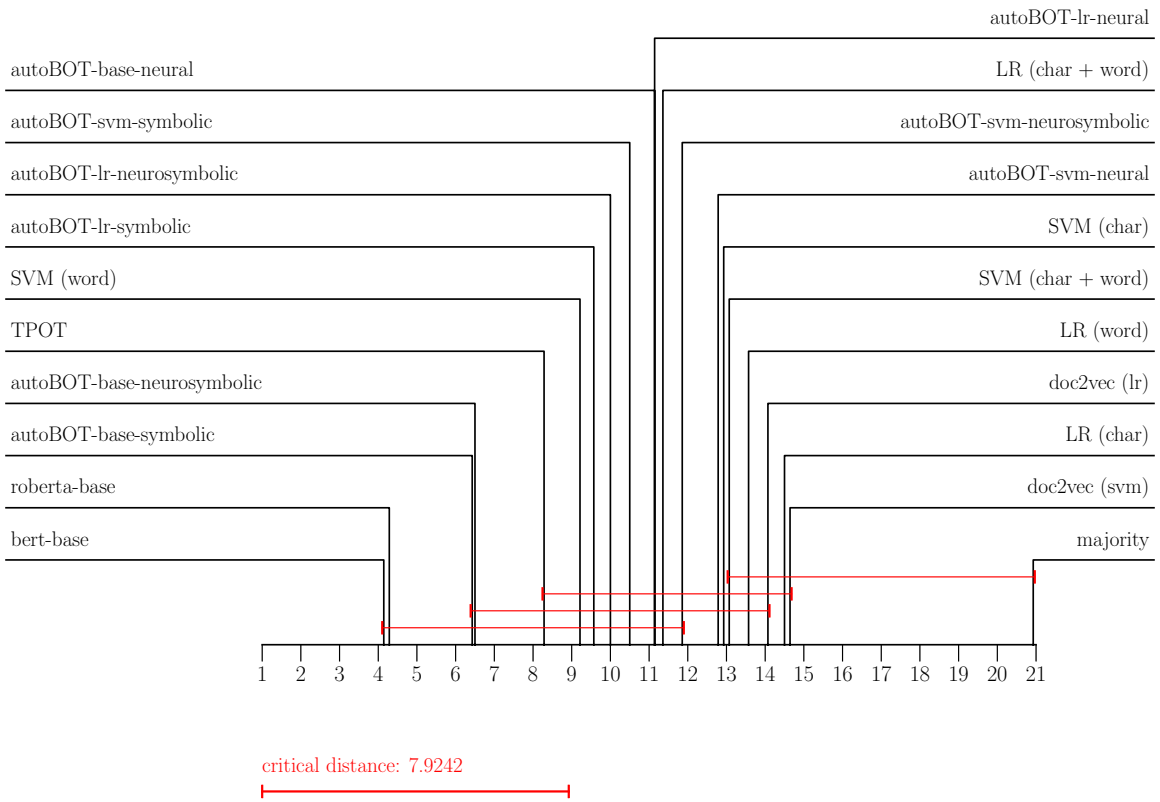


Fig. 3 Critical distance diagrams showing average ranks based on the Accuracy scores

that the proposed autoBOT-neurosymbolic performs competitively to the other state-of-the-art approaches, even though it is outperformed by BERT (and to some extent by RoBERTa). Surprisingly, the symbolic-only version of autoBOT (autoBOT-symbolic) is also highly competitive. The performance is similar if compared against TPOT, and significantly higher than the weak baselines such as the majority classifier (the red lines do not join the classifiers). We also observe that RoBERTa (125M parameters) performed marginally worse than BERT (110M parameters), which we believe is due to the fact that we did not perform extensive hyperparameter search, especially exploring various regularization settings. Another interpretation of this result is that due to the large number of parameters, overfitting on the validation set occurred. Such behavior can be problematic for low resource scenarios where many classes are predicted (e.g., *mbti*). Current results indicate that language models perform sub-optimally, if multiple classes are considered (e.g., five or more), however, the results could also be due to the class imbalance, which is present in the most multiclass problems.

The overall performance can be, based on the diagrams, summarised as follows. The neural language models, as discussed, on average out-perform other approaches. The proposed autoBOT variants including either the combination of symbolic and non-symbolic features (autoBOT-neurosymbolic) and only symbolic features (autoBOT-symbolic) are ranked next, performing on average better than e.g., TPOT (autoML baseline) and other variants of linear learners trained on the constructed representation, which, however, do not consider the evolved representation. The LR (char + word) baseline performed surprisingly well, and was, out of the weaker baselines, out-performed only by the symbolic feature space of autoBOT + SVM classifier (autoBOT-svm-symbolic). The doc2vec-only representations were amongst the worst-performing ones (doc2vec (svm) and doc2vec (lr)), indicating their potential complementarity with symbolic features (as observed

in e.g., autoBOT-base-neurosymbolic). Interestingly, if the two neural representations were evolved, the performance increased, however did not reach the neuro-symbolic combinations.

In terms of the performance across individual data sets, we highlight the following observations. The news-based data sets were rather easy to classify – in e.g., *bbc*, the strong learners all achieved around 99% accuracy. The data sets, where the discrepancy was larger, are for example the ones with more classes. One such example is the *mbti*, where TPOT outperformed the other learners, however was followed closely by the autoBOT-symbolic variant. On data sets such as *sarcasm*, the discrepancy between the neural language models and other types of methods was the largest. For example, BERT and RoBERTa achieved > 90% accuracy, the closest autoBOT implementation was again the symbolic one which scored with 82%, which is substantially lower. Interestingly, on the data sets with a large number of instances, the proposed autoBOT came within two percentage points w.r.t. the neural language models. Finally, when considering the *hatespeech* data set, the proposed autoBOT performed on par with neural language models, albeit being completely explainable, which can be the decision factor when deploying a model on a this type of task. Overall, the clear win of neural language models is in alignment with previous work (e.g., Devlin et al. (2019)), where such models performed very well across a spectrum of multiple tasks. In terms of the interpretable methods, autoBOT was shown to offer a viable alternative a user can obtain with minimal input (and setup), and no specialized hardware (GPUs in this case).

5.2 Towards meta transfer learning

As the proposed approach yields solution vectors that uniquely determine the importance of each type of features, we explored further whether the obtained solution vectors *share* properties across similar data sets. The clustered solution space is shown in Figure 4. The colors represent the scale of solution weights—weights that correspond to the individual feature types.

We observe that distinct clustering patterns emerge, roughly grouping the data sets based on the type of classification task. For example, the *yelp* and *bbc* data sets appear to have similar solutions, similarly the insults, questions and the sarcasm data sets. As we conducted two-way (hierarchical) clustering, insights into relations between *types* can also be observed. The POS and relational features appear to have the most in common, and similarly word-, character- and the keyword-based features. The two types of document embeddings behave similarly, and were recognized by autoBOT as such, which is an expected result that validates the purpose of such visualization. The image also offers insights into the question whether the embedding-based representations are always useful (assuming high weights correspond to relevance). For data sets such as *sarcasm* and *insults*, keyword and word-level features emerged with higher weights, however, when considering for example the *yelp* data set, the embedding-based representation appears to have had the most impact on the success of learning. Another apparent benefit of such visualization is the inspection of how relevant a given feature type is across multiple data sets. Current results indicate that POS tag-based features and the relational features appear to improve the predictive performance very selectively. For example, the POS tags appear to work well when considering the *sarcasm* data set, and relational features help, albeit moderately, when considering *semeval2019* and *hatespeech* data sets. We believe the visualizations like the proposed one are a very transparent option for *efficient exploration* of which

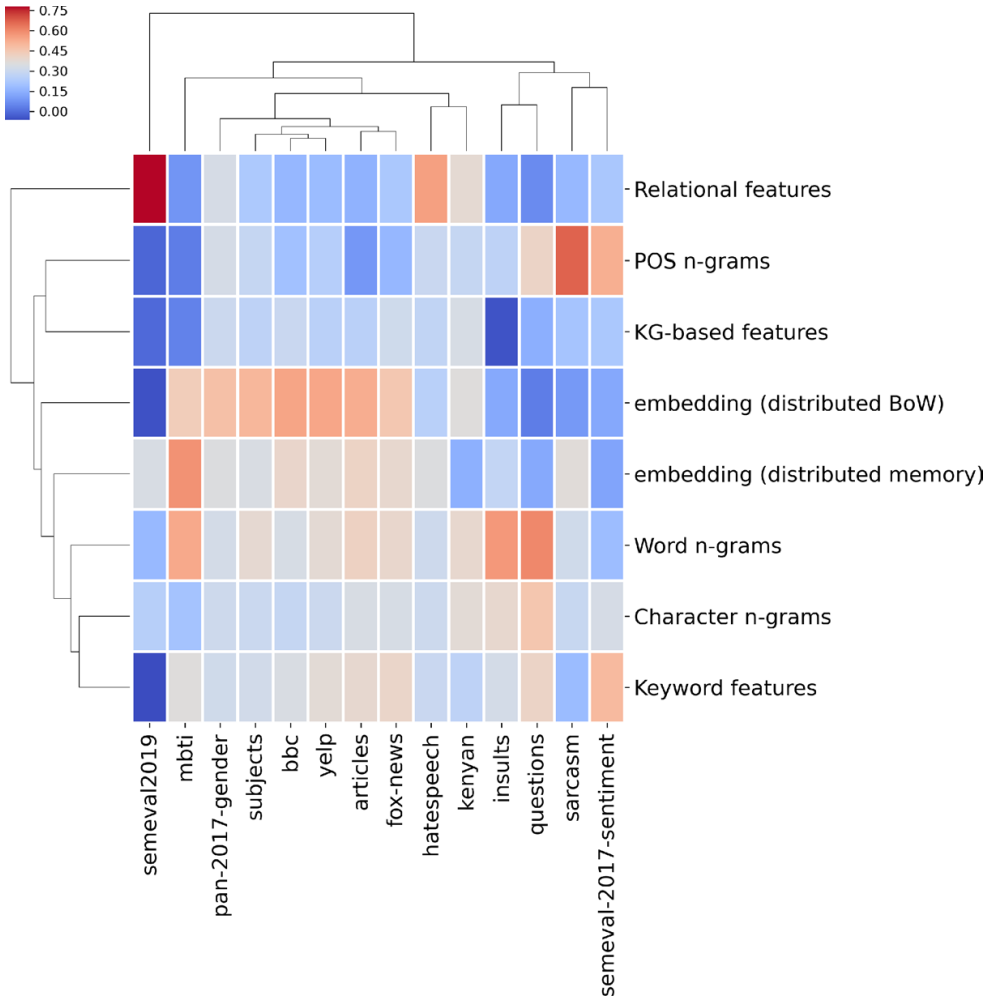


Fig. 4 Similarity of the solution vectors across considered data sets. It can be observed that data sets related to similar tasks group together, indicating potential transfer learning possibilities at the evolution solution level. The importances were re-scaled to 0-1 range

feature types carry the most information, and could be potentially further inspected (or extended). Current results indicate that the observed clustering is related to the properties of the addressed task (e.g., embedding relevance for *bbc*, *yelp* and the *articles*)

5.3 Explainability

One of the key features of autoBOT is its two-level transparency scheme. The first level corresponds to weights, representing parts of a given feature space, and can be used to understand what autoBOT emphasizes across data sets (Figure 4). However, autoBOT can also offer *direct importances*, based on the absolute coefficients of linear classifiers employed. An example for the *bbc* data set is given in Table 4. The tokens such as "blair", "election" and similar emerged as the most relevant, which is in alignment with the task that addresses differentiation between the *topics*. Note that proper nouns (nnp – noun-noun-pronoun), either one or two in a sequence, were found to be the most relevant POS tags. The table demonstrates that even though importances can be computed for each feature separately, if the feature itself is non-symbolic, such feature importances contribute very little to the interpretation (or nothing at all). Hence, we see token or knowledge graph-level features as the most relevant when attempting to interpret what impacts the autoBOT's

Table 4 Top six features for different feature subspaces (*bbc* data set). The row index corresponds to considered feature types, columns are top six features. Each cell consists of the feature name and the absolute importance extracted with autoBOT. Note that even though importances can be extracted for the embedded space, they are not informative—these features are only numbered dimensions. Note that e.g., char-based features appear the same, as the difference can be in the whitespace next to a given token (part of the feature)

Index	Char features	Word features	keyword features	POS features	Relational features	KG features	Neural features v1	Neural features v2
0	film : 0.04	iaaf : 0.12	blair : 0.16	nnp nnp : 0.02	–2–e : 0.4	atlocation(committee,government) : 0.03	1951 : 1.37	3620 : 1.19
1	ilm : 0.04	mr brown : 0.07	music : 0.16	nns : 0.02	–2–n : 0.29	hascontext (fall,uk) : 0.03	3731 : 1.21	1420 : 1.18
2	mr : 0.03	drug : 0.05	brown : 0.14	cd : 0.0	e–8–l : 0.21	hascontext (mr,uk) : 0.02	1021 : 1.15	1960 : 1.09
3	fil : 0.03	mr blair : 0.05	election : 0.12	rb : 0.0	u–2–c : 0.2	relatedto (minister,british) : 0.02	4241 : 1.13	80 : 0.99
4	mr : 0.03	g8 : 0.04	athletics : 0.1	cc : 0.0	–3–l : 0.2	relatedto (secretary,government) : 0.02	1211 : 1.09	4730 : 0.98
5	mr : 0.03	mr howard : 0.04	blackpool : 0.1	ex : 0.0	a–9–o : 0.2	synonym (minister,secretary) : 0.02	4361 : 1.05	4240 : 0.97
6	fil : 0.03	rail : 0.04	party : 0.1	in : 0.0	s–7–i : 0.2	synonym (movie,film) : 0.02	4601 : 1.03	4280 : 0.95
7	mr : 0.03	wto : 0.04	straw : 0.09	nn : 0.0	–2–r : 0.18	usedfor (film,movie) : 0.02	671 : 1.02	380 : 0.94
8	mus : 0.02	big brother : 0.03	athletes : 0.08	pos : 0.0	s–6–t : 0.18	hascontext(average,uk) : 0.01	4061 : 1.0	780 : 0.91
9	mus : 0.02	hunt : 0.03	committee : 0.08	rp : 0.0	p–2–n : 0.18	hascontext(chancellor,britain) : 0.01	3711 : 0.95	2800 : 0.91

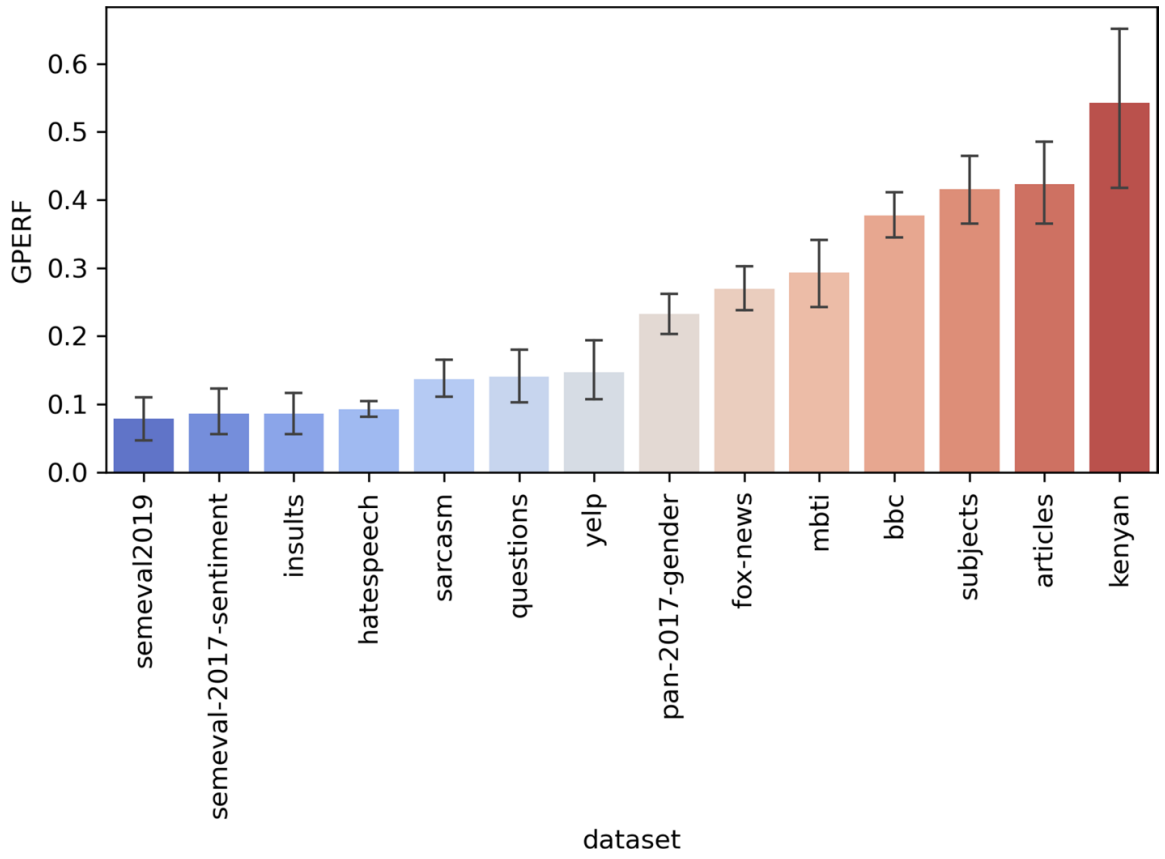


Fig. 5 GPERF across considered data sets. The standard deviations entail different hyperparameter settings (mutation, crossover)

decisions. Further, the proposed ConceptNet features also offer interesting insight into what predicates emerged as the most relevant. For example, *synonym(movie, film)* indicates the relevance of synonyms, however, the *hascontext(fall, uk)* offers insight into symbolic context, previously not considered in such setting.

5.4 The Evolution's behavior

We next present aggregations of autoBOT's GPERF scores when varying the evolution hyperparameters in Figure 5.

We observe the following. There exist distinct distribution differences among the data sets. For example, the *articles* and *subjects* data sets, and also *bbc* are characterized with high GPERF scores. On the other hand, *yelp*, *insults* and *semeval2019* data sets are on the lower end of the spectrum. As GPERF considers both the percentage of generations needed to convergence, as well as performance, we conjecture that the data sets with high GPERF are indeed *easier to learn*. For example, when considering *bbc*, both the F1 scores are above 95%, and also converge to the final maximum in the first couple of generations.

In contrast, we observe gradual evolution when considering e.g., the *insults* data set, and when this information is combined with the fact that F1 scores for this data set are lower than e.g., when considering *bbc*, we can conclude that this data set is harder to learn from and requires more time (generations). Another observation is that *fox*, *bbc* and *subjects* data sets are all focusing on topic prediction, where word-level semantics (and keywords) can play a dominant role. Note that comparison of multiple data

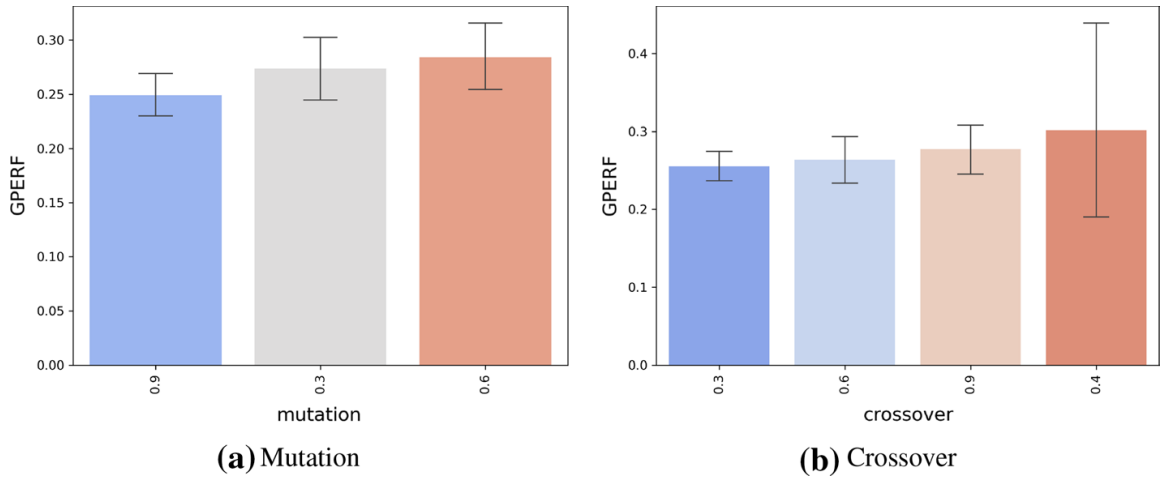


Fig. 6 Relation between GPERF and the crossover and mutation hyperparameters of evolution. Mutation of 0.3 and crossover of 0.9 offer a good trade-off between performance and evolution convergence, and were considered as the default setting

sets yields different distributions even if only performances are considered—the GPERF only offers additional insight into the nature of the evolution trace that led to a certain performance. For example, the *semeval2019*'s GPERF is very low, even though its final F1 performance is around 60%. We believe GPERF (or its variants) could serve for inspecting how the evolution progresses and potentially serve as a mechanism for *automatic stopping*, however we leave such evaluation for further work. Note also, that if autoBOT would be expected to perform well on a particular collection of data sets of the same type, this type of measurement (and visualization) would offer immediate insight into its success (e.g., detection of insults, hate speech and fake news) and potentially interesting task hardness ranking.

We next discuss the behavior of the two main hyperparameters; the crossover and mutation, on the GPERF score in Figure 6. It can be observed that very high mutation rates result in, on average, lower GPERF scores (0.3 and 0.6 yield similar results). On the contrary, current results indicate that high crossover values are beneficial for the considered problem setting.

In Figure 7 we present the interesting evolution traces we observed and discuss their implications. The figure shows four distinct evolution traces we observed when further investigating the conducted experiments. One of the key observations is that a fixed amount of time (8 hours) is not necessarily enough, and can vary highly when considering different data sets. For example, the *kenyan* data set appears relatively simple compared to e.g., the *semeval2019* data set, when gradual progress is observed, however there is no visual evidence of convergence (evolution, when considering the *kenyan* data set, converges rather quickly in the first 10% of generations). An interesting trace was observed when considering the *insults* data set, where at first larger performance increases were observed, however, when a certain point was reached, only minor improvements were present. Even though not systematically addressed, the results indicate neuro-symbolic learning is subject to *faster convergence*. Further, we acknowledge the existence of many approaches that could help with further analysis of such traces (e.g., Eiben et al. (1990)), however we consider them for further work, as the purpose of this paper was to evaluate whether autoML systems for text are feasible at all and in what scenarios.

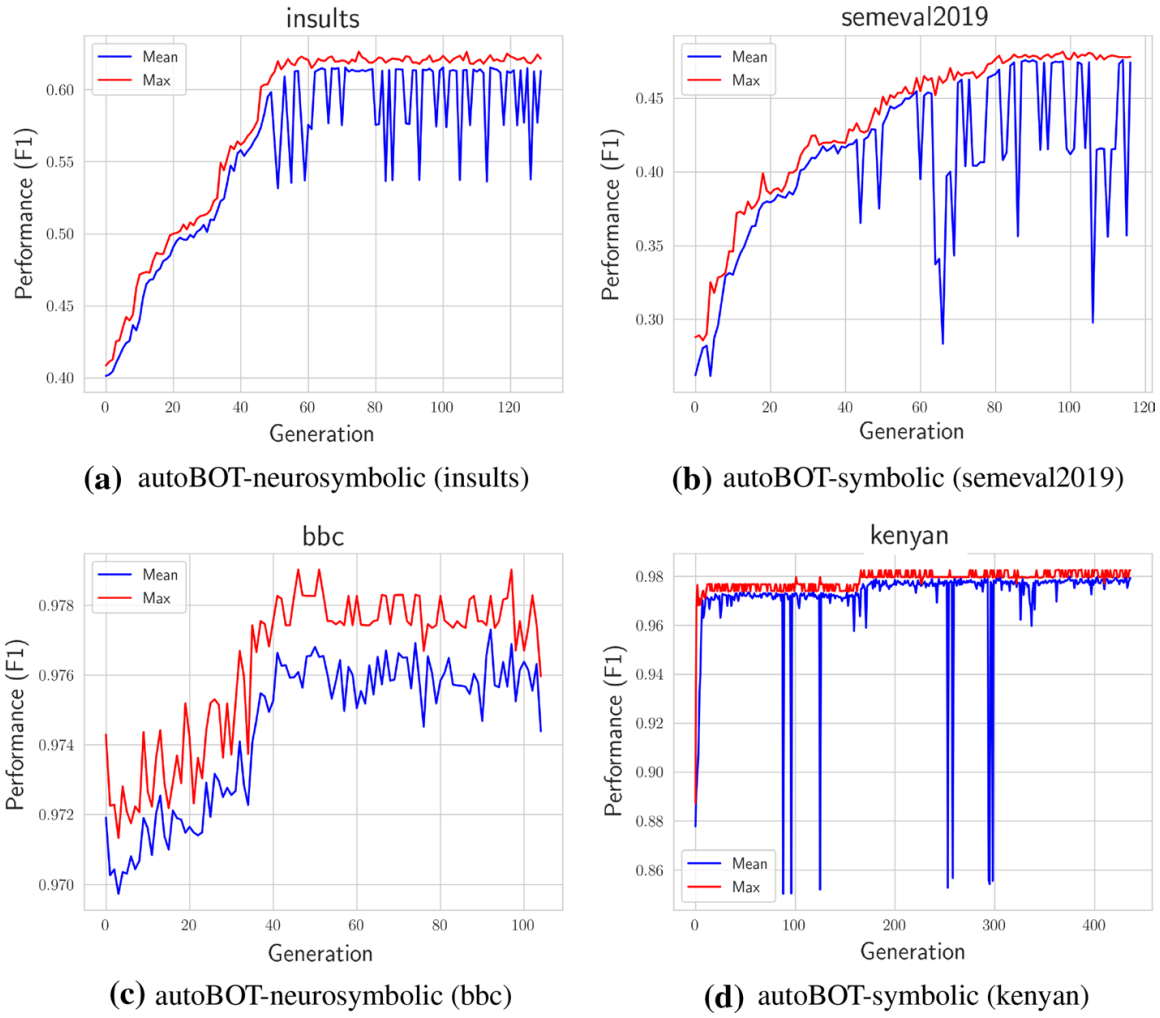


Fig. 7 Examples of evolution traces. The blue lines represent mean and red ones maximum fitness values. It can be observed (c,b) that in some cases, the dedicated evolution time of 8 hours, was not necessarily enough to achieve convergence. On the other hand, as seen for example when considering the *kenyan* data set (d), relatively fast convergence is observed due to a relatively simple classification task. The evolution either gradually unveils a relevant representation (b), or in a few generations, as can be seen in (d)

6 Discussion and conclusions

The focus of this paper is the proposed autoBOT system for automatic learning of classifiers and representations for texts. We demonstrate the system's competitive performance on multiple data sets, when compared to strong baselines such as other autoML systems or *neural*, transformer-based language models. We additionally investigate the evolution's behavior for selected examples, showing that instead of evolving a heterogeneous ensemble of learners, as performed by existing state-of-the-art approaches, evolution on the representation level proves to be a feasible and computationally more sensible option.

The proposed autoBOT system currently considers six symbolic and two non-symbolic document representations, however it is by no means limited to feature types considered in this work—these were selected to take multiple possible text representations into account, as well as to explore potentially interesting implications for meta transfer learning, where the solution vectors could be directly transferred across similar problems. As part of the future work, we believe incorporation of translational distance-based features could also be a promising approach. Here, a feature would be a conjunct of e.g., pairs of

presentAtDistance predicates, which approximate the distance between the considered pair of tokens. This type of features could potentially entail more complex relations between tokens that can be otherwise hard to detect.

The proposed autoBOT approach can also be considered in analogy to the attention mechanism, used in contemporary transformer-based architectures (Devlin et al. 2019). The neural attention, during backpropagation, *prioritizes* parts of the byte pair encoded space, yielding sparse signals that are highly dependent on the context. The evolution, as implemented in this work, effectively optimizes a single vector of weights, each corresponding to a particular *collection* of features. Similarly to the attention, however, particular collections are left out (e.g., character-level features when considering semantics-rich texts). In this way, the evolution is responsible for distillation of the feature space (and not backpropagation). Finally, we believe that also the granularity of the considered space is different. While the attention mechanism emphasized e.g., individual tokens (or pairs), the autoBOT importances are related to larger feature subsets related to feature types.

Even though the proposed implementation of autoBOT is not meant for online execution, a potentially interesting research direction would be its adaptation for operation with e.g., *data streams*. Here, we see two main opportunities on how this setting could be considered. First, the existing, pre-initialized evolution weight space could be used to evolve a collection of classifiers just for a few iterations, potentially adapting to the new properties of the data, and second, as the learners are trained with stochastic gradient descent, their weights could be updated in a minibatch manner; in this scenario, the evolution iteration would not be considered after each learning update but more seldom, lifting the potentially time expensive re-training.

The proposed dimensionality estimation procedure operates based on a simple assumption that there exist useful high-dimensional feature spaces that have the same memory footprint as the commonly used low-dimensional ones (e.g., of size 128). This intuitively means that one can select the dimensions with the spatial footprint of a reasonable size, e.g., a 128 dimensional dense representation (the dimension is a hyperparameter), for which we already got an insight into its behavior on a given hardware. The estimation assumes the same dimension for all feature types, making it possible to happen that e.g., there are fewer POS-based features than the estimated dimension permits. This could be solved via some form of dynamic assignment procedure, despite the apparently low expected effect on the overall performance.

In terms of computational load, we observed the following. As the proposed autoBOT was developed with sparse representation structure in mind, its memory footprint never exceeded that of available in individual cluster jobs (16GB). As the runtime is coupled with the parameter denoting the time, current results indicate that in 8h (e.g., over-night), autoBOT is able to find good classifiers, an explanation as to what are the relevant parts of the feature space, and the features themselves that matter for the final classification. We observed that even though TPOT performs competitively, it is not able to leverage the sparseness of input matrices, resulting in potentially high memory overhead. Finally, as the neural language models were evaluated on specialized hardware, and could not be easily fine-tuned on an off-the-shelf laptop due to high working memory, disk and computation requirements, we believe this branch of models does not cover all the low-resource scenarios in which symbolic or neuro-symbolic approaches should operate well.

In terms of explainability, the proposed autoBOT offers insight into feature type and feature-level importances that are jointly learned. Potentially, a similar level of explainability can be obtained by combining explanations based on linear learners that learn based on individual features in conjunction with learners that learn on the subspaces governed by the separate feature types. The main difference between the two paradigms is that the

feature-type weights are obtained by evolution, offering potentially easier incorporation of additional type-related constraints or simultaneous consideration of multiple objectives related to a given representation's properties. The bags-of-features-based approaches can be, on the contrary, faster and are potentially an interesting future research direction in terms of weight screening prior to the main, more computationally intense evolution part. We leave a more detailed study of the explanatory power and combinations of the two paradigms for further work. Note that the evolution performs feature selection only in the scenario where the weights are exactly zero (for a given type). This type of features will be omitted entirely during classification (extreme feature discarding). In most of the experiments conducted to this end, the evolution merely re-weighted parts of the feature space, which is used in a regularization-based approach (as part of the fitness function). Even though document embeddings could be obtained with existing language models, and potentially further improve the performance, such implementation would defeat the current purpose of autoBOT, which emphasizes low resource learning. To our knowledge current state-of-the-art language models (e.g., RoBERTa) are not yet necessarily suitable for commodity hardware, even though due to increasingly more computational power, this statement might change in the future. Overall, as autoBOT was built with modular representation learning in mind, should the need arise, contextual document space could also be included as one of the considered feature types (see Section 3.1). Further, we observed that large language models struggle with problems where the amount of data is not large, and there are many classes (e.g., *mbti*). Such behaviour will be further studied, as it is not clear whether this is a general limitation.

One of the emphasis of this paper is autoBOT's capability to operate on sparse spaces. The sparsity of the considered document representations can be the result of two different procedures. First, the classifier, evolved as part of the evolution is regularized so that it potentially prunes out parts of the feature space. One of the classifiers explored as a part of each individual is also lasso, hence the classifier-based sparseness is obtained if the classifier performs well. Further, sparseness can also be induced at the representation level by the evolution itself; here, typed parts of the feature space can be jointly neglected (weight $= 0$) if e.g., character-based features are non-informative.

Current autoBOT implementation considers very basic evolution principles, known for at least 30 years. This choice is intentional, aiming to demonstrate that by considering a simple tournament-based evolution with mutation and crossover, the system already offers competitive performance. An apparent direction of future work is thus to explore more advanced evolution schemes, including the exploration of Pareto optimal representations (as for example discussed by Deb and Jain (2013))—simultaneous optimization of multiple metrics could be beneficial in many real-life scenarios (Ishibuchi et al. 2008), and shall be considered in future work.

Another design choice of autoBOT was the adoption of simple, well regularized linear learners instead of more computationally intensive ones. This choice was due to the emphasis on representation evolution, which can otherwise be out-sourced to the model itself (e.g., with deeper neural network models). Furthermore, the current implementation of autoBOT offers relatively simple (drop-in replacement) exploration of more involved models, which we leave for further work.

Finally, as the main result of this work we recognize the autoBOT's performance to offer reasonable results with *zero* human hyperparameter tuning, while at the same time offering insights into which parts of the input space, either at the level of feature types, or at the level of individual features is relevant. Even though we employed simple coefficient normalization, we believe importance assessment can already be useful for low-risk scenarios such as e.g., model debugging for news classification, however more involved

normalization schemes with statistical guarantees should be adopted if systems of this type were to be used in more high-risk (e.g., biomedical) domains. The proposed implementation offers a straightforward way of obtaining relatively strong classifiers with as little human input as possible, whilst remaining interpretable.

Appendix 1: Hardware used for neural language model training

The following is the hardware specification of the machine used for training neural language models. Note that GPUs were not used for autoBOT, as it performs as a parallel, CPU-only algorithm.

```
    ############################ CPU ###################################
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              6
On-line CPU(s) list: 0-5
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           6
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Gold 6150 CPU @ 2.70GHz
Stepping:            4
CPU MHz:             2700.000
BogoMIPS:            5400.00
Hypervisor vendor:   VMware
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            1024K
L3 cache:            25344K
NUMA node0 CPU(s):   0-5

###################### RAM ####################################
128GB

###################### GPU for NLMs ############################
Tue Nov 10 07:49:57 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 430.26       Driver Version: 430.26       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:03:00.0 Off |                    0 |
| N/A   31C    P0    51W / 300W |      0MiB / 32510MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Table 5 Macro F1 performance across data sets and classifiers

dataset model	articles	bbc	fox	hatespeech	insults	kenyan	mbti	pan-2017	questions	sarcasm	seme-val-2017	seme-val2019	subjects	yelp
dummy-stratified	0.05 (0.0)	0.31 (0.02)	0.18 (0.01)	0.77 (0.0)	0.3 (0.02)	0.53 (0.03)	0.14 (0.01)	0.52 (0.11)	0.2 (0.01)	0.49 (0.01)	0.38 (0.01)	0.34 (0.0)	0.37 (0.03)	0.29 (0.01)
LR (word)	0.62 (0.0)	0.96 (0.0)	0.86 (0.0)	0.84 (0.0)	0.58 (0.0)	0.96 (0.0)	0.58 (0.0)	0.77 (0.24)	0.78 (0.0)	0.78 (0.0)	0.51 (0.0)	0.54 (0.02)	0.96 (0.0)	0.49 (0.0)
SVM (word)	0.63 (0.0)	0.96 (0.0)	0.86 (0.0)	0.81 (0.0)	0.63 (0.0)	0.97 (0.0)	0.65 (0.0)	0.82 (0.26)	0.82 (0.0)	0.78 (0.0)	0.45 (0.0)	0.53 (0.03)	0.95 (0.0)	0.49 (0.0)
LR (char)	0.68 (0.0)	0.95 (0.0)	0.8 (0.0)	0.83 (0.0)	0.64 (0.0)	0.95 (0.0)	0.4 (0.0)	0.8 (0.22)	0.76 (0.0)	0.77 (0.0)	0.42 (0.0)	0.54 (0.0)	0.98 (0.0)	0.48 (0.0)
SVM (char)	0.69 (0.0)	0.94 (0.0)	0.82 (0.0)	0.83 (0.0)	0.62 (0.0)	0.96 (0.0)	0.48 (0.0)	0.79 (0.24)	0.78 (0.0)	0.77 (0.0)	0.43 (0.0)	0.56 (0.04)	0.98 (0.0)	0.46 (0.0)
LR (char + word)	0.7 (0.0)	0.96 (0.0)	0.86 (0.0)	0.83 (0.0)	0.64 (0.0)	0.91 (0.0)	0.65 (0.0)	0.81 (0.26)	0.82 (0.0)	0.81 (0.0)	0.42 (0.0)	0.56 (0.0)	0.97 (0.0)	0.5 (0.0)
SVM (char + word)	0.64 (0.0)	0.95 (0.0)	0.88 (0.0)	0.81 (0.0)	0.59 (0.01)	0.97 (0.0)	0.5 (0.0)	0.75 (0.21)	0.78 (0.0)	0.8 (0.04)	0.54 (0.0)	0.56 (0.03)	0.98 (0.0)	0.43 (0.0)
bert-base	**0.84 (0.0)**	**0.99 (0.0)**	**1.0 (0.0)**	0.88 (0.0)	**0.78 (0.01)**	**1.0 (0.02)**	0.33 (0.09)	0.68 (0.16)	**0.96 (0.0)**	0.92 (0.0)	0.67 (0.01)	**0.67 (0.01)**	0.99 (0.01)	**0.58 (0.01)**
roberta-base	0.82 (0.0)	**0.99 (0.0)**	**1.0 (0.0)**	**0.89 (0.01)**	0.77 (0.01)	0.99 (0.02)	0.26 (0.07)	0.69 (0.15)	**0.96 (0.0)**	0.93 (0.0)	**0.71 (0.13)**	0.67 (0.25)	0.98 (0.0)	0.56 (0.01)
TPOT	0.64 (0.0)	0.97 (0.0)	0.93 (0.01)	0.84 (0.0)	0.62 (0.01)	0.97 (0.0)	**0.67 (0.01)**	0.82 (0.22)	0.82 (0.0)	0.8 (0.0)	0.53 (0.0)	0.40 (0.0)	0.97 (0.0)	0.53 (0.0)
doc2vec (lr)	0.65 (0.0)	0.98 (0.01)	0.77 (0.01)	0.82 (0.0)	0.39 (0.01)	0.97 (0.01)	0.54 (0.01)	0.81 (0.23)	0.47 (0.0)	0.75 (0.0)	0.34 (0.0)	0.36 (0.01)	0.95 (0.01)	0.49 (0.01)
doc2vec (svm)	0.64 (0.0)	0.97 (0.01)	0.7 (0.01)	0.82 (0.0)	0.39 (0.01)	0.95 (0.01)	0.5 (0.01)	0.79 (0.22)	0.54 (0.01)	0.76 (0.0)	0.34 (0.0)	0.37 (0.01)	0.95 (0.01)	0.47 (0.0)
autoBOT-lr-neural	0.81 (0.0)	**0.99 (0.0)**	0.85 (0.0)	0.81 (0.0)	0.28 (0.02)	0.95 (0.0)	0.57 (0.0)	0.83 (0.21)	0.53 (0.01)	0.7 (0.0)	0.45 (0.0)	0.36 (0.01)	0.98 (0.0)	0.52 (0.0)
autoBOT-svm-neural	0.81 (0.0)	0.98 (0.0)	0.84 (0.01)	0.82 (0.0)	0.49 (0.01)	0.97 (0.0)	0.59 (0.01)	0.82 (0.19)	0.56 (0.0)	0.74 (0.0)	0.47 (0.01)	0.48 (0.01)	0.98 (0.0)	0.49 (0.0)

Table 5 (continued)

dataset model	articles	bbc	fox	hatespeech	insults	kenyan	mbti	pan-2017	questions	sarcasm	seme-val-2017	seme-val2019	subjects	yelp
autoBOT-lr-sym-bolic	0.81 (0.0)	0.98 (0.0)	0.85 (0.0)	0.81 (0.0)	0.29 (0.02)	0.96 (0.0)	0.58 (0.0)	0.83 (0.21)	0.54 (0.01)	0.7 (0.0)	0.46 (0.01)	0.37 (0.01)	0.97 (0.0)	0.52 (0.01)
autoBOT-svm-symbolic	0.81 (0.0)	0.98 (0.0)	0.84 (0.01)	0.82 (0.0)	0.48 (0.01)	0.97 (0.0)	0.59 (0.01)	0.82 (0.19)	0.56 (0.01)	0.74 (0.0)	0.47 (0.01)	0.48 (0.01)	**0.99 (0.0)**	0.49 (0.01)
autoBOT-lr-neuro-symbolic	0.81 (0.0)	**0.99 (0.0)**	0.84 (0.0)	0.81 (0.0)	0.29 (0.02)	0.96 (0.0)	0.58 (0.0)	0.83 (0.21)	0.54 (0.01)	0.7 (0.0)	0.46 (0.01)	0.36 (0.01)	0.97 (0.0)	0.52 (0.0)
autoBOT-svm-neuros-ymbolic	0.81 (0.0)	0.98 (0.0)	0.84 (0.0)	0.82 (0.0)	0.48 (0.02)	0.97 (0.0)	0.58 (0.0)	0.82 (0.19)	0.56 (0.0)	0.74 (0.0)	0.47 (0.01)	0.47 (0.0)	0.98 (0.0)	0.49 (0.01)
autoBOT-base-neural	0.8 (0.0)	**0.99 (0.0)**	0.86 (0.01)	0.82 (0.0)	0.49 (0.08)	0.97 (0.01)	0.6 (0.01)	**0.84 (0.21)**	0.37 (0.06)	0.67 (0.02)	0.34 (0.08)	0.46 (0.06)	**0.99 (0.0)**	0.52 (0.0)
autoBOT-base-neuros-ymbolic	0.8 (0.01)	**0.99 (0.0)**	0.86 (0.01)	0.82 (0.01)	0.63 (0.04)	0.97 (0.01)	0.61 (0.01)	**0.84 (0.21)**	0.78 (0.02)	0.79 (0.01)	0.54 (0.03)	0.57 (0.04)	**0.99 (0.0)**	0.52 (0.01)
autoBOT-base-symbolic	0.78 (0.01)	**0.99 (0.0)**	0.88 (0.01)	0.82 (0.01)	0.66 (0.04)	0.96 (0.01)	0.63 (0.0)	0.83 (0.23)	0.79 (0.01)	0.82 (0.01)	0.56 (0.04)	0.63 (0.06)	**0.99 (0.0)**	0.48 (0.01)

Table 6 Accuracy performance across data sets and classifiers

dataset model	articles	bbc	fox	hatespeech	insults	kenyan	mbti	pan-2017	questions	sarcasm	seme-val-2017	seme-val2019	subjects	yelp
dummy-stratified	0.05 (0.0)	0.32 (0.02)	0.18 (0.01)	0.77 (0.0)	0.63 (0.01)	0.54 (0.04)	0.14 (0.01)	0.53 (0.11)	0.2 (0.01)	0.51 (0.01)	0.38 (0.01)	0.56 (0.01)	0.38 (0.03)	0.29 (0.01)
LR (word)	0.62 (0.0)	0.96 (0.0)	0.86 (0.0)	0.87 (0.0)	0.78 (0.0)	0.96 (0.0)	0.59 (0.0)	0.77 (0.24)	0.78 (0.0)	0.79 (0.0)	0.53 (0.0)	0.73 (0.01)	0.97 (0.0)	0.5 (0.0)
SVM (word)	0.64 (0.0)	0.96 (0.0)	0.86 (0.0)	0.87 (0.0)	0.84 (0.0)	0.97 (0.0)	**0.66 (0.0)**	0.82 (0.25)	0.81 (0.0)	0.8 (0.0)	0.54 (0.0)	0.73 (0.01)	0.96 (0.0)	0.51 (0.0)
LR (char)	0.68 (0.0)	0.95 (0.0)	0.8 (0.0)	0.87 (0.0)	0.83 (0.0)	0.95 (0.0)	0.43 (0.0)	0.8 (0.22)	0.76 (0.0)	0.78 (0.0)	0.53 (0.0)	0.73 (0.0)	0.98 (0.0)	0.49 (0.0)
SVM (char)	0.69 (0.0)	0.94 (0.0)	0.82 (0.0)	0.87 (0.0)	0.83 (0.0)	0.96 (0.0)	0.5 (0.0)	0.8 (0.24)	0.78 (0.0)	0.78 (0.0)	0.53 (0.0)	0.73 (0.04)	0.98 (0.0)	0.47 (0.0)
LR (char + word)	0.7 (0.0)	0.96 (0.0)	0.86 (0.0)	0.87 (0.0)	0.83 (0.0)	0.9 (0.0)	0.66 (0.0)	0.81 (0.26)	0.82 (0.0)	0.82 (0.0)	0.53 (0.0)	0.73 (0.0)	0.97 (0.0)	0.51 (0.0)
SVM (char + word)	0.64 (0.0)	0.95 (0.0)	0.88 (0.0)	0.81 (0.0)	0.79 (0.0)	0.97 (0.0)	0.5 (0.0)	0.75 (0.21)	0.78 (0.0)	0.8 (0.02)	0.54 (0.0)	0.73 (0.04)	0.98 (0.0)	0.43 (0.0)
bert-base	**0.84 (0.0)**	**0.99 (0.0)**	**1.0 (0.0)**	0.89 (0.0)	0.89 (0.0)	**1.0 (0.02)**	0.34 (0.04)	0.68 (0.15)	**0.96 (0.0)**	0.92 (0.0)	0.68 (0.01)	0.78 (0.01)	0.99 (0.01)	**0.58 (0.01)**
roberta-base	0.82 (0.0)	**0.99 (0.0)**	**1.0 (0.0)**	**0.9 (0.01)**	0.88 (0.01)	0.99 (0.02)	0.27 (0.02)	0.68 (0.12)	**0.96 (0.0)**	**0.93 (0.0)**	**0.72 (0.08)**	**0.78 (0.04)**	0.98 (0.0)	0.56 (0.01)
TPOT	0.64 (0.0)	0.97 (0.0)	0.93 (0.01)	0.87 (0.0)	0.83 (0.0)	0.97 (0.0)	0.68 (0.01)	0.83 (0.18)	0.82 (0.0)	0.8 (0.0)	0.57 (0.0)	0.7 (0.0)	0.98 (0.0)	0.53 (0.0)
doc2vec (lr)	0.66 (0.0)	0.97 (0.01)	0.77 (0.01)	0.87 (0.0)	0.76 (0.0)	0.97 (0.01)	0.53 (0.01)	0.8 (0.22)	0.47 (0.0)	0.76 (0.0)	0.51 (0.0)	0.71 (0.0)	0.95 (0.01)	0.5 (0.01)
doc2vec (svm)	0.66 (0.0)	0.97 (0.01)	0.7 (0.01)	0.87 (0.0)	0.77 (0.0)	0.95 (0.01)	0.48 (0.01)	0.78 (0.22)	0.55 (0.01)	0.77 (0.0)	0.51 (0.0)	0.72 (0.0)	0.95 (0.01)	0.49 (0.0)
autoBOT-lr-neural	0.82 (0.0)	**0.99 (0.0)**	0.85 (0.0)	0.87 (0.0)	0.77 (0.0)	0.95 (0.0)	0.62 (0.0)	0.82 (0.18)	0.53 (0.01)	0.73 (0.0)	0.55 (0.0)	0.71 (0.0)	0.98 (0.0)	0.53 (0.0)
autoBOT-svm-neural	0.81 (0.0)	0.98 (0.0)	0.84 (0.01)	0.86 (0.0)	0.78 (0.0)	0.97 (0.0)	0.6 (0.01)	0.81 (0.17)	0.56 (0.01)	0.75 (0.0)	0.53 (0.01)	0.68 (0.01)	0.98 (0.0)	0.49 (0.0)

Table 6 (continued)

dataset model	articles	bbc	fox	hatespeech	insults	kenyan	mbti	pan-2017	questions	sarcasm	seme-val-2017	seme-val2019	subjects	yelp
autoBOT-lr-sym-bolic	0.82 (0.0)	0.98 (0.0)	0.85 (0.0)	0.87 (0.0)	0.78 (0.0)	0.96 (0.0)	0.62 (0.0)	0.82 (0.18)	0.54 (0.01)	0.72 (0.0)	0.56 (0.01)	0.71 (0.0)	0.98 (0.0)	0.53 (0.01)
autoBOT-svm-symbolic	0.81 (0.0)	0.98 (0.0)	0.84 (0.01)	0.86 (0.0)	**0.78 (0.01)**	0.97 (0.0)	0.6 (0.01)	0.81 (0.17)	0.56 (0.01)	0.75 (0.0)	0.53 (0.01)	0.69 (0.01)	**0.99 (0.0)**	0.5 (0.01)
autoBOT-lr-neuro-symbolic	0.82 (0.0)	**0.99 (0.0)**	0.84 (0.0)	0.87 (0.0)	0.78 (0.0)	**0.96 (0.0)**	0.62 (0.0)	0.82 (0.18)	0.54 (0.01)	0.73 (0.0)	0.56 (0.0)	0.71 (0.0)	0.98 (0.0)	0.53 (0.0)
autoBOT-svm-neuros-ymbolic	0.81 (0.0)	0.98 (0.0)	0.84 (0.0)	0.86 (0.0)	0.77 (0.0)	0.97 (0.0)	0.6 (0.0)	0.82 (0.17)	0.56 (0.0)	0.75 (0.0)	0.53 (0.01)	0.68 (0.0)	0.98 (0.0)	0.49 (0.01)
autoBOT-base-neural	0.81 (0.0)	**0.99 (0.0)**	0.86 (0.01)	0.87 (0.0)	0.77 (0.01)	0.97 (0.01)	0.61 (0.01)	**0.84 (0.2)**	0.37 (0.04)	0.68 (0.0)	0.51 (0.09)	0.71 (0.0)	**0.99 (0.0)**	0.54 (0.0)
autoBOT-base-neuros-ymbolic	0.81 (0.0)	**0.99 (0.0)**	0.86 (0.01)	**0.87 (0.03)**	0.82 (0.02)	0.97 (0.01)	0.61 (0.01)	**0.84 (0.2)**	0.78 (0.02)	0.79 (0.01)	0.57 (0.03)	0.72 (0.05)	**0.99 (0.0)**	0.54 (0.01)
autoBOT-base-symbolic	0.79 (0.01)	**0.99 (0.0)**	0.89 (0.01)	0.86 (0.02)	0.84 (0.03)	0.96 (0.01)	0.65 (0.0)	0.83 (0.21)	0.79 (0.01)	0.82 (0.02)	0.58 (0.04)	0.77 (0.06)	**0.99 (0.0)**	0.52 (0.01)

Appendix 2: Tabular results

Acknowledgements The work of the first author was funded by the Slovenian Research Agency through a young researcher grant. The work of other authors was supported by the Slovenian Research Agency (ARRS) core research programme *Knowledge Technologies* (P2-0103), an ARRS funded research project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078) and European Union's Horizon 2020 research and innovation programme under grant agreement No 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media). We also gratefully acknowledge the support of NVIDIA Corporation for the donation of Titan-XP GPU. This research was also partially supported by TAILOR (a project funded by the EU Horizon 2020 research and innovation programme under GA No 952215) and AI4EU (GA No 825619). We would also like to thank the reviewers for their valuable comments.

Data availability autoBOT's repository will be available at https://github.com/SkBlaz/autoBOT.

References

Agarwal, B., Mittal, N. (2014) Text classification using machine learning methods - A survey. In: *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012* (pp. 701–709). Springer.

Belinkov, Y., & Glass, J. (2019). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics, 7,* 49–72.

Beyer, H. G., Schwefel, H. P., & Wegener, I. (2002). How to analyse evolutionary algorithms. *Theoretical Computer Science, 287*(1), 101–130.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: Analyzing text with the natural language toolkit*. California: O'Reilly Media Inc.

Bougouin, A., Boudin, F., Daille, B. (2013) TopicRank: Graph-based topic ranking for keyphrase extraction. In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing* (pp. 543–551). Asian Federation of Natural Language Processing, Nagoya, Japan.

Campos, R., Mangaravite, V., Pasquali, A., Jorge, A. M., Nunes, C., & Jatowt, A. (2018). A text feature based automatic keyword extraction method for single documents. In G. Pasi, B. Piwowarski, L. Azzopardi, & A. Hanbury (Eds.), *Advances in Information Retrieval* (pp. 684–691). Germany: Springer.

Chambers, L. D. (2000). *The Practical Handbook of Genetic Algorithms: Applications*. Florida: CRC Press.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology, 2*(3), 1–27.

Davis, L. (Ed.). (1991). *Handbook of Genetic Algorithms*. London: Chapman & Hall.

De Rainville, F.M., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C. (2012) Deap: A python framework for evolutionary algorithms. In: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation* (pp. 85–92).

Deb, K., & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE transactions on evolutionary computation, 18*(4), 577–601.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research., 7,* 1–30.

Denysiuk, R., Gaspar-Cunha, A., & Delbem, A. C. (2019). Neuroevolution for solving multiobjective knapsack problems. *Expert Systems with Applications, 116,* 65–77.

Devlin, J., Chang, M.W., Lee, K., Toutanova, K. (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186) . Minneapolis, Minnesota : Association for Computational Linguistics.

Dorronsoro, B., Pinel, F. (2017) Combining machine learning and genetic algorithms to solve the independent tasks scheduling problem. In: *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)* (pp. 1–8). IEEE.

Dua, D., Graff, C. (2017) UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.

Eiben, A.E., Aarts, E.H., Van Hee, K.M. (1990) Global convergence of genetic algorithms: A Markov chain analysis. In: *Proceedings of the International Conference on Parallel Problem Solving from Nature* (pp. 3–12). Springer.

El-Beltagy, S. R., & Rafea, A. (2009). KP-Miner: A keyphrase extraction system for English and Arabic documents. *Information Systems, 34*(1), 132–144.

English, T.M. (1996) Evaluation of evolutionary and genetic optimizers: No free lunch. In: *Evolutionary Programming* (pp. 163–169).

Fellbaum, C. (2012) WordNet. The Encyclopedia of Applied Linguistics.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F. (2019) Auto-sklearn: Efficient and robust automated machine learning. In: textitAutomated Machine Learning (pp. 113–134). Springer.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The Elements of Statistical Learning* (Vol. 1). New York, USA: Springer Series. (**in Statistics**).

Gijsbers, P., & Vanschoren, J. (2019). Gama: Genetic automated machine learning assistant. *Journal of Open Source Software, 4*(33), 1132.

Greene, D., Cunningham, P. (2006) Practical solutions to the problem of diagonal dominance in kernel document clustering. In: W.W. Cohen, A.W. Moore (eds.) Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006, *ACM International Conference Proceeding Series* (pp. 377–384). ACM.

Hajj, N., Rizk, Y., & Awad, M. (2019). A subjectivity classification framework for sports articles using improved cortical algorithms. *Neural Computing and Applications, 31*(11), 8069–8085.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S. (2018) Amc: Automl for model compression and acceleration on mobile devices. In: *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 784–800).

Ishibuchi, H., Tsukamoto, N., Nojima, Y. (2008) Evolutionary many-objective optimization: A short review. In: *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (pp. 2419–2426). IEEE.

Jennings, P. C., Lysgaard, S., Hummelshøj, J. S., Vegge, T., & Bligaard, T. (2019). Genetic algorithms for computational materials discovery accelerated by machine learning. *NPJ Computational Materials, 5*(1), 1–6.

Jing, K., Xu, J. (2019) A survey on neural network language models. arXiv preprint arXiv:1906.03591

Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017) In-datacenter performance analysis of a tensor processing unit. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture* (pp. 1–12).

Khosrovian, K., Pfahl, D., Garousi, V. (2008) Gensim 2.0: A customizable process simulation model for software process evaluation. In: *Proceedings of the International Conference on Software Process* (pp. 294–306). Springer.

Kipf, T.N., Welling, M. (2017) Semi-supervised classification with graph convolutional networks. In: *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Komer, B., Bergstra, J., Eliasmith, C. (2014) Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In: *ICML workshop on AutoML* (p. 50). Citeseer.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0?: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research, 18*(25), 1–5.

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information, 10*(4), 150.

Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning, 109*(7), 1465–1507.

Le, Q.V., Mikolov, T. (2014) Distributed representations of sentences and documents. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, JMLR Workshop and Conference Proceedings* vol. 32 (pp. 1188–1196). JMLR.org.

Li, X., Roth, D. (2002) Learning question classifiers. In: *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, vol. 1 (pp. 1–7).

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V. (2019) RoBERTa: A robustly optimized BERT pretraining approach.

Madrid, J. (2019) Autotext: AutoML for text classification. https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/1950/1/MadridPJG.pdf

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Scoring, term weighting and the vector space model. *Introduction to information retrieval, 100,* 2–4.

Martinc, M., Škrjanec, I., Zupan, K., Pollak, S. (2017) Pan 2017 Author profiling - gender and language variety prediction. In: *Working Notes Papers of the CLEF*.

Mihalcea, R., Tarau, P. (2004) TextRank: Bringing order into text. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (pp. 404–411). Barcelona, Spain: Association for Computational Linguistics.

Mirończuk, M. M., & Protasiewicz, J. (2018). A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications, 106,* 36–54.

Misra, R., Arora, P. (2019) Sarcasm detection using hybrid neural network.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

Mohr, F., Wever, M., & Hüllermeier, E. (2018). Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning, 107*(8), 1495–1515.

Moradi, M., Dorffner, G., & Samwald, M. (2020). Deep contextualized embeddings for quantifying the informative content in biomedical text summarization. *Computer Methods and Programs in Biomedicine, 184,* 105117.

Myers, I. B. (1962). *The Myers-Briggs Type Indicator: Manual*. Germany: Consulting Psychologists Press.

Nakov, P., Rosenthal, S., Kozareva, Z., Stoyanov, V., Ritter, A., Wilson, T. (2013). SemEval-2013 task 2: Sentiment analysis in Twitter. Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval. (2013). *Second Joint Conference on Lexical and Computational Semantics (*SEM)* (pp. 312–320). Atlanta, Georgia, USA: Association for Computational Linguistics.

Olson, R.S., Moore, J.H. (2019) Tpot: A tree-based pipeline optimization tool for automating machine learning. In: *Automated Machine Learning* (pp. 151–160). Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12,* 2825–2830.

Pilat, M., Křen, T., Neruda, R. (2016) Asynchronous evolution of data mining workflow schemes by strongly typed genetic programming. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 577–584). IEEE.

Pollak, S., Coesemans, R., Daelemans, W., & Lavrač, N. (2011). Detecting contrast patterns in newspaper articles by combining discourse analysis and text mining. *Pragmatics, Quarterly Publication of the International Pragmatics Association (IPrA)., 21*(4), 647–683.

Qian, M., Zhai, C. (2014) Unsupervised feature selection for multi-view clustering on text-image web news data. In: J. Li, X.S. Wang, M.N. Garofalakis, I. Soboroff, T. Suel, M. Wang (eds.) *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management CIKM* (pp. 1963–1966). Shanghai, China :ACM.

Rappl, G. (1989). On linear convergence of a class of random search algorithms. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 69*(1), 37–45.

Reif, M., Shafait, F., & Dengel, A. (2012). Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning, 87*(3), 357–380.

Rose, S., Engel, D., Cramer, N., & Cowley, W. (2010). *Automatic keyword extraction from individual documents* (pp. 1–20). New Jersey: Wiley Online Library.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence, 1*(5), 206–215.

Sennrich, R., Haddow, B., Birch, A. (2016) Neural machine translation of rare words with subword units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715–1725). Berlin, Germany : Association for Computational Linguistics.

Škrlj, B., Repar, A., Pollak, S. (2019) RaKUn: Rank-based keyword extraction via unsupervised learning and meta vertex aggregation. In: *International Conference on Statistical Language and Speech Processing* (pp. 311–323) Springer.

Snoek, J., Larochelle, H., Adams, R.P. (2012) Practical bayesian optimization of machine learning algorithms. In: P.L. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (eds.) *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012* (pp. 2960–2968), Lake Tahoe, Nevada, United States.

Speer, R., Chin, J., & Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In S. P. Singh & S. Markovitch (Eds.), *Proceeding of the Thirty-First AAAI Conference on Artificial Intelligence* (pp. 4441–4451). San Fransisco, California, USA: AAAI Press.

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence, 1*(1), 24–35.

Sterckx, L., Demeester, T., Deleu, J., Develder, C. (2015) Topical word importance for fast keyphrase extraction. In: *Proceedings of the 24th International Conference on World Wide Web* (pp. 121–122). New York: ACM.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In S. P. Singh & S. Markovitch (Eds.), *Proc of the Thirty-First AAAI Conference on Artificial Intelligence* (pp. 4278–4284). San Francisco, California, USA: AAAI Press.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, et al. (Eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD 2013* (pp. 847–855). Chicago, IL, USA: ACM.

Vafaie, H., & De Jong, K. (1998). Feature space transformation using genetic algorithms. *IEEE Intelligent Systems and their Applications, 13*(2), 57–65. https://doi.org/10.1109/5254.671093.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). Scipy 10 Fundamental algorithms for scientific computing in Python. *Nature Methods, 17*(3), 261–272.

Wan, X., & Xiao, J. (2008). Single document keyphrase extraction using neighborhood knowledge. *Proceedings of the AAAI Conference, 8,* 855–860.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., Rush, A. (2020) Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2020.emnlp-demos.6. https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Yang, C., Akimoto, Y., Kim, D.W., Udell, M. (2019) Oboe. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J.G., Salakhutdinov, R., Le, Q.V. (2019) Xlnet: Generalized autoregressive pretraining for language understanding. In: H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E.B. Fox, R. Garnett (eds.) *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*(pp. 5754–5764) Vancouver, BC, Canada : NeurIPS 2019.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., Kumar, R. (2019) Predicting the type and target of offensive posts in social media. In: textitProceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 1415–1420). Linguistics, Minneapolis, Minnesota : Association for Computational.

Zimmer, M., & Doncieux, S. (2017). Bootstrapping q-learning for robotics from neuro-evolution results. *IEEE Transactions on Cognitive and Developmental Systems, 10*(1), 102–119.

Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V. (2018) Learning transferable architectures for scalable image recognition. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition CVPR 2018* (pp. 8697–8710). Salt Lake City, UT, USA: IEEE Computer Society.

Authors and Affiliations

Blaž Škrlj[1,2] · Matej Martinc[1,2] · Nada Lavrač[1,3] · Senja Pollak[1]

Matej Martinc
matej.martinc@ijs.si

Nada Lavrač
nada.lavrac@ijs.si

Senja Pollak
senja.pollak@ijs.si

[1] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

[2] Jožef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia

[3] University of Nova Gorica, Glavni trg 8, 5271 Vipava, Slovenia

4.2.4 Implementing Grid-scale Neuro-symbolic autoML

The introduced autoML system offers a straightforward exploration of the representation-model space on a single machine. It was designed to use the available (free) threads; this functionality was tested with up to 64 threads, indicating that even single-machine-level parallelism can have a significant impact on the portion of the hyperparameter space explored. As individual autoML jobs (evolutions) were designed to last for a predefined amount of time, we investigated how this type of learning can further scale *to the grid*, i.e., a collection of machines. This section presents our attempt at scaling autoBOT to all available supercomputing resources in as friction-less manner as possible – our goal was to implement a computational framework, which addresses the following main components required for scaling:

- automatic *job management*,
- automatic result *gathering/summarization*,
- intermediary *checkpoints* for a given model,
- cleaning up *redundant files* when needed,
- appropriate *containerization*,
- proxy maintenance and *job scheduling*.

The remaining sections address the mentioned points in more detail. We conclude with an overview of the developed framework.

Framework Overview. We begin the description of the proposed framework with an overview of the key components required for its *continuous operation*. An overview of the main building blocks with corresponding descriptions is shown in Figure 4.2. The key parts of the framework include job generation, retrieval, summarization/selection and checkpoint construction. Even though, conceptually simple, these steps need to be able to adapt to different computing nodes (hardware-wise) and be robust enough that even if whole job batches fail (due to, e.g., proxy misconfiguration), a solution is produced. We continue with a more detailed description of the individual steps next.

Job Construction. The initial part of every search iteration includes the construction of 'jobs', i.e., self-contained instruction sets which, given a data split, return the performance of a given autoML configuration. To make this step unified so that it can be adopted beyond autoBOT, based on which we developed the framework, a unified JSON-based format specifying individual jobs was introduced. The idea of this global configuration file is that it stores the information regarding the autoML configuration (e.g., crossover and mutation rates), and also describes to Singularity-based environments, which enable execution on an arbitrary computing node that has Singularity installed. Jobs thus include the information on where the data is stored, how long should a given evolution last, and what are the expected result files.

Once the jobs are generated in the form of simple bash scripts, they are automatically converted to the *xrsl* language suitable for grid-scale job specification.

Job Execution and Scheduling. The next step involves sending the *xrsl* specifications to the cluster alongside the relevant files and monitoring their execution. To achieve this, the user needs to specify as an argument two main parameters. First, the proxy server, which gets periodically called to update the permission session, and second, a list of computing

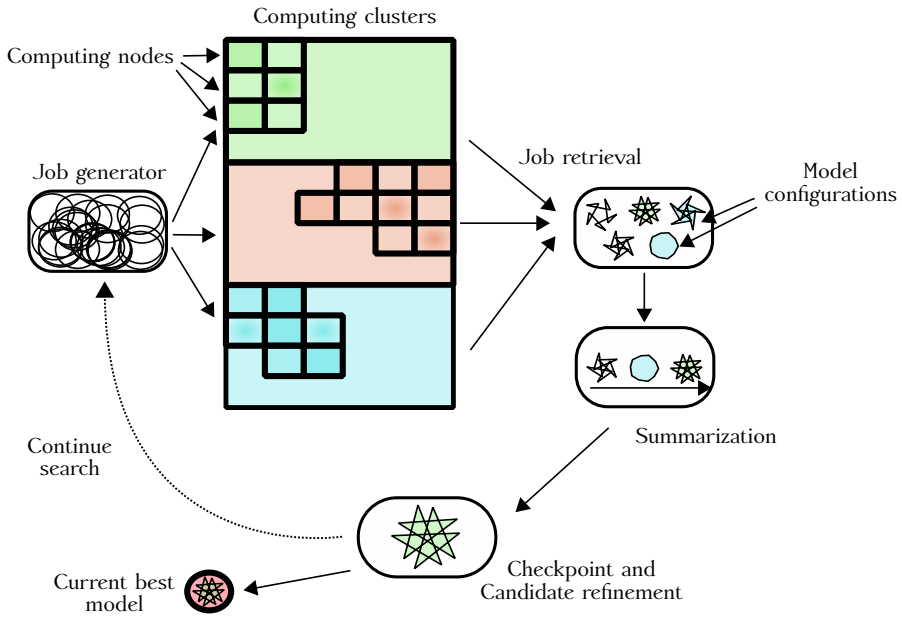


Figure 4.2: An overview of the key components comprising the autoML (grid-scale) framework. Instead of conducting a single evolution run, the proposed framework performs *continuous* evolution. By iterative solution refinement and job generation, it offers seamless scaling to an arbitrary number of computing clusters that support Singularity-based dockerization and InfiniBand-based caching. The different textures of the computing nodes symbolize different hardware properties (not all nodes are the same).

clusters to which a given user can access, and can send *xrsl*-based jobs. The generated jobs are next partitioned into small batches based on how long was the evolution specified to last and how long a particular *xrsl* job is allowed to run. For example, if we conduct evolution for four hours and the available compute time is one day, the framework will automatically generate five subsequent evolution runs (with some time left for file copying).

The jobs are periodically generated and sent to a randomly selected cluster (out of the user-specified list). This step assures a fair distribution of jobs. After jobs are sent, the framework checks for existing jobs, their success/failure and retrieves them.

Job Retrieval and Summarization. The next step includes retrieving the finished jobs and their summarization. As each job submission generates a collection of files that include the exact job IDs, this step first traverses these IDs and checks the jobs' state. If a given job has finished, it retrieves it into a designated folder so that all results are in one place. If the job was retrieved, its IDs are removed from the current collection of 'active' jobs' IDs.

For each resulting job, the framework traverses its result JSON-based report and stores the result into a simple TSV table that is easy to inspect. This way, the framework, running continuously, constantly provides the user with the current best result. As the framework also maintains the global results table, it is able, with each update, to re-rank the models/identify possible better models. Should a better model be identified, the current best model (stored separately) is replaced. Furthermore, as all other models are automatically *removed*, the space overhead of this retrieval/ranking step is very low – the main node responsible for sending out and retrieving jobs can be an off-the-shelf laptop if needed.

Automating model containerization. This section addresses the problem of *model deployment*. By being able to obtain/select models at the grid-scale automatically, we demonstrated that finding good classifiers can be, to some extent, automated. A model, however, remains useless if it is not deployed/utilized in a realistic setting. Even though the produced models come in the form of compressed *pickle* objects that are easy to share/distribute, their use is limited to the set of people able to load/parse such objects. The additional step that potentially further facilitates model development/deployment is automatic dockerization of models alongside simple Application Programming Interfaces (APIs). We aimed at developing a simple-to-use dockerization wrapper, which takes a given pickle model and outputs the Docker image, which, when run, serves the models in a simple-to-use API. We next discuss this software's architecture, followed by an example.

Server Architecture and Build Procedures. This section provides an overview of the considered server architecture and its link to the automatic model construction framework discussed in the previous sections. The key goal of the presented dockerization regime is to be *very simple* to use and offers adequate, out-of-the-box performance. The container construction can be split into three main steps discussed next.

Model verification. The first step of building a model container performs a collection of tests which assess its *validity*, i.e., suitability for being hosted. The tests include model loading/decompression and example predictions.

Container specification. Even though the current solution already includes a generic container specification capable of handling pickled models with a *scikit-learn*-like class structure, the model object should include all required weights/background knowledge. Further it should include the standard *fit* and *transform* methods if required. Furthermore, the core API request handler can be modified to the user's preference, should, e.g., a non-pickle-based object be required (e.g., model binaries directly).

Container building. Should the specification be correct, the container can be built and used for a downstream application. The container can be subsequently loaded in a monitoring system like *podman* (to ensure its uptime) and used in further design/development of a given application. The build procedure will automatically install the project's requirements (requirements.txt) alongside an uvicorn-based API[1] (SimpleAPI), which serves the pre-defined app at some generic port (which can be forwarded when needed during hosting). The full project is available freely at https://gitlab.com/skblaz/autobot-api/.

Table 4.1: Overview of different sparse matrix data structures.

Representation Identifier	Description	Advantages	Disadvantages
CSC	Compressed Sparse Column	Col. slicing, arithmetic, matrix-vector products	Row slicing, structure changes
BSR	Block Sparse Row	Block-containing matrices	Non-block-like matrices
COO	Coordinate format	Fast conversions to other formats	Slicing and arithmetic
CSR	Compressed Sparse Row	Arithmetic, row slicing, matrix-vector products	Col. slicing, structure changes
DIA	Diagonal storage	Arithmetic	Transformations, non-diagonal structure
DOK	Dictionary of Keys	Arithmetic, $O(1)$ access	Duplicate handling, slicing
LIL	List of lists	Slicing, structure changes	Arithm., matrix-vector products, col. slicing

A note on sparsity. Sparse representations can be represented in two main ways: as dense matrices or as sparse data structures corresponding to the sparse matrix itself. Should

[1] https://www.uvicorn.org/

the amount of memory be sufficient to represent a given sparse representation with a dense matrix, this is the preferred option for the following reasons. First, all subsequent operations do not require investigation of which sparse matrix types are the most suitable, simplifying the final implementation. And second, dense matrices can be more natively ported to specialized hardware such as GPUs, offering faster execution. On the contrary, if a sparse representation, when represented with a dense matrix, does not fit into memory, a multitude of data structures capable of holding the sparse structure directly can be considered (and, in fact, were considered). Examples include the CSR, COO, BSR, CSC, DIA, DOK and LIL formats[2]. A summary of different sparse matrix formats and their intended use is given in Table 4.1.

The common denominator to all formats is that they are more suitable for some operations and less for others. For example, CSR matrices are suitable when performing dot product-based operations, but not indexing.

[2]See more extensive descriptions at https://docs.scipy.org/doc/scipy/reference/sparse.html

Chapter 5

Neuro-symbolic Learning from Tabular Data

> A chain is only as strong as its weakest link.
>
> Thomas Reid (1786)

Many real-life data sets are in tabular format. Examples include biomedical, genomics, demographic and sensor-based data. In this chapter, we discuss the contribution, which addresses the question of how the input feature space can be linked with a part of a neural network and whether this connection resembles feature ranking.

5.1 Learning from Data Tables

Tabular data remains one of the most widely studied data formats, and is commonly considered in biology, medicine, finance, engineering, and physics to describe different problems. A schematic overview of the core learning task related to this chapter is shown in Figure 5.1. The input space (green) is a real-valued matrix; each column represents a feature. The target space (yellow) can similarly be a real-valued matrix, with each column representing a target variable. In this work, targets are discrete – we are considering classification. The goal of learning is to find f, which, given the input space, correctly outputs the target values. The figure shows a high-level statement of the problem; we are interested in automatically obtaining a function capable of associating inputs to outputs. However, an essential aspect of many tabular methods is understanding which parts of the input are crucial for producing a given output. This problem is addressed in the following section.

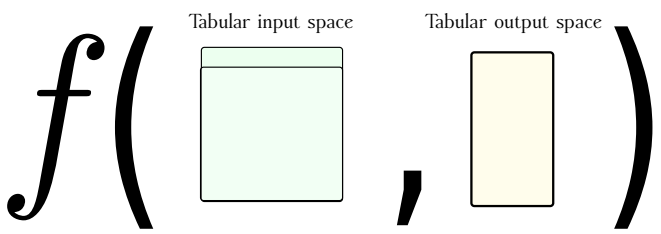


Figure 5.1: Overview of learning from tabular data.

5.2 Evaluating Neural Attention as Feature Ranking

As discussed in Chapter 2, neural networks have overcome the conventional approaches in the domains of image recognition and text-based learning. Learning from propositional data, however, remains an interesting open problem. This section presents one of the first attempts at designing neural network architectures capable of maintaining the link between individual attributes and a particular subset of weights. The purpose of this paper was to demonstrate that the weights associated with this link can be interpreted as *feature ranking*, i.e. an ordered set reflecting the importance of individual features. As feature rankings can be evaluated, we conducted an extensive empirical evaluation to demonstrate that the feature self-attention, as we term it, can offer ranking-like properties. The contribution was strongly inspired by the attention mechanism (Devlin et al., 2019), which was the key architectural breakthrough that enabled neural network-based models to proliferate in the domain of text mining. The paper is structured first to present the notions of feature ranking and the relevant neural network concepts, followed by the formulation of the idea, its implementation, theoretical analysis, empirical results and the implications of being able to maintain the link to the feature space. The paper relevant to this section is the following one:

> Škrlj, B., Džeroski, S., Lavrač, N., & Petkovič, M. (2020). Feature Importance Estimation with Self-Attention Networks. In G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, & J. Lang (Eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)* (pp. 1491–1498). IOS Press. https://doi.org/10.3233/FAIA200256

5.2.1 Key Contributions

We next present the key contributions of the conducted work.

1. We proposed SAN (Self-Attention Networks), a neural network architecture inspired the recent developments in attention-based neural network design applicable to propositional data.

2. We identified a link between the self-attention vectors (diagonals of attention matrices) and the notion of *feature ranking*.

3. To evaluate the hypothesis that self-attention resembles feature ranking, we conducted an extensive empirical evaluation against strong baselines such as the Relief branch of algorithms.

4. We offered a theoretical analysis of the algorithm's computational complexity, offering insights into when and why SAN potentially works and when it might encounter problems.

5. We proposed multiple types of aggregations, which could lead to self-attention vectors (rankings).

5.2.2 Addressed Hypotheses and Discussion

We next discuss the key research questions addressed in this work. The questions by no means reflect all the work considered, but, summarise the main, at the time, unknown

relations explored. We are interested in the following association. A recent branch of neural network-based methods relies heavily on the notion of the attention mechanism – a specialized layer that maintains the input-weight correspondences. We hypothesize that self-attention vectors (diagonals of the attention matrix) offer similar rankings of features as can be obtained via conventional feature ranking approaches such as, e.g., Relief. By demonstrating a persistent link between a part of a neural network architecture and a corresponding part of the instance space, a plethora of possibilities for further inspection of key patterns relevant to a neural network could emerge.

We next discuss how the presented work addresses the raised hypotheses and potential implications. We remind the reader that the main hypothesis related to this chapter was whether the self-attention mechanism, if adopted for use in a propositional (tabular) learning setting, yields feature scoring that could be interpreted as *feature ranking*. This hypothesis addresses both the *understanding* and *explainability* aspects (Hypotheses 1 and 2, Section 1.3), addressed more broadly as part of this thesis. First, by quantitatively evaluating to what extent can attention-based ranking resemble conventional rankings, this work offers a better understanding of the nature of the attention mechanism for propositional data and its relation to feature importances (Hypothesis 1). Second, by demonstrating that attention vectors can indeed represent rankings in a given learning scenario, the explainability increases (Hypothesis 2).

To verify these hypotheses, we conducted extensive experiments, demonstrating that the rankings obtained via the proposed neural approach offer competitive performance when evaluated with an external classifier – in the conducted experiments, this was the *myopic* logistic regression. Further, we have for the first time studied the associations between the neural rankings and conventional ones via the adoption of the Fuzzy Jaccard Index (Petković et al., 2021), a recently proposed score that generalizes the commonly used Jaccard index. This part of the study corresponds to the first hypothesis concerning *better understanding* of a given black-box procedure (self-attention ranking in this case). We demonstrated that self-attention rankings are more similar to the Relief-branch of algorithms and less similar to myopic ones such as mutual information-based rankings. In terms of explainability (Hypothesis 2), current results support the following conclusions. One of the key goals of the paper was to evaluate whether the self-attention mechanism indeed offers a potential link to the concept of *feature ranking* as known in the wider data mining/machine learning community. Current results indicate that as long as the full neural network converges to a well-performing model, the attention vectors offer feature rankings compared to the existing ones. This result indicates that further study of how the neural attention mechanism offers better insight into what the model emphasizes both at the instance and at the global level is a promising research endeavour. This claim is partially supported by related studies, such as the TabNet (Arik, Sercan Ö. and Pfister, Tomas, 2021) paper, which were submitted for publication at around the same time. Finally, we demonstrated that the proposed architecture scales to real-life data sets with tens of thousands of features; however, we acknowledge its expensive space overhead ($\mathcal{O}(|F|)^2$, which could be further optimized.

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Feature Importance Estimation with Self-Attention Networks

Blaž Škrlj[1] and **Sašo Džeroski**[1] and **Nada Lavrač**[1] and **Matej Petković**[1]

Abstract. Black-box neural network models are widely used in industry and science, yet are hard to understand and interpret. Recently, the attention mechanism was introduced, offering insights into the inner workings of neural language models. This paper explores the use of attention-based neural networks mechanism for estimating feature importance, as means for explaining the models learned from propositional (tabular) data. Feature importance estimates, assessed by the proposed Self-Attention Network (SAN) architecture, are compared with the established ReliefF, Mutual Information and Random Forest-based estimates, which are widely used in practice for model interpretation. For the first time we conduct scale-free comparisons of feature importance estimates across algorithms on ten real and synthetic data sets to study the similarities and differences of the resulting feature importance estimates, showing that SANs identify similar high-ranked features as the other methods. We demonstrate that SANs identify feature interactions which in some cases yield better predictive performance than the baselines, suggesting that attention extends beyond interactions of just a few key features and detects larger feature subsets relevant for the considered learning task.

1 Introduction

Deep neural networks have been successfully applied to text, graph and image-based classification tasks, as well as to learning accurate classifiers from propositional (tabular) data [10, 13, 20]. However, with increasing number of parameters, neural networks are becoming less human-understandable. The currently adopted paradigm to tackle this problem involves using *post hoc* explanation tools, such as SHAP and LIME [23]. Such approaches operate by approximating the outputs of a given black-box model via some tractable scheme, such as efficient computation of Shapley values or local approximation via a linear model. Such methods do not take into account the inner structure of a given (deep) neural network, and treat it as a black box, only considering its inputs and outputs.

Recent advances in the area of language processing, however, offer the opportunity to link parts of a given neural network with the input space directly, via the *attention mechanism* [33]. Achieving super-human performance on many language understanding tasks, attention-based models are becoming widely adopted throughout scientific and industrial environments [34]. Models, such as BERT [11] and XLNet [37], exploit this *learnable lookup* to capture relations between words (or tokens). The attention layers, when inspected, can be seen to map real values to parts of the human-understandable input space (e.g., sentences). Exploration of the potential of the attention mechanism is becoming a lively research area on its own [7, 21]. We believe similar ideas can be investigated in the context of propositional (tabular) data (where every row represents an individual data instance) that remain one of the most widely used data formats in academia and industry.

In this work, we propose the concept of *Self-Attention Networks* (SANs) and explore, whether the representations they learn can be used for feature importance assessment, comparable to feature importance estimates returned by the established feature ranking approaches ReliefF [27], Mutual Information and Genie3 [16, 24]. The main contributions of this work are:

1. The Self-Attention Network (SAN) architecture, a novel neural network architecture capable of assessing feature importance directly, along with three ways of obtaining such importance estimates.
2. Extensive empirical evaluation of SANs against ReliefF, Mutual Information and Genie3 feature ranking approaches, demonstrating comparable performance to the SAN architecture. The most important features according to SAN are shown to be in agreement with the ones detected by the established approaches used in the comparison.
3. Direct comparison of feature importance estimates, highlighting similarities between the considered algorithms' outputs.
4. A theoretical study of SAN's properties, considering its space and time complexity.

In the remainder of this paper we first discuss the related work, followed by the formulation and empirical evaluation of SANs.

2 Related work

In this section we present selected feature ranking approaches and briefly survey the approaches to learning from propositional data, followed by a presentation of neural attention mechanisms.

2.1 Feature ranking

Feature importance estimation refers to the process of discovering parts of the input feature space, relevant for a given predictive modeling problem, i.e. identifying the (most) important features to be used in model construction.

The simplest task of estimating feature importance is partitioning the features into groups of irrelevant and relevant ones, which is equivalent to assigning every feature either importance 1 (feature is relevant) or 0 (feature is irrelevant). This task is referred to as *feature selection*. Here, one can only partially order the features. In a more general case, when every feature is assigned an arbitrary importance score, one can sort the features with respect to these scores

[1] Jožef Stefan Institute and International Postgraduate School Jožef Stefan, Slovenija, matej.petkovic@ijs.si

and obtain *feature ranking*. In this case, the features can be totally ordered. Given a feature ranking, one can partition the features into relevant and irrelevant ones by thresholding, i.e., proclaiming relevant the features whose importance is higher than some threshold.

The main motivation for performing feature ranking (or selection) is that an appropriately chosen subset of features may reduce the computational complexity of learning, while potentially increasing the learner's performance. Moreover, the learned models that use a smaller number of features are easier to explain.

In this work we consider a feature space F with a corresponding class label set C. We focus on feature ranking algorithms, but use thresholding (for many different values of the threshold) at the ranking-evaluation stage, i.e., we measure the quality of the ranking as the predictive performance of the models that are built from sets of top-ranked features.

In this work we consider two types of feature ranking algorithms:

Wrappers. A feature ranking algorithm of this family comes together with a supervised learner. Once trained on the training data, the learner is, apart from prediction, also capable of estimating how relevant is each of the features for the task at hand. An important member of this family is Genie3 feature ranking, computed directly from Random Forest [4].

Filters. Algorithms such as the Relief family or Mutual Information do not need a predictive model to obtain feature rankings. Since they are model-agnostic, the corresponding feature rankings are typically computed very efficiently, e.g., Mutual Information. However, the computational efficiency often comes at the cost of myopia, i.e., they ignore possible feature interactions.

Recent advances in deep learning mostly address model-based ranking, where methodology, such as e.g., SHAP is used for *post-hoc* analysis of a trained model. Note that such a methodology can also be model-agnostic, yet *it needs* a model to compute feature importance.

2.2 Propositional learning and neural networks

In this work we focus on learning from (propositional) data tables, where columns represent features (attributes) and rows correspond to individual data instances. Despite its simplicity, training neural network architectures on this type of data might be non-trivial in the cases of small data sets, noisy feature spaces, spurious correlations, etc. Further, recurrent or convolutional neural architectures are not that useful for tabular data as there is frequently no spatial or temporal information present in the data. Methods, such as Extreme Gradient Boosting [5] and similar, e.g., tree-based ensemble algorithms often dominate competitive shared tasks on such input spaces.

Neural network approaches recently used in modeling propositional data are discussed next. The approach by Sakakibara [28] addresses learning of context-free grammars based on tabular (propositional) representations. Further, multilayer perceptrons have been widely used, already in the previous century [18]. The recently introduced Regularization Learning Networks (RLNs) represent one of the most recent attempts at training deep neural networks on propositional data sets [31]; the paper shows that correctly regularized neural networks perform on par with e.g., gradient boosting machines, demonstrating competitive performance on multiple data sets. Further, the authors also explored the information content of the feature space, showing that RLNs detect highly informative features. The attention-based propositional neural networks were also considered in the recently introduced TabNet [2]. Finally, ensemble-based learners were also successfully applied to propositional data, including time series modeling [26].

2.3 Attention-based neural network architectures

The area of deep learning has witnessed many novel ideas in recent years. The notion of *attention* has emerged recently for language model learning [11, 33], as well as geometric deep learning [36]. In short, the attention mechanism enables a neural architecture to pinpoint parts of the feature space that are especially relevant, and filter out the remainder. The attention mechanism effectively sparsifies the input space, emphasizing only the elements relevant for the task at hand (e.g., language understanding). This way, the attention mechanism is commonly used to e.g., better learn the relations between words. Even though this mechanism has been widely adopted to text-based and graph-based tasks, less attention has been devoted to its application to propositional learning from tabular data.

3 Self-Attention Networks

This section presents the proposed Self-Attention Network (SAN) approach, illustrated in Figure 1. We begin by a formal description of the architecture, followed by feature importance computation.

3.1 Propositional Self-Attention Networks

This section sources some of the ideas from the seminal works on the attention mechanism [6, 33]. We refer the interested reader to the aforementioned publications for a detailed description and explain here only the essential ideas implemented as part of SANs in a propositional learning setting. The neural network architecture that implements the attention mechanism can be stated as:

$$l_2 = \sigma(W_2 \cdot (a(W_{|F|} \cdot \Omega(X) + \boldsymbol{b}_{l_1})) + \boldsymbol{b}_{l_2}).$$

The first neural network layer Ω is designed specifically to maintain the connection with the input features F. We define it as:

$$\Omega(X) = \frac{1}{k} \bigoplus_{k} \left[X \otimes \text{softmax}(W^k_{l_{\text{att}}} X + \boldsymbol{b}^k_{l_{\text{att}}}) \right].$$

Input vectors X are first used as input to a softmax-activated layer containing the number of neurons equal to the number of features $|F|$, where the softmax function applied to the j_i-th element of a weight vector v is defined as:

$$\text{softmax}(\boldsymbol{v}_{j_i}) = \frac{\exp(\boldsymbol{v}_{j_i})}{\sum_{j=1}^{|F|} \exp(\boldsymbol{v}_j)},$$

where $v \in \mathbb{R}^{|F|}$. Note that k represents the number of *attention heads*—distinct matrices representing *relations* between the input features. The $\otimes$ sign corresponds to the Hadamard product and the $\oplus$ refers to the Hadamard summation across individual heads. Ω thus represents the first layer of a SAN, its output is of dimensionality $|F|$. a corresponds to the activation function SELU [19], defined as:

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}.$$

where λ and α are hyperparameters. The proposed architecture enables *self-attention* at the feature level, intuitively understood as follows. As Ω maintains a *bijection* between the set of features F and

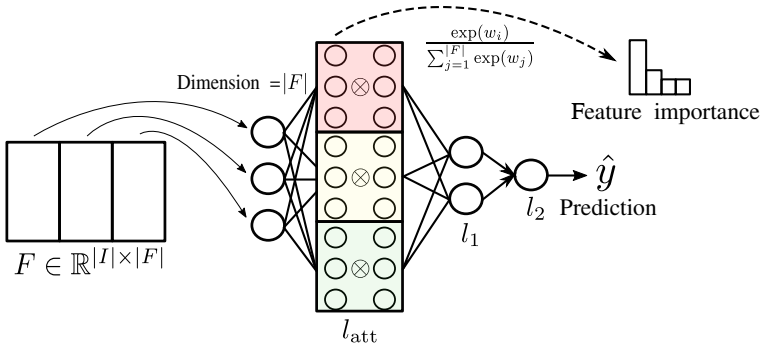


Figure 1: Overview of Self-Attention Networks. l_{att} corresponds to the attention layer, where the element-wise product with the input space is computed on each forward pass. l_1 and l_2 correspond to two dense layers, yielding a prediction $\hat{y}$. The $\frac{1}{k}\oplus$ box represents element-wise means across the attention heads (k). Note that, if extracted, the attention vectors from l_{att} can be used to compute the *feature importance* estimates.

the set of weights in individual heads $W^k_{l_{att}}$, the weights in the $|F| \times |F|$ weight matrix can be understood as *relations between features*. Finally, SANs are additionally regularized using Dropout [32][2]. In this work, we are interested exclusively in self-attention—the relation of a given feature *with itself*. We posit that such self-relations correspond to features' importance. Once a SAN is trained, we next discuss the ways of computing feature importance.

3.2 Computing feature importance with SANs

Let us show how the considered architecture can be used to obtain real values which we argue represent feature importance. We explore the procedures for obtaining the final vector, as a mapping from the feature space to the space of non-negative real values, i.e. function $f : F \to \mathbb{R}^+_0$.

Instance-level aggregations (attention). Let $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $1 \leq i \leq n$, be the instances. Let $\text{SAN}(\boldsymbol{x}_i)$ represent the attention output of the i-th instance, defined as

$$\text{SAN}(\boldsymbol{x}_i) = \frac{1}{k}\bigoplus_k \left[\text{softmax}(W^k_{l_{\text{att}}}\boldsymbol{x}_i + \boldsymbol{b}_{l_{\text{att}}})\right].$$

The first option for computing feature importance performs the following operation, where the outputs are *averaged* on-instance basis.

$$R_I = \frac{1}{n}\sum_{i=1}^{n}\text{SAN}(\boldsymbol{x}_i).$$

Counting only correctly predicted instances (attentionPositive). The second variant of the mechanism operates under the hypothesis, that only correctly predicted instances should be taken into account, as they offer extraction of representative attention vectors with less noise. Such scenarios are suitable when the number of instances based on which features' importance are to be computed is low, and the classification performance is not optimal. Let $\hat{\boldsymbol{y}}_i$ represent the final prediction of an architecture (not only attention vectors). This version of the approach assesses feature importance R^c_I (c stands for clean) as follows:

$$R^c_I = \frac{1}{n}\sum_{i=1}^{n}\text{SAN}(\boldsymbol{x}_i)\left[\hat{\boldsymbol{y}}_i = \boldsymbol{y}_i\right].$$

Global attention layer (attentionGlobal). The previous two approaches construct the global feature importance vector incrementally, by aggregating the attention vectors based on individual instances used for training. However, such schemes can depend on the aggregation scheme used. The proposed global attention approach is more natural, as it omits the aggregation: once trained, we simply activate the attention layer's weights by using softmax. This scenario assumes, that the weight vector itself contains the information on feature importance, and can be inspected **directly**[3]. Global attention is defined as follows:

$$R_G = \frac{1}{k}\bigoplus_k \left[\text{softmax}(diag(W^k_{l_{\text{att}}}))\right]; W^k_{l_{\text{att}}} \in \mathbb{R}^{|F|\times|F|}.$$

4 Theoretical considerations

In this section we discuss the relevant theoretical aspects of the considered neural network architecture, starting with SAN's space and time complexity, and followed by an overview of the computational complexity of the considered methods.

4.1 Space and time complexity of SANs

We first discuss the space complexity with respect to the number of parameters, as we believe this determines the usefulness of SANs in practice. Assuming the number of features to be $|F|$, the most computationally expensive part of SANs is the computation of the attention vector. The attention, as formulated in this work, operates in the space of the same dimensionality as the input space; the number of parameters is $\mathcal{O}(|F|^2)$. This complexity holds if a single attention layer is considered. As theoretically, there can be multiple such mappings to the input space in a single *attention block*, the space complexity in such case rises to $\mathcal{O}(|F|^2 \cdot k)$, where k is the number of considered attention weight matrices (heads). In practice, however, even very high dimensional data sets with, e.g., more than $60{,}000$ features can be processed, which we prove in the empirical section. Note that $(6\cdot 10^4)^2 = 3.6\cdot 10^9$ ($\approx$ 11GB if stored as floating points), which is the scale at which language models such as RoBERTa [22] and similar operate, thus the complexity does not prohibit the use on high dimensional data that of practical relevance. We believe, however, that sparsity at the input level could significantly reduce the complexity, which we leave for further exploration.

[2] For readability purposes, we omit the definition of this well known regularization method.

[3] In Appendix A we discuss another way how global attention can be obtained, yet do not consider it in this work.

We finally discuss the time complexity of computing feature importance for a single instance, as well as for a set of instances. In Section 3.2 we introduced three different implementations of SANs considered in this work: two instance-based, where attention is computed for each instance and aggregated afterwards, as well as global, where after training, the attention vector is simply extracted and used to represent features' importances. The first situation, where predictions up to the end of the second layer need to be conducted (up to the activated attention vector), is of linear complexity with respect to the number of test instances, across which the final importances shall be aggregated. On the other hand, if the attention vector is activated directly after the training phase, the computational complexity reduces to the computation of the softmax, and is in fact $\mathcal{O}(1)$ w.r.t. the number of test instances. Note that such computation is fundamentally faster and scales to massive production environments seamlessly.

4.2 Overview of computational complexity

To contextualize the computational complexity of SANs, we compare it with some of the established feature ranking algorithms that will offer the reader insights into possible benefits, as well as drawbacks of individual methods considered in this work.

In Tables 1 and 2, t denotes the number of trees in the Random Forest, while k denotes the number of attention matrices (heads) used.

Table 1: Computational complexity of feature importance estimation: The learning stage.

Algorithm	time complexity	space complexity
ReliefF	$\mathcal{O}(\|F\| \cdot \|I\|^2)$	$\mathcal{O}(\|F\|)$
Random Forest	$\mathcal{O}(t\|F\|\|I\| \log^2 \|I\|)$	$\mathcal{O}(\|I\| + \|F\|)$
Mutual Information	$\mathcal{O}(\|F\| \cdot \|I\|)$	$\mathcal{O}(\|I\|)$
SAN	$\mathcal{O}(\|F\|^2 \cdot \|I\|)$	$\mathcal{O}(k \cdot \|F\|^2)$

Table 2: Computational complexity of feature importance estimation: The post-learning stage. NA (not applicable) denotes the cases where feature importance is obtained directly from the learning stage.

Algorithm	time complexity	space complexity
ReliefF	NA	NA
Random Forest	$\mathcal{O}(t\|I\|)$	$\mathcal{O}(\|I\|)$
Mutual Information	NA	NA
SAN	$\mathcal{O}(1)$ or $\mathcal{O}(\|I\|)$	$\mathcal{O}(k \cdot \|F\|^2)$

In Table 1, we compare the computational complexity of the learning process: this refers to either building a predictive model (Random Forest, SAN) or computing the feature importances from the training data (ReliefF, Mutual Information). Further, in Table 2, we discuss the complexity of obtaining feature importances after the learning stage. Note that filter methods, such as the ReliefF and Mutual Information only require the first step of the computation, whereas SAN and Random Forest require an additional computational step that extracts feature importances based on the trained model.

5 Experimantal setting

In this section we discuss the empirical evaluation setting used in this work. The evaluation is split into two parts: measuring the feature ranking similarity and feature ranking quality. When measuring feature ranking similarity, we compare pairs of the rankings via FUJI score. When measuring feature ranking quality, we explore how a series of classifiers behave, when top n features are considered for the task of classification.

5.1 Pairwise comparisons with FUJI

The output of a feature ranking can be defined as a real-valued list, whose j-th element is the estimated feature importance of the j-th feature. A typical approach for comparing such lists is to compute the Jaccard indices [17] between the sets of top-ranked features. However, this score takes into account feature importances only implicitly (via the order of the features) and is consequently unstable and often too pessimistic. Thus, we use its fuzzy version FUJI (the Fuzzy Jaccard Index) [25].

Even though FUJI takes importances directly into account, it is still scale-free and thus appropriate for comparing the outputs of different feature ranking algorithms. Given a pair of feature rankings, e.g., Mutual Information and Attention, FUJI is computed for different sizes of top-ranked feature sets that belong to the two rankings. The obtained values are then plotted as a single curve, as shown in Figure 2. Such a comparison is very informative, since one can observe how the similarity of the two rankings changes when the number of features considered grows, but aggregation of a curve into a single scalar may sometimes be preferred. We do so by computing the area under the FUJI curve. Either trapezoidal or Simpson's rule can be employed; in this work we use the latter[4].

In the experiments, we obtain feature rankings for similarity comparison as follows. The model-based ranking algorithms (SANs and Random Forests) were trained by first selecting optimal model based on ten fold stratified cross validation. This step is skipped in the case of ReliefF and Mutual Information algorithms. In the second step, we use the whole data set to estimate feature importances (either directly or using the chosen model). The whole data set can be used since these feature rankings are only used in the similarity computation (and not for predicting the target values), and should be used since the feature rankings are expected to be better when computed from more data.

5.2 Classification performance evaluation

In the second set of the experiments, we investigate how the top n features assessed by a given method influence the performance of the Logistic Regression (LR) classifier, for different values of n. We selected this learner for the following reasons. As evaluation of individual values of n requires hundreds (or thousands) of models to be built, LR is fast enough to be used in such setting. Next, it is sensitive to completely non-informative features. Should a given feature importance assessment method prioritize completely irrelevant features, the classifier will not perform well. In terms of hyperparameters, we use the same parametrization of SANs across all data sets. The rationale for this decision is that neural networks can easily overfit a given data set. We decided to use the same set of hyperparameters to showcase the overall adequate performance of SANs. The l_1 dimension was set to 128, the network was trained for 32 epochs, with the batch size of 5. The learning rate for the Adam optimizer was set to 0.001, and the dropout, which we used for regularization, was set to 20%. We used stratified 10 fold cross validation where a feature ranking is computed from the training set and evaluated on the testing set. A single attention head was used ($k = 1$). We report the average Logistic Regression performance for each n.

[4] This implicitly assumes the FUJI curve is a spline of quadratic polynomials. We average the normalized areas across all data sets to obtain a single scalar representing the relation between two methods.

5.3 Experimental data sets

We conducted the experiments on the following data sets.

DLBCL [3]. A series of translocations specify a distinct gene expression profile that distinguishes a unique leukemia. The data set consists of 7,070 features and 77 instances.
Genes [35]. The cancer genome atlas pan-cancer analysis project. The data set consists of 20,531 features and 801 instances. Features correspond to gene expression vectors.
p53 [9]. Predicting Positive p53 Cancer Rescue Regions Using Most Informative Positive (MIP) Active Learning. The data set consists of 5,409 features and 16,772 instances.
Chess [30] (King-Rook vs. King-Pawn). The data set describes various endgames, and consists of 3,196 instances and 36 features.
pd-speech [29]. Collection and Analysis of a Parkinson Speech data set with Multiple Types of Sound Recordings. The data set consists of 26 features and 1,040 instances.
aps-failure [8]. IDA 2016 Industrial Challenge: Using Machine Learning for Predicting Failures. The data set consists of 171 features and 60,000 instances.
biodeg [12]. Biodegradability of commercial compounds. the data set consists of 62 features and 328 instances.
optdigits [1]. Optical Recognition of Handwritten Digits. The data set consists of 64 features and 5,620 instances.
madelon [15]. This is a synthetic data set published at NIPS 2003, aimed to test feature selection capabilities of various algorithms. The data set consists of 500 features and 4,400 instances.

The selected data sets span across multiple topics, and were selected to provide insights into the quality of feature importance assessment methods. Finally, we also evaluated the difference in the attention across relevant and irrelevant features based on the algorithm proposed by Guyon et al. [14][5]. For the purpose of this synthetic experiment, we generated a binary classification problem data set comprised of 100 features, where only 50 were deemed relevant, with 1,000 samples of the data used for training SANs. The experiment was conducted via three repetitions of three-fold cross validation. We aggregated the attention values separately for the positive, as well as the negative class if the classification accuracy for a given fold was more than 50%. The attentions were aggregated only if a given prediction was correct—this step was considered to reduce the effect of the neural network's performance on the attention vectors' properties, as we only attempted to extract the attention relevant for the discrimination between the classes.

6 Results of experiments

In this section, we present the results of the empirical evaluation. For improved readability, the results are presented graphically.

6.1 Differences in attention vectors

We first visualize the distributions of attention with respect to both positive and negative instances in Figure 2.

It can be observed that the attention is on average observably higher when positive instances are considered. As the neural network is trained to recognize such examples, such outcome indicates the attention mechanism potentially detects the relevant signal. Note that

[5] Implemented in `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html`

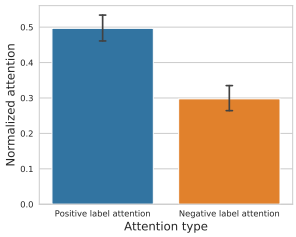


Figure 2: Differences in attention, aggregated w.r.t. correctly predicted positive and negative examples.

Figure 2 shows attention, aggregated only on the features that were assessed as important (half of the features).

6.2 Importance similarities

We next discuss the similarities between the considered feature importance approaches. The results are presented as FUJI curves, where we omit the space of all possible (pairwise) comparisons to the ones, that are compared with the proposed SANs. The results are shown in Figure 3.

The considered visualizations depict at least two general patterns. First, the attentionClean and attention-based importances are indeed very similar. We remind the reader that the difference between the two is that attentionClean only takes into account the correctly classified instances. Next, we can observe that ReliefF and Random Forest-based importances are different. Further, the global attention appears the most similar to Random Forest importances. Overall areas below FUJI curves are shown in Figure 4.

6.3 Classification performance

We next discuss the classification performance results. Let us remind the reader that the results show how the logistic regression (C = 1, L2 normalization) classifier performs when parts of the feature space are pruned according to a given feature importance vector. The results are shown in Figure 5.

The first observation—to our knowledge not known to the community—is that SANs indeed perform *competitively*. Especially on data sets with a large number of features, such as the p53, genes and dlbcl (all biological ones), the attention mechanism detects larger subsets of features that can improve the performance of the logistic regression classifier, in e.g., dlbcl even outperforming all other approaches by a margin of more than 15%. On smaller data sets, SANs perform competitively (e.g., madelon, optdigits, chess), or do not perform that well (biodeg-p2-discrete). Overall, worse performance of SANs coincides with smaller data sets, indicating that not enough data was present for SANs to distill the relevant parts of the feature space. Further, it can be observed that ReliefF on data sets such as pd-speech and madelon outperforms all other approaches, indicating this state-of-the-art approach is suitable for low-resource situations.

7 Discussion and conclusions

Operating on propositional data, SANs have some inherent benefits, as well as drawbacks. The most apparent drawback is the architecture's space complexity, as with more attention heads, it could increase greatly for the data sets with a large number of features. In comparison, language models are maintaining a fixed sequence

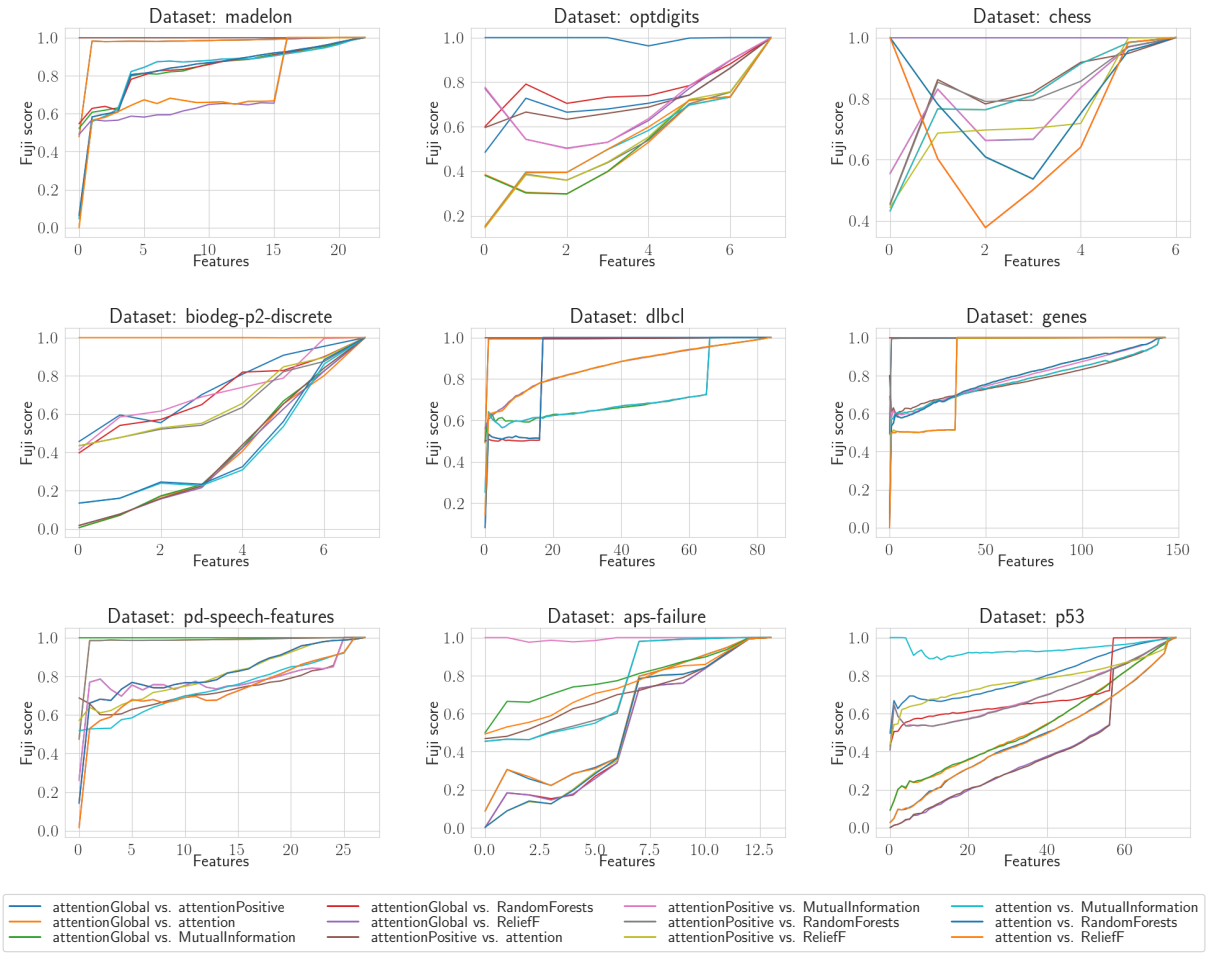


Figure 3: Results of FUJI-based ranking comparison. Higher values imply the pair of algorithms is more similar when a certain number of features (x-axis) is considered.

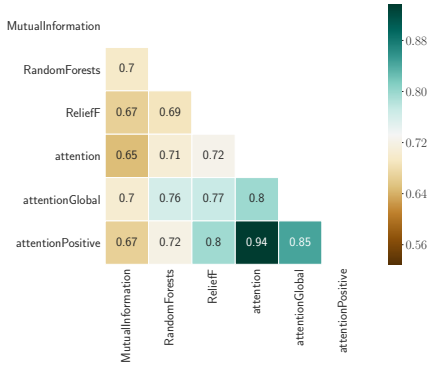


Figure 4: Similarities of considered feature importance assessment approaches. The attention-based importances are the most similar to each other, as well as to the ReliefF algorithm.

length during training, which is commonly below e.g., 1,024, and can thus afford multiple attention heads more easily.

Another key difference is the mapping to the input space. As with text, each input instance consists of potentially different tokens, attention matrices represent different parts of the vocabulary for each e.g., sentence. In contrast, the proposed SANs exhibit a consistent mapping between the input space and the meaning of individual features, making possible the aggregation schemes considered in this work.

In terms of general performance, this paper proves empirically that SANs can emit attention vectors that offer similar feature rankings relevant for an external classifier (e.g., Logistic Regression). As the purpose of this work was not excessive grid search across possible architectures, which we leave for further work. We believe SANs's performance could be notably improved by improving the underlying neural network's performance, as well as by using other types and variations of the attention mechanism. Further, the proposed SANs shall be compared to the recently introduced TabNet [2], an approach that also attempts to unveil the feature importance landscape via sequential attention mechanism.

An additional contribution of this work is the FUJI score based comparison, where we uncover similarities between *all* considered approaches. The SANs are similar to the Genie3 and ReliefF, indicating the importance computation is possibly non-myopic (as MI), yet we leave validation of this claim for further work.

8 Availability

The code to reproduce the results is freely accessible for academic users at: `https://gitlab.com/skblaz/attentionrank`.

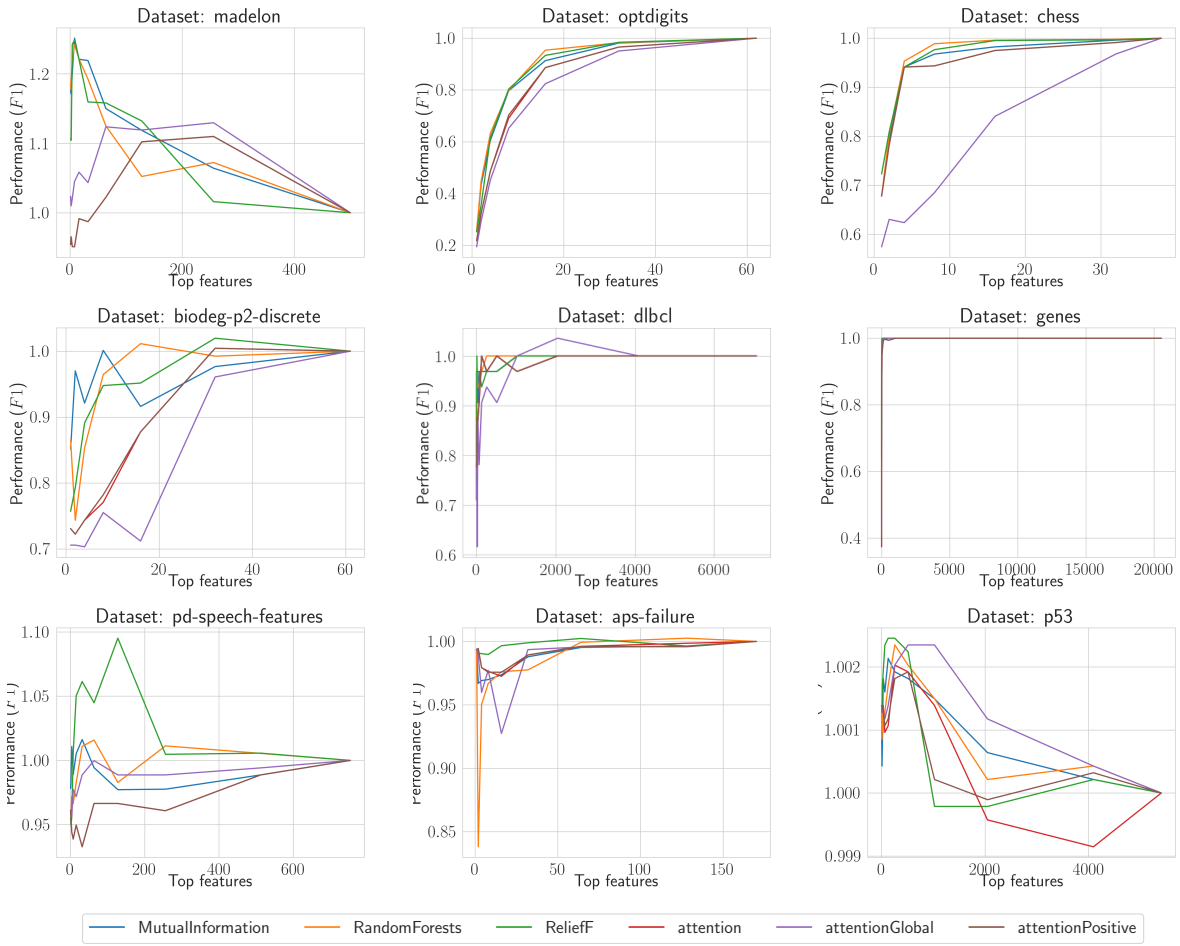


Figure 5: Feature ranking performance for different classifiers across different data sets. We report the *relative* F1 score, i.e., performance with respect the setting where all features are used as inputs to the logistic regression classifier.

ACKNOWLEDGEMENTS

We acknowledge the financial support from the Slovenian Research Agency through core research program P2-0103, and project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078). The authors have received funding also from the European Union's Horizon 2020 research and innovation program under grant agreement No 825153 (EMBEDDIA). The work of the first and the last author was funded by the Slovenian Research Agency through a young researcher grant.

A Obtaining attention via feature-level activation

We defined global attention as:

$$R_g = \frac{1}{k} \bigoplus_k \left[\text{softmax}(diag(W^k_{l_{\text{att}}})) \right]; W^k_{l_{\text{att}}} \in \mathbb{R}^{|F| \times |F|}.$$

One could consider a similar scheme, with the difference that the softmax could be applied for each feature w.r.t. the remainder of the features, and simply extracted as the main diagonal of the obtained matrix. Let RWS be defined as:

$$\text{RWS} = \text{concatByRows}(\text{softmax}(W_{l_{\text{att}}}(i))); \text{i is a row.}$$

Where $W_{l_{\text{att}}}(i)$ corresponds to the i-th row of the weight matrix. Thus, the global attention can be computed as:

$$R_g = \frac{1}{k} \bigoplus_k \left[diag(\text{RWS}(W^k_{l_{\text{att}}})) \right].$$

REFERENCES

[1] Ethem Alpaydin and Cenk Kaynak, 'Cascading classifiers', *Kybernetika*, **34**(4), 369–374, (1998).
[2] Sercan O Arik and Tomas Pfister, 'Tabnet: Attentive interpretable tabular learning', *arXiv preprint arXiv:1908.07442*, (2019).
[3] Scott A Armstrong, Jane E Staunton, Lewis B Silverman, Rob Pieters, Monique L den Boer, Mark D Minden, Stephen E Sallan, Eric S Lander, Todd R Golub, and Stanley J Korsmeyer, 'Mll translocations specify a distinct gene expression profile that distinguishes a unique leukemia', *Nature genetics*, **30**(1), 41–47, (2002).
[4] Leo Breiman, 'Random forests', *Machine Learning*, **45**(1), 5–32, (2001).
[5] Tianqi Chen and Carlos Guestrin, 'Xgboost: A scalable tree boosting system', in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM, (2016).

[6] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio, 'Attention-based models for speech recognition', in *Advances in neural information processing systems*, pp. 577–585, (2015).

[7] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning, 'What does bert look at? an analysis of bert's attention', *arXiv preprint arXiv:1906.04341*, (2019).

[8] Camila Ferreira Costa and Mario A Nascimento, 'Ida 2016 industrial challenge: Using machine learning for predicting failures', in *International Symposium on Intelligent Data Analysis*, pp. 381–386. Springer, (2016).

[9] Samuel A Danziger, Roberta Baronio, Lydia Ho, Linda Hall, Kirsty Salmon, G Wesley Hatfield, Peter Kaiser, and Richard H Lathrop, 'Predicting positive p53 cancer rescue regions using most informative positive (mip) active learning', *PLoS computational biology*, **5**(9), e1000498, (2009).

[10] Li Deng, Dong Yu, et al., 'Deep learning: methods and applications', *Foundations and Trends® in Signal Processing*, **7**(3–4), 197–387, (2014).

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805*, (2018).

[12] Sašo Džeroski, Hendrik Blockeel, Boris Kompare, Stefan Kramer, Bernhard Pfahringer, and Wim Van Laer, 'Experiments in predicting biodegradability', in *International Conference on Inductive Logic Programming*, pp. 80–91. Springer, (1999).

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[14] Isabelle Guyon, 'Design of experiments of the nips 2003 variable selection benchmark', in *NIPS 2003 workshop on feature extraction and feature selection*, (2003).

[15] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror, 'Result analysis of the nips 2003 feature selection challenge', in *Advances in neural information processing systems*, pp. 545–552, (2005).

[16] Vân Anh Huynh-Thu, Alexandre Irrthum, Louis Wehenkel, and Pierre Geurts, 'Inferring regulatory networks from expression data using tree-based methods', *PLoS One*, **5**(9), 1–10, (2010).

[17] Paul Jaccard, 'Etude de la distribution florale dans une portion des alpes et du jura', *Bulletin de la Societe Vaudoise des Sciences Naturelles*, **37**, 547–579, (1901).

[18] David H Kemsley, Tony R Martinez, and D Campbell, 'A survey of neural network research and fielded applications', *International journal of neural networks*, **2**, 123–123, (1992).

[19] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter, 'Self-normalizing neural networks', in *Advances in neural information processing systems*, pp. 971–980, (2017).

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, 'Deep learning', *nature*, **521**(7553), 436–444, (2015).

[21] Yongjie Lin, Yi Chern Tan, and Robert Frank, 'Open sesame: Getting inside bert's linguistic knowledge', *arXiv preprint arXiv:1906.01698*, (2019).

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, 'Roberta: A robustly optimized bert pretraining approach', *arXiv preprint arXiv:1907.11692*, (2019).

[23] Scott M Lundberg and Su-In Lee, 'A unified approach to interpreting model predictions', in *Advances in Neural Information Processing Systems*, pp. 4765–4774, (2017).

[24] Hanchuan Peng, Fuhui Long, and Chris Ding, 'Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy', *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8), 1226–1238, (2005).

[25] Matej Petković, Luke Lucas, Dragi Kocev, Sašo Džeroski, Redouane Boumghar, and Nikola Simidjievski, 'Quantifying the effects of gyroless flying of the mars express spacecraft with machine learning', in *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 9–16, (July 2019).

[26] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga, 'Ensemble deep learning for regression and time series forecasting', in *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pp. 1–6. IEEE, (2014).

[27] Marko Robnik-Šikonja and Igor Kononenko, 'Theoretical and empirical analysis of relieff and rrelieff', *Machine learning*, **53**(1-2), 23–69, (2003).

[28] Yasubumi Sakakibara, 'Learning context-free grammars using tabular representations', *Pattern Recognition*, **38**(9), 1372–1383, (2005).

[29] Betul Erdogdu Sakar, M Erdem Isenkul, C Okan Sakar, Ahmet Sertbas, Fikret Gurgen, Sakir Delil, Hulya Apaydin, and Olcay Kursun, 'Collection and analysis of a parkinson speech dataset with multiple types of sound recordings', *IEEE Journal of Biomedical and Health Informatics*, **17**(4), 828–834, (2013).

[30] Alen David Shapiro, 'The role of structured induction in expert systems', (1984).

[31] Ira Shavitt and Eran Segal, 'Regularization learning networks: Deep learning for tabular datasets', in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 1386–1396, USA, (2018). Curran Associates Inc.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, 'Dropout: a simple way to prevent neural networks from overfitting', *The journal of machine learning research*, **15**(1), 1929–1958, (2014).

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', in *Advances in Neural Information Processing Systems*, pp. 5998–6008, (2017).

[34] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman, 'Superglue: A stickier benchmark for general-purpose language understanding systems', *arXiv preprint arXiv:1905.00537*, (2019).

[35] John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, Joshua M Stuart, Cancer Genome Atlas Research Network, et al., 'The cancer genome atlas pan-cancer analysis project', *Nature genetics*, **45**(10), 1113, (2013).

[36] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu, 'A comprehensive survey on graph neural networks', *arXiv preprint arXiv:1901.00596*, (2019).

[37] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le, 'Xlnet: Generalized autoregressive pretraining for language understanding', *arXiv preprint arXiv:1906.08237*, (2019).

5.3 Embedding-based Feature Ranking in High Dimensions

The final core contribution of this thesis explores the possibilities of performing fast feature ranking when high-dimensional data sets are considered. We investigated to what extent can the existing, Relief-based (Kira & Rendell, 1992), feature ranking approaches benefit from utilizing *learned* representations of both input and output spaces to learn more efficiently (Hypothesis 4, Section 1.3). Even though adopting low-dimensional representations to learn more efficiently has been previously explored to speed up learning, many such approaches are not adapted to handle *sparse* data well. This, in practice, results in large memory consumption, which renders a given algorithm less useful when considering, e.g., biomedical data sets, where many dimensions are present, albeit the data sets can be very sparse.

We explored whether the current state-of-the-art embedding algorithm UMAP (McInnes et al., 2018) can offer sufficient performance when considering high-dimensional data sets. Furthermore, by rewriting the multiclass and multilabel variants of the RelieF(F) algorithm to operate with sparse matrices, we demonstrated significant speedups and relatively low-performance degradation. Even though UMAP handles larger sparse data sets, its initialization remains prohibitive when considering data sets with many instances. This is due to its spectral initialization, which at some point results in quadratic space requirements (pairwise distances between the instances are stored for graph minimization). The proposed ReliefE detects such scaling problems up-front and adapts by considering only a sub-sample of the whole data set if needed. Hence, the presented contribution does not focus solely on building an embedding-based Relief algorithm but also explores how to push further the scaling capabilities of existing embedding algorithms. The paper relevant to this section is the following one:

> Škrlj, B., Džeroski, S., Lavrač, N., & Petković, M. (2021). ReliefE: feature ranking in high-dimensional spaces via manifold embeddings. *Machine Learning*, 1–45. https://doi.org/10.1007/s10994-021-05998-5

5.3.1 Key Contributions

We next present the key contributions of the conducted work.

1. We present ReliefE, an algorithm from the Relief family capable of feature ranking by exploiting learned, low-dimensional representations of instances.

2. The implementation of ReliefE that supports embeddings of both target and the input spaces scales seamlessly to very high-dimensional data sets and was shown to perform competitively.

3. To scale ReliefE, we also improved the training regime of the selected embedding algorithm; if a given input data set is too large, it selects a representative subspace and estimates the projection of instances/targets based on this subspace, scaling beyond the capabilities of out-of-the-box implementation.

4. ReliefE automatically determines the *latent dimension* of the input/target space prior to embedding, thus omitting the need for manual tuning of this hyperparameter.

5. We demonstrate both theoretically and empirically that embedding-based ranking is a promising research endeavour.

5.3.2 Addressed Hypotheses and Discussion

The developed ReliefE algorithm adopts the instance/target embeddings to scale the Relief-based feature ranking better. The main hypothesis related to the presented work concerns model scalability (Hypothesis 4, Section 1.3). We demonstrated, both in theory and in practice, that substantial speedups can be observed by reducing the number of features ($|F| \rightarrow d$). Furthermore, by considering only representative subspaces, the proposed approach scales better also with respect to the number of instances. This second constraint, in particular, appeared problematic when attempting to leverage contemporary embedding learners (UMAP (McInnes et al., 2018) in this case). The need to better scale the embedding estimation based on the representative subspace was not one of the goals of the paper; however, it was needed for ReliefE to scale properly to the largest considered data sets. A substantial amount of time was spent on implementing ReliefE via Numba-based (compilable) operations. Such implementation, at least in theory, maintains the simplicity of use (a simple-to-use library without the need for explicit compilation). As reported, we achieved mixed results with this approach. On the one hand, substantial speedups were observed if the operations could be easily compiled (multi-fold). However, in some scenarios, we were not certain which parts offered speedups and which did not; hence additional profiling was conducted. This step resulted in the realization that in order to implement very fast Numba-compatible, e.g., Relief updates, the code becomes harder to maintain, slowly converging towards, e.g., C++-like syntax. We believe, however, that transpillation [1] of higher-order languages, when possible, is a very interesting and straightforward approach to obtaining substantial speedups on a given problem. In time, such libraries/approaches for transpilation are expected to become only better. Hence, having an understandable and easy-to-maintain codebase with the benefit of parts of it being compiled, remains an interesting design decision for scientific computing. Overall, the presented work demonstrates that adopting contemporary embedding algorithms, which enable the comparison of instances/targets in the latent space, offers substantial speedups. The proposed method can be interpreted as a neuro-symbolic feature ranking approach, where the embedding learning is considered sub-symbolic (neural), and the ranking is considered symbolic.

The results indicate that exploration of how the existing paradigm of representation learning could be of use when improving algorithms that have been present and used for decades is a promising research venue. Furthermore, the results indicate that if the learned representations are not of high quality, i.e. the latent space is not reflective of the original space, the subsequent ranking will most likely underperform. This trade-off is expected, as it adheres to the noise-in noise-out principle. The first hypothesis (better understanding; Section 1.3) was also partially addressed. As ReliefE leverages both the symbolic and the sub-symbolic learning paradigm, its behaviour could be studied with respect to the learned representation's quality. As the embedding algorithms can be understood as *lossy compression engines*, performance degradation is a potential side effect of coupling the two paradigms. However, it is not necessarily the case that immediate performance degradation takes place – note that the initial data sets can include much noise, indicating that compressing the data set to a high-enough dimension could also have de-noising effects (better performance). To summarize the findings, we confirmed that the neuro-symbolic paradigm can offer more scalable solutions to existing Relief-based feature ranking approaches and that the overall performance of such a system depends on the embeddings' quality, which can be a limiting factor.

[1] Transpillation corresponds to translation of source from one language to source in a different language. In the context of this thesis, it was mostly considered for performance reasons (e.g., speeding up Python).

Machine Learning
https://doi.org/10.1007/s10994-021-05998-5

ReliefE: feature ranking in high-dimensional spaces via manifold embeddings

Blaž Škrlj[1,2] · **Sašo Džeroski**[1,2] · **Nada Lavrač**[1,3] · **Matej Petković**[1,2]

Received: 6 March 2020 / Revised: 25 January 2021 / Accepted: 13 May 2021

Abstract

Feature ranking has been widely adopted in machine learning applications such as high-throughput biology and social sciences. The approaches of the popular Relief family of algorithms assign importances to features by iteratively accounting for nearest relevant and irrelevant instances. Despite their high utility, these algorithms can be computationally expensive and not-well suited for high-dimensional sparse input spaces. In contrast, recent embedding-based methods learn compact, low-dimensional representations, potentially facilitating down-stream learning capabilities of conventional learners. This paper explores how the Relief branch of algorithms can be adapted to benefit from (Riemannian) manifold-based embeddings of instance and target spaces, where a given embedding's dimensionality is intrinsic to the dimensionality of the considered data set. The developed ReliefE algorithm is faster and can result in better feature rankings, as shown by our evaluation on 20 real-life data sets for multi-class and multi-label classification tasks. The utility of ReliefE for high-dimensional data sets is ensured by its implementation that utilizes sparse matrix algebraic operations. Finally, the relation of ReliefE to other ranking algorithms is studied via the Fuzzy Jaccard Index.

Keywords Feature ranking · Representation learning · Relief

Editors: Petra Kralj Novak, Tomislav Šmuc.

✉ Matej Petković
matej.petkovic@ijs.si

Blaž Škrlj
blaz.skrlj@ijs.si

Sašo Džeroski
saso.dzeroski@ijs.si

Nada Lavrač
nada.lavrac@ijs.si

[1] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

[2] Jožef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia

[3] University of Nova Gorica, Vipavska 13, 5000 Nova Gorica, Slovenia

Published online: 17 June 2021

1 Introduction

Contemporary machine learning has found its use in many scientific disciplines, ranging from biology, sociology, logistics and engineering sciences to physics. Data sets are often available in tabular form and consist of instances (rows) and features (columns), where attributes denote column names and individual features correspond to individual attribute values.

Even though predictive models can offer insights into how well a certain aspect of a given system can be predicted, researchers and industry practitioners are frequently interested in *which* parts of the input space are the most relevant. Having such knowledge can yield novel insights into relevant aspects of the studied problem, leading to improved human understanding of the studied phenomenon. For example, in modern molecular and systems biology, discovery of novel biomarkers is of high relevance—once the researchers know which, e.g., compounds or proteins indicate the presence of the studied condition, they can be used for preliminary condition detection, but also to advance the human understanding of the conditions leading to the construction of novel hypotheses. We next discuss the types of feature ranking algorithms.

Feature ranking algorithms can be split into two main groups: myopic and non-myopic. Myopic algorithms do not consider multiple features simultaneously and thus potentially neglect *interactions* between features. Examples of myopic feature ranking algorithms include, e.g., the information gain-based ranking. Algorithms from the non-myopic Relief branch, originating in the early Relief algorithm (Kira and Rendell 1992), are among the most widely used non-myopic algorithms for *feature ranking*, where each feature is assigned a real-valued score, offering insights into its importance. The Relief family of algorithms has been successfully applied to numerous real-life problems (Stiglic and Kokol 2010; Stokes and Visweswaran 2012; Petković et al. 2021). In this work we propose ReliefE, an embedding-based feature ranking algorithm built on the ideas of the original Relief and ReliefF (Robnik-Šikonja and Kononenko 2003), as well as their extensions to a multi-label classification setting (Petković et al. 2018). ReliefE does not compute feature importances based on the original, high-dimensional feature space, but via low-dimensional embeddings of the input and/or output spaces. The key contributions of this work are summarized as follows:

- We present ReliefE, an algorithm for feature ranking implemented using sparse matrix algebraic computation of distances between low-dimensional manifold embeddings of both instances and targets (when considering multi-label classification).
- The latent dimension of the space, in which the distances are computed, is *inferred automatically* in an efficient manner.
- We show that the number of neighbors to be considered can be automatically inferred based on the distribution of distances to the considered instances, rather than hard-coded as part of the input.
- Theoretically grounded sparsification of the input was considered as a preprocessing step, potentially decreasing the execution time.
- We offer evidence that ReliefE performs significantly faster than many state-of-the-art methods, especially in high-dimensional input (and output) spaces.
- Theoretical properties of ReliefE, including the properties of the embedding spaces, their relations and computational complexity analysis are studied.

- We showcase the ReliefE's performance against six strong (widely used) baselines on 20 real-life multi-class and multi-label classification data sets.
- We perform extensive Bayesian and frequentist performance comparisons assessing the statistical evidence of ReliefE's utility and potential drawbacks.

The rest of this paper is structured as follows. In Sect. 2, we discuss the key ideas that have led to the developments described in this paper. Section 3 presents the proposed ReliefE methodology, followed by a description of the experimental setting in Sect. 4 and the results of the empirical evaluation in Sect. 5. The overall findings are discussed in Sect. 6, followed by the conclusions and plans for further work in Sect. 7. The paper includes numerous appendices presenting detailed results of empirical comparisons and case studies.

2 Background

In this section we discuss the works that have impacted this paper the most, starting with the notion of feature ranking and the description of the Relief branch of feature ranking algorithms. Next, we discuss how embedding-based learning can be of use when solving otherwise intractable problems, serving as a motivation for the proposed work.

2.1 Feature ranking

Feature ranking can be considered as the process of learning to prioritize the feature space with respect to a given learning task. Algorithms that rank features can operate in non-myopic (considering feature interactions) or myopic manner (ignoring feature interactions). One of the first and most widely used algorithms for non-myopic feature ranking is Relief (Kira and Rendell 1992), introduced in the early 1990s. This iterative algorithm operates by randomly selecting parts of the instance space (e.g., rows), followed by iterative update of weights corresponding to individual features based on the closest instances from the positive and negative classes. The original Relief performs well for binary classification, however was unable to generalize to more complex learning tasks such as multi-class classification. Its extension, ReliefF (Robnik-Šikonja and Kononenko 2003), introduced a prior-based weighting scheme that can take different classes into account. In the following years, multiple adaptations of both Relief and ReliefF were introduced, varying mostly in terms of schemes for taking into account a given instance's neighborhood and its (aggregated) properties. For example, SURF (Greene et al. 2009) unifies the considered per-class neighborhoods, whereas SURFstar (Greene et al. 2010) additionally considers distant neighbors. Further, MultiSURFstar (Granizo-Mackenzie and Moore 2013) takes neighborhood boundaries into account based on the average and standard deviation of distances from the target instance to all others. The TuRF adaptation (Moore and White 2007) of any Relief algorithm also employs recursive feature elimination, whilst applying the dedicated Relief implementation iteratively during feature pruning. TuRF attempts to address some of the problems that arise in very large feature spaces (e.g., more than 20,000 features), yet at the cost of higher computational complexity. In terms of scalability, for example, VLSReliefF (Eppstein and Haake 2008) samples *random subspaces* whilst simultaneously offering competitive performance at a far lower computational cost. The above

ReliefF variants mostly attempt to correct some of the original ReliefF drawbacks by re-considering the update step and how the neighbors are taken into account.

In recent years, the Relief algorithms have also been extended to a multi-label classification setting (MLC)—a learning problem where multiple labels for an instance are simultaneously possible. Examples of this task include gene function prediction. The first attempt of extending ReliefF to MLC that scales well with the number of labels (Petković et al. 2018) uses the Hamming distance as the distance measure between two label sets. Since Hamming loss can be expressed as a sum of component-wise differences, it is not able to detect the possible label-label interactions, which may lead to sub-optimal quality of the obtained rankings as shown recently (Petković et al. 2018), where other MLC error measure-based distances between labels were shown to offer superior performance.

2.2 Embeddings

The rise of embedding-based learning can be nowadays observed in virtually all areas of science and industry. Since the 2010s, for example, deep neural networks are successfully used in fields such as computer vision, where state-of-the-art results are consistently demonstrated, e.g., in face recognition and anomaly detection (LeCun et al. 2015; Pouyanfar et al. 2018). Further, in the recent years, natural language processing has been transformed first by the introduction of recurrent neural networks, followed by the attention-based neural networks (transformers), showing prominent results in the areas of question answering, language understanding and text classification (Vaswani et al. 2017). Similar results have been observed in the areas of network mining (Kipf and Welling 2016) and time series analysis (based on the initial work of Connor et al. 1994).

Neural networks offer an elegant alternative for learning a latent representation of the input data set that can be *transferred*, as well as directly used for problem solving. More recent works on representation learning, however, suggest that a low-dimensional manifold is a suitable topological object for learning rich and transferable representations (Bronstein et al. 2017; Masci et al. 2015). Even though representations learned by neural networks can be associated with manifold learning, the development of algorithms capable of approximating such low-dimensional manifolds has been an active research area even before the era of deep learning, and is of particular relevance to this work. For example, algorithms such as Isomap (Balasubramanian and Schwartz 2002) and Locally linear embedding (Roweis and Saul 2000) have been successfully used for data visualization and more efficient learning. Further, modern *omics* sciences have widely adopted t-SNE (Maaten and Hinton 2008), an approximation method capable of producing two dimensional embeddings of high-dimensional spaces, making it suitable for e.g., gene expression visualization. Hyperbolic embeddings have been also recently demonstrated to be useful when considering hierarchical multi-label classification (Stepišnik and Kocev 2020).

This work builds on the notion of uniform manifold projections (UMAP) (McInnes et al. 2018a, b), a recently introduced algorithm with a strong theoretical grounding in *manifold theory*. We explore, whether low-dimensional manifold approximations of sparse input spaces can be a natural extension to the Relief family of algorithms.

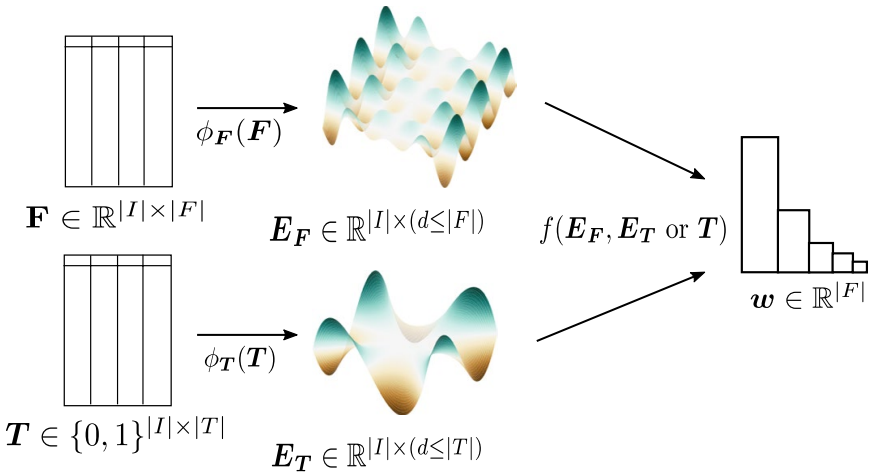


Fig. 1 Overview of the core idea behind ReliefE. Distances in both the feature and target space are computed based on instance embeddings

3 Proposed methodology

One of the main criticisms of the Relief branch of algorithms is their lack of ability to handle very high-dimensional, potentially sparse input spaces, where problems arise either due to increased space complexity or due too incremental weight update steps that result in similar importance scores for many features (i.e. non-informative rankings). We developed the proposed ReliefE algorithm with the goal to addresses these issues. In this section, we discuss in detail ReliefE, the proposed embedding-based feature ranking algorithm for multi-class and multi-label classification problems, along with its implementation, currently one of the fastest Python-based implementations compiled to machine code capable of handling both multi-class and multi-label classification problems.

The proposed ReliefE algorithm is summarized in Fig. 1. Here, the input feature space ($\boldsymbol{F}$) is mapped (by $\boldsymbol{\phi_F}$) to its corresponding low-dimensional approximation ($\boldsymbol{E_F}$). Finally, the Relief feature ranking (f) is adapted to operate via this low-dimensional representation to obtain final feature rankings $\boldsymbol{w}$. Further, in a multi-label setting, distances between targets ($\boldsymbol{T}$) can also be computed in the latent space $\boldsymbol{E_T}$, constructed via $\boldsymbol{\phi_T}$.

3.1 Rationale for embedding-based ranking

Embedding the instances in the feature space prior to feature ranking is sensible due to the fact that volume increases *exponentially* with dimension. Many classifiers benefit from increasing the number of dimensions, however, once a certain dimensionality is reached, their performance can degrade (Hughes 1968). Feature ranking aids this problem by prioritizing parts of the feature space that are relevant for learning.

Higher-dimensional feature spaces render feature importance detection in the initial feature space harder, as more subtle differences between instances need to be taken into account. Embedding (in an unsupervised manner) offers the construction of instance representations that potentially carry additional *semantic context*, as comparing instances in the embedded space does not compare only the considered pairs of instances but also their potential *roles* in the joint latent space, offering more meaningful comparisons (as shown

in e.g., Mikolov et al. 2013 for words and phrases). We next discuss the embedding method considered throughout this work.

3.2 Uniform manifold approximation and projection

This work builds on the recently introduced idea of Uniform Manifold Approximation and Projection (UMAP) (McInnes et al. 2018a) for the task of low-dimensional, unsupervised representation learning. Even though a detailed treatment of the theoretical underpinnings of UMAP is beyond the scope of this paper, we discuss some of the key ideas underlying the actual implementation, and refer the interested reader to McInnes et al. (2018a) for a detailed overview of the theory.

UMAP is formulated with concepts from both topological manifold theory and applied category theory. Riemannian manifolds are topological spaces that have a locally constant metric, and are locally connected. UMAP assumes that high dimensional data can be uniformly mapped across a low-dimensional Riemannian manifold. The locality of the metric, connectivity and uniformity of the mapping are the three main assumptions of this method. Even though such assumptions are not necessarily fulfilled, appropriate selection of the metric used by UMAP can offer better performance and generalization when the learned representations are used for down-stream learning tasks. In contrast to t-SNE, which is effective for two dimensional embeddings, UMAP is highly efficient also for embeddings into *higher-dimensional* vector spaces. UMAP thus serves as a general unsupervised representation learner.[1] It has been successfully used for exploration of biological and other high-dimensional data sets (Cao et al. 2019). In summary, the topological manifold theory underlying UMAP offers a very general representation learning framework, applicable beyond the current implementation of UMAP, which we discuss in more detail below.

Even though the original formulation assumes continuity, in practice, discrete graph-theoretical analogs of the continuous concepts are employed. The representations are a result of a two-step procedure, where in the first step, a weighted graph is constructed based on the distances between the closest instances. The second step resembles conventional graph layout computation, which is normally computed in two or three dimensions for visualization purposes, where, e.g., two coordinates of a 2D layout represent two features. Analogously, UMAP extends the idea to higher dimensions, where the instances are positioned in a d-dimensional space (with d going up to several hundred in most cases). The resulting space is not useful for direct visualization, but serves as a representation suitable for a down-stream learning task—in this work, feature ranking. The computation of the UMAP embedding can be described as follows.

Weighted graph construction. Assume a user-specified dissimilarity measure $\delta : \mathbb{R}^{|F|} \times \mathbb{R}^{|F|} \to [0, \infty)$ and the number of nearest neighbours k, computed via an approximation algorithm (Dong et al. 2011). We refer to the ordered set of instances nearest to instance $\boldsymbol{r}_i$ as $\{\boldsymbol{r}_{i_1}, \ldots, \boldsymbol{r}_{i_k}\}$. For each instance, let

$$\omega_i = \min \{\delta(r_i, r_{i_j}) | 1 \leq j \leq k, \delta(r_i, r_{i_j}) > 0\},$$

[1] UMAP can also perform supervised embeddings, yet this functionality is not considered in this work.

and β_i such that

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, \delta(r_i, r_{r_j}) - \omega_i)}{\beta_i}\right) = \log_2(k).$$

Next, a weighted directed graph $G = (N, E, w)$ is constructed, where N is the set of considered instances, $E = \{(r_i, r_{i_j}) | 1 \leq j \leq k\}$ and

$$w((r_i, r_{i_j})) = \exp\left(\frac{-\max(0, \delta(r_i, r_{i_j}) - \omega_i)}{\beta_i}\right).$$

The final adjacency matrix $\boldsymbol{B}$ is computed as $\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{A}^T - \boldsymbol{A} \odot \boldsymbol{A}^T$ where, $\boldsymbol{A}$ is the adjacency matrix of G and $\odot$ denotes the Hadamard product.

Layout computation. In the second step, the graph undergoes a process resembling energy minimization, where repulsive and attractive forces are iteratively applied across pairs of instances, resulting in a d-dimensional layout, which is effectively a real-valued embedding. The attractive forces (F_+) are computed, for a given pair of vertices n_i and n_j and their corresponding coordinate representations (embeddings) $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ as follows:

$$F_+ = \frac{-2 \cdot a \cdot b \cdot ||\boldsymbol{r_i} - \boldsymbol{r_j}||_2^{2(b-1)}}{1 + ||\boldsymbol{r_i} - \boldsymbol{r_j}||_2^2} \cdot w(n_i, n_j) \cdot (\boldsymbol{r_i} - \boldsymbol{r_j}),$$

and similarly, the repulsive forces (F_-) are computed as

$$F_- = \frac{b \cdot (1 - w(n_i, n_j)) \cdot (\boldsymbol{r_i} - \boldsymbol{r_j})}{(\eta + ||\boldsymbol{r_i} - \boldsymbol{r_j}||_2^2)(1 + ||\boldsymbol{r_i} - \boldsymbol{r_j}||_2^2)}.$$

Here, η is introduced for numerical stability (a small constant), while a and b are hyperparameters. Note that the F_- update is computationally very expensive, and is addressed via sampling. The initial coordinates are computed by using spectral layout—here, the two eigenvectors corresponding to the two largest eigenvalues are used as the starting set of coordinates.

The two steps, when implemented efficiently, offer fast construction of d-dimensional, real-valued representations. The considered UMAP implementation exploits the Numba framework (Lam et al. 2015) for compiling parts of Python code, making it scalable whilst maintaining user-friendly API-based execution. Note that UMAP's computational complexity depends heavily on the approximate k-nearest neighbor computation. In the following sections, we discuss how we further facilitate the embedding computation, as the current version of UMAP still has *high space complexity* (graph construction).

3.3 Input sparsification

The proposed methodology is capable of handling highly sparse inputs without additional memory overheads. However, real-life data frequently comes in the form of dense matrices, as is the case with, e.g., gene expression data sets. As a part of the proposed methodology, we explored whether input sparsification can speed up the feature ranking process

with minimal ranking quality degradation. We implemented the theoretically grounded Probabilistic Matrix Sparsification algorithm (PrMS) for matrix sparsification (Arora et al. 2006), given in Algorithm 1.

The mathematical intuition behind PrMS is as follows. Given a real-valued matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, let $S = \sum_{ij} |A_{ij}|$. A single PrMS pass through $\boldsymbol{A}$ guarantees at least $\mathcal{O}(\frac{\sqrt{n \cdot S}}{\Delta})$ elements with probability $1 - \exp(-\Omega(\frac{\sqrt{n \cdot S}}{\Delta}))$. Here, Ω represents the lower bound, and $\Delta = \epsilon / \|\boldsymbol{A}\|_2$, where parameter $\epsilon > 0$ determines the approximation level. Further, with probability $1 - \exp(-\Omega(n))$, $||\boldsymbol{A} - \hat{\boldsymbol{A}}||_2 \leq \mathcal{O}(\epsilon)$ holds.[2] Note that, as the majority of real-life data sets are not represented by symmetric matrices (typically, they are not even square matrices), the transformation $\boldsymbol{B} \mapsto \boldsymbol{A} = \begin{bmatrix} 0 & \boldsymbol{B} \\ \boldsymbol{B}^T & 0 \end{bmatrix}$ of the initial matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ has to be employed since $\boldsymbol{A}$ is symmetric and $\|\boldsymbol{A}\|_2 = \|\boldsymbol{B}\|_2$. We heuristically consider $\epsilon = \|\boldsymbol{A}\|_\infty / (m + n)$, i.e. the maximal column-average (of absolute values) of matrix $\boldsymbol{A}$.

The sparsification procedure remains highly dependent only on ϵ, the parameter determining the reconstruction error that is allowed.

We have used this estimate of ϵ as it is fast to compute and avoids the need for user specification of ϵ, whilst simultaneously guaranteeing reasonable performance (see Sect. 5). One of the most crucial hyperparameters related to representation learning in general is the dimension of the space in which the constructed representation resides. We have attempted to automate the choice of this—otherwise hard-coded—parameter and discuss the considered estimate next.

3.4 Estimation of latent dimension

Following (Facco et al. 2017), we improve upon the idea of latent dimension estimation via top two nearest neighbors. To compute the latent dimension d of the data (under the assumption of a locally constant probability density), it suffices to define two (hyper) spheres S_1 and S_2. Both are centred at a random data sample and have radii equal to the distances between the sample and its two nearest neighbors (Facco et al. 2017). The radii and the dimension d define the volumes of the spheres and it turns out that the value of d can be easily estimated from the empirical probability distribution of the ratio $V(S_2 \backslash S_1)/V(S_1)$ of the volumes of the shell $S_2 \backslash S_1$ and sphere S_1. The method, implemented as a part of ReliefE is summarized in Algorithm 2.

Algorithm 1: Probabilistic Matrix Sparsification (PrMS) [5]

Data: Input matrix $\boldsymbol{A}$, number of elements n, approximation constant ϵ

for $i, j \in n$ **do**
  if $|\boldsymbol{A}_{ij}| > \frac{\epsilon}{\sqrt{n}}$ **then**
    $\hat{\boldsymbol{A}_{ij}} = \boldsymbol{A}_{ij}$
  else
    $\hat{\boldsymbol{A}}_{ij} = \begin{cases} \text{sgn}(\boldsymbol{A}_{ij}) \cdot \frac{\epsilon}{\sqrt{n}}; \text{ with probability } p_{ij} = \frac{\sqrt{n}|\boldsymbol{A}_{ij}|}{\epsilon} \\ 0; else \end{cases}$
return $\hat{\boldsymbol{A}}$

[2] For extensive theoretical treatise, please consult (Arora et al. 2006).

Algorithm 2: Latent dimension estimation [16].

Data: Set of instance indices I, (sparse) feature matrix $\mathbf{F} \in \mathbb{R}^{|I| \times |F|}$
distanceMatrix ← pairwiseDistancesNonzero($\boldsymbol{F}$)
μ ← distanceMatrix[:,1]/ distanceMatrix[:,0]
empiricalDist ← computeEmpiricalDistribution(μ)
d ← leastSquaresLine(log(μ), log(1 - empiricalDist))
return d

The algorithm first computes distanceMatrix, i.e. a matrix comprised of distances to the top two nearest neighbors. In this work, we improve upon the original idea of simply computing the two nearest neighbors of a given instance. Instead, we ignore the neighbors that are equal to a given instance (are at distance 0), and take into account the two nearest neighbors that are at a positive distance to the given instance (method pairwiseDistancesNonzero). The rationale for this step is that this method is also used on the output space of multi-label classification data, where two (or more) examples can often have the same output value (vector describing the labels assigned to the instance). If we had followed the original method, some components of the vector μ would equal ∞ or NaN. Using the modified procedure, numerical instability is avoided whilst observing the same, or very similar, results. Next, μ, a quotient between the two distance vectors is computed (second closest against closest). An empirical distribution is derived from μ. This distribution can be defined as:

$$\mathrm{EMP}_n(x) = \frac{1}{|I|} \sum \mathbb{I}_{x_i < x},$$

where $\mathbb{I}$ represents the indicator function (presence). Thus, for a given x, F_n represents the relative number of elements that are smaller than x. The logarithms of μ, as well as (1 - $\mathrm{EMP}_n(x)$) are computed next. The line between the two quantities intersects 0, and its coefficient, when rounded to the nearest integer, corresponds to the estimated latent dimension. For example, the intrinsic dimension of the *genes* data set is estimated to 33 (see Fig. 2).

3.5 Scaling up UMAP: learning to embed based on representative subspaces

As the most apparent memory (space) bottleneck, we (empirically) identified the UMAP's graph construction phase, where the entirety of the instance space is used for learning to approximate the low-dimensional manifold (embedding). This paper explores whether representative subspaces of the instance space can serve similarly well for learning to embed. The idea was inspired by recent work on random output space selections for ensemble learning (Breskvar et al. 2018). The two adaptations we needed to consider were the initialization and training on a representative subspace of the instances. In the original UMAP, the initialization is based on the spectral decomposition of the instance graph. We instead considered random initialization, which already notably reduced the memory requirement, and kept the quality of the ranking at the approximately same level. However, on larger data sets (see Sect. 4), the embedding computation was still beyond the capabilities of an off-the-shelf computer (Lenovo Carbon X1). To overcome this issue, we employ the following two-step sampling scheme. In the first step, the set of target values appearing in the data was ordered into a list. In the second step, we *cyclically* iterate through the list and

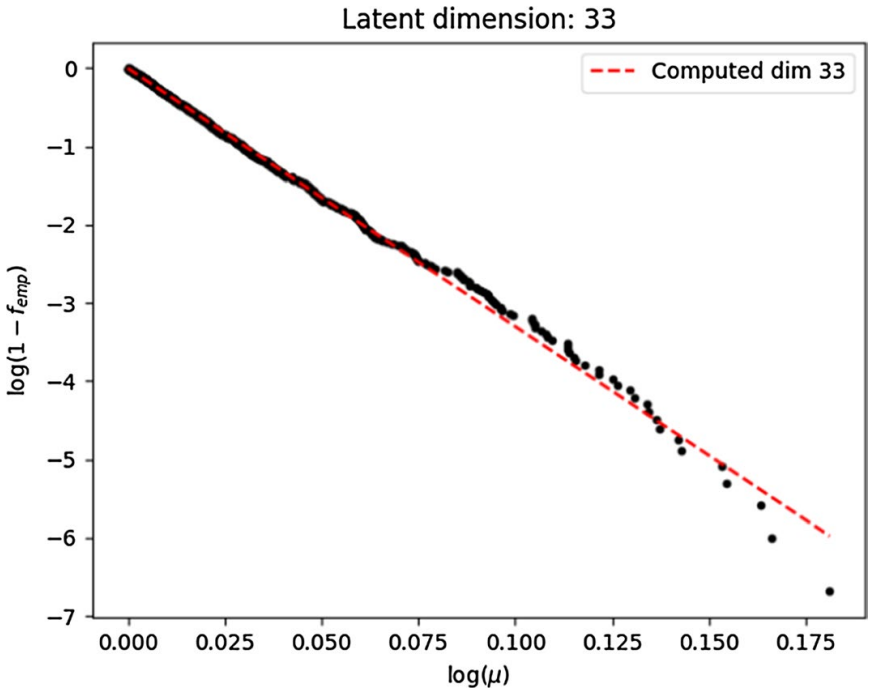


Fig. 2 Intrinsic dimension for the *genes* (Weinstein et al. 2013) data set

(without repetition) choose an element with a given target value, or skip this value if all samples labeled with the considered target value have already been chosen.

The multi-class (and binary) classification examples thus adhere to straightforward mapping from possible classes to the corresponding *multisets* of instances. We also extended this idea to the task of multi-label classification (MLC) where multiple labels can be simultaneously assigned a given example. However, a finite number of possible different label sets allows for first enumerating them, and then proceeding as in the multi-class classification scenario. We consider the theoretical properties of this procedure in Sect. 3.8.

3.6 Adaptive neighbor selection and comparison to average neighbor representation

The first improvement we introduce next, removes the hyperparameter k—the number of neighbors considered for an update step w.r.t. a single randomly chosen instance. Commonly, k is a user defined hyperparameter: However, we explored whether this part of the user input can be removed entirely and replaced by a distance distribution-based heuristic that *dynamically allocates* a number of neighbors suitable for a given randomly selected instance, albeit at some additional computational cost. The rationale for such a heuristic is that real-life data sets are often not uniformly distributed (Liu et al. 2005), indicating only a few other instances are mostly well connected with a given one (scale-free property).

We implement this (optional) estimation as follows. For each instance, ReliefF computes its distance to the remainder of the other instances in order to obtain the top k nearest ones. Such hard-coded selection scheme is not optimal, as it does not take into account the shape of the distance distribution with respect to an individual instance. To overcome this issue, we propose the following procedure:

1. Sort distances w.r.t. the selected instance $\boldsymbol{r}_i$.

2. Compute the difference between each pair of consequent entries in the distance vector.
3. Select the value of k based on the maximum difference observed.

This procedure intuitively takes into account the shape of the distance distribution as follows. Assuming that all instances are similarly far from the selected instance, the difference vector will mostly consist of small values (a given pair of distances is not all that different). This can result in large k, as the global difference maximum can occur very late. On the contrary, if only a handful of instances are close, and the remainder is very far, only this handful shall serve to determine k, which will be consequently lower. Furthermore, if very high k is selected and all distances are approximately the same, their mean will be similar no matter how many are selected. If a similar situation is considered for highly non-uniform distance distribution, the mean of selected k nearest instances should represent only the ones that are indeed similar to the selected instance, and do not take into account the remainder which is further and possibly not as relevant.

Even if the standard IID assumption holds when sampling a data set, we are always given a finite sample. The neighborhoods of the samples on the border of the convex hull of the data set are most probably different than the neighborhoods of those in the interior of the hull. Additional theoretical properties of this neighbor number selection are given in Sect. 3.8, where computational complexity is also considered.

The proposed adaptation of ReliefF was further extended with an additional option to use the *average neighbor* for the update step, instead of performing updates based on all neighbors and averaging the updates. Albeit subtle, this difference potentially improves the update part's robustness, making the update step less prone to possible outliers amongst the nearest neighbors—such situations could emerge, if the value of k is hard-coded, as is commonly the practice. The intuition behind this incremental change in weight update is as follows. As the distance computation is conducted in the *latent* space, averaging the representation of the neighborhood can also be considered as constructing a *semantic representation* of the considered instance's neighborhood.

3.7 ReliefE—ranking via manifold embeddings

In the following sections, we discuss more formally the proposed ReliefE algorithm incorporating the possible improvements stated in the previous sections. The solution (that handles both multi-class and multi-label classification) is given as Algorithm 3. We can see that this is an iterative algorithm where a random sample $\boldsymbol{r}$ is chosen on each of the n iterations, and distances between $\boldsymbol{r}$ and the remaining instances are computed (lines 6 and 18). We next discuss the two main parts of ReliefE with respect to the addressed classification task.

Algorithm 3: ReliefE.

Data: feature set F, instance index set I, (sparse) feature matrix $\mathbf{F} \in \mathbb{R}^{|I|\times|F|}$, target classes C, (sparse) target space $\mathbf{T} \in \{0,1\}^{|I|\times|C|}$, neigborhood size k, boolean adaptiveThreshold, distance score δ, latent dimension d, boolean estimateLatentDimension, MLC distance τ, number of iterations s, indices ν of dimension estimation examples

$\boldsymbol{w} \leftarrow$ zero list of length $|F|$ ▷ `Initiate importances.`
if *estimateLatentDimension* **then**
  $d \leftarrow$ latentDimension($\mathbf{F}, \nu$) $(\in \mathbb{N}_+)$
$\mathbf{E} \leftarrow$ manifoldProjection($|\mathbf{F}|$,d,ν) $\left(\in \mathbb{R}^{|I|\times d}\right)$
for *i in 1 ... s* **do**
  $\boldsymbol{r}_i \leftarrow$ randomInstance($\mathbf{F}$) ▷ `Sample instance.`
  if *MCC* **then**
    for $c \in C$ **do**
      dists $\leftarrow$ computeDistances($\boldsymbol{r}_i$,δ,$\mathbf{E}$,c)
      sortedIndices $\leftarrow$ argSortDistances(dists)
      if *adaptiveThreshold* **then**
        $k \leftarrow$ adaptiveThresholdMethod(sortedIndices, dists)
      nearestNeighbors $\leftarrow$ select(sortedIndices,k)
      for j *in* $1 \ldots |F|$ **do**
        priorWeight $\leftarrow$ ComputeWeight(c, dists, k)
        $\boldsymbol{w}[j] \leftarrow \boldsymbol{w}[j]$ + updateScore(priorWeight, nearestNeighbors, j)
  else if *MLC* **then**
    dists $\leftarrow$ computeDistances($\boldsymbol{r}_i$,δ,$\mathbf{E}$)
    sortedIndices $\leftarrow$ argSortDistances(dists)
    if *adaptiveThreshold* **then**
      $k \leftarrow$ adaptiveThresholdMethod(sortedIndices, dists)
    nearestNeighbors $\leftarrow$ sortedIndices[:k]
    targetDistances $\leftarrow \{\}$ ▷ `Note the direct use of T compared to MCC.`
    descriptiveDistances $\leftarrow \{\}$
    for *neighborIndex* $\in$ *nearestNeighbors* **do**
      distTar $\leftarrow$ distanceToTarget($\boldsymbol{T}[i]$, $\boldsymbol{T}$[neighbor], τ)
      distDes $\leftarrow$ distanceToDesc($\boldsymbol{E}[i]$, $\boldsymbol{E}$[neighbor], δ)
      targetDistances.add(distTar)
      descriptiveDistances.add(distDes)
    meanTarDist $\leftarrow$ mean(targetDistances)
    meanDesDist $\leftarrow$ mean(descriptiveDistances)
    TD-DIFF, D-DIFF, T-DIFF $\leftarrow$ expectedDist(meanTarDist, meanDesDist)
    for j *in* $1 \ldots |F|$ **do**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \frac{\text{TD-DIFF}}{\text{T-DIFF}} - \frac{\text{D-DIFF}-\text{TD-DIFF}}{1-\text{T-DIFF}}$. ▷ `See Eq. 1.`
return $\boldsymbol{w}$

The adaptive threshold step can be further formalized as shown in Algorithm 4.

Algorithm 4: adaptiveThresholdMethod.

Data: distances, sortedIndices
1 sortedDist ← reorderAscending(dists, sortedIndices)
2 sortedDiff[i]← sortedDist[$i+1$]−sortedDist[i] $\left(\in \mathbb{R}_{+}^{|F|-1}\right)$
3 $k \leftarrow \arg\max$ sortedDiff ▷ `Peak point.`
4 **return** k

3.7.1 Multi-class classification

We first discuss the part of ReliefE algorithm that handles multi-class classification (MCC) tasks. Here, the classes are traversed (line 8) as follows. If *adaptiveThreshold* is enabled, the number of neighbors to be considered is determined dynamically for each sample (see Sect. 3.6 for more details). Next, the neighbors are selected and used for the final weight update, where the prior probabilities of individual classes are also considered. The *updateScore* method iteratively updates the weights (line 16), and is in this work for the j-th feature and i-th sample defined as follows:

$$w[j] \mathrel{+}= \underbrace{\left| \boldsymbol{r}_i^j - \mathbb{E}[\text{nearestNeighbors}(i)][j] \right|}_{\text{absMean weight update}} \cdot \underbrace{\begin{cases} P[c]/(1-P[c_i]) & ; c \neq c_i \\ -1 & ; c = c_i \end{cases}}_{\text{Prior information}} .$$

In the proposed update step, an instance is compared directly to the mean of its neighbors which reduces the noise compared to the original updates of Relief (Kira and Rendell 1992) where the order of the averaging and $|\cdot|$ operators is reversed.

The nearestNeighbors represents the ordered set of indices of the top k neighbors, thus $\mathbb{E}[\text{nearestNeighbors}(i)][j]$ represents the first moment w.r.t. the j-th feature based on the nearest neighbors of the i-th instance (there are k such neighbors). We set the prior to -1 and the offset for considering the nearest neighbors to +1 if $c = c_i$, i.e., the considered class c_i is equal to the currently considered class c.

3.7.2 Multi-label classification

For multi-label classification (MLC option), the weight update step differs substantially from the MCC case, after selecting a random instance ($\boldsymbol{r}_i$) and determining its distance to the other examples. First, the indices of the closest k neighbors are stored in nearestNeighbors. As the values in the target space $\boldsymbol{T}$ are sets of (multiple) labels per instance, the simple iteration considered in the MCC case does not take the interactions between classes into account (label co-occurrences). Hence, distances are also computed between the target values of $\boldsymbol{r}_i$ and its nearest neighbors (line 26), by using one of the implemented options of ReliefE, which are given in Table 1.

Note that we also consider the cosine and hyperbolic distances which are applicable if the label space *is embedded* prior to the ranking step. We believe employment of manifold projections that operate on sparse spaces can be of relevance for high-dimensional output spaces, as for example seen when considering gene function prediction (Urbanowicz et al. 2018). Once the distances are obtained both based on the input space and the output space,

Table 1 Considered distances between rows in the multi-label output matrix $\boldsymbol{T}$

Distance	Definition
F1	$\text{dist} = \begin{cases} 1 - \frac{2\boldsymbol{t}_1\boldsymbol{t}_2^T}{(\text{nnz}(\boldsymbol{t}_1)+\text{nnz}(\boldsymbol{t}_2))} & ;(\text{nnz}(\boldsymbol{t}_1)+\text{nnz}(\boldsymbol{t}_2))>0 \\ 0 & ;\text{otherwise} \end{cases}$
Accuracy	$\text{dist} = \begin{cases} 1 - \frac{\boldsymbol{t}_1\boldsymbol{t}_2^T}{(\text{nnz}(\boldsymbol{t}_1+\boldsymbol{t}_2))} & ;(\text{nnz}(\boldsymbol{t}_1+\boldsymbol{t}_2))>0 \\ 0 & ;\text{otherwise} \end{cases}$
Subset	$\text{dist} = \begin{cases} 1 & ;\boldsymbol{t}_1 == \boldsymbol{t}_2; \\ 0 & ;\text{otherwise} \end{cases}$
Hamming	$\text{dist} = \sum_{i=1}^{\lvert\boldsymbol{t}_1\rvert} \lvert\boldsymbol{t}_{1,i} - \boldsymbol{t}_{2,i}\rvert / \lvert\boldsymbol{t}_1\rvert$
Cosine (if embedded)	$\text{dist} = \boldsymbol{t}_1\boldsymbol{t}_2^T / (\lVert\boldsymbol{t}_1\rVert_2 \lVert\boldsymbol{t}_2\rVert_2)$
Hyperbolic (if embedded)	$\text{dist} = \text{arcCosh}\,(-\boldsymbol{t}_1\boldsymbol{t}_2^T)$

The $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ correspond to two rows, the nnz represents the count of non-zero elements in a given row vector. Note that the considered vectors are binary in all but the cosine and hyperbolic cases (last two rows)

this information is used for updating feature weights as follows. Let K represent the set of considered nearest neighbors. Let T-DIFF represent the mean of the target distances TAR-DIFF. Let D-DIFF represent the mean of the (descriptive) distances to neighbors (DES-DIFF), as also considered for the MLC case, i.e. the absolute difference between the selected instance $\boldsymbol{r}_i$ and the mean of the nearest neighbors. More formally

$$\text{T-DIFF} = \mathbb{E}[\text{dist}(\boldsymbol{t}_i, \boldsymbol{T}[\boldsymbol{n} \in K])] \quad \text{and} \quad \text{D-DIFF} = \mathbb{E}[\text{dist}(\boldsymbol{r}_i, \boldsymbol{E}[\boldsymbol{n} \in K])],$$

where $\boldsymbol{X}[\boldsymbol{n} \in K]$ keeps only the rows of the matrix $\boldsymbol{X}$ that correspond to the neighbors $\boldsymbol{n} \in K$.

We further define

$$\text{TD-DIFF} = \mathbb{E}[\text{TAR-DIFF} \odot \text{DES-DIFF}],$$

and the weight update can be defined as:

$$w[j] \mathrel{+}= \frac{\text{TD-DIFF}}{\text{T-DIFF}} - \frac{\text{D-DIFF} - \text{T-DIFF}}{1 - \text{TD-DIFF}}. \tag{1}$$

The weight update concludes the ranking for multi-label classification.

3.8 Theoretical analysis

We next discuss the relevant theoretical aspects of ReliefE, ranging from computational complexity analysis (both time and space) to the implications of the adaptations considered.

3.8.1 Time complexity

The time complexity of ReliefE can be studied with respect to the two main modes of function—multi-class and multi-label classification.

Dimensionality estimation. The dimensionality estimation step, as optionally considered in this work, requires pairwise distance computation between the instances. Thus, $\Theta(|\nu|^2 \cdot |F|)$ operations are required, where ν represents the indices of the samples for dimension estimation, and $|\nu| < |I|$, as discussed in Sect. 3.5. Note that if all instances are considered, the complexity rises to $\Theta(|I|^2 \cdot |F|)$.

Manifold projections. Learning low-dimensional representations represents one of the computationally more intensive parts of ReliefE. Following (McInnes et al. 2018b), the UMAP's complexity can be split into two main parts. First, approximate nearest neighbor computation was shown to have empirical complexity of $\mathcal{O}(|I|^{1.12} \cdot |F|)$ (Dong et al. 2011). However, in the sampling limit, if all instances are considered, the computational complexity is equal to that of pairwise comparisons—$\mathcal{O}(|I|^2 \cdot |F|)$ The optimization of embeddings requires additional $\mathcal{O}(k \cdot |I|)$ steps, where k is the number of nearest neighbors (a hyperparameter). Overall complexity is in the worst case thus $\mathcal{O}(|I|^2 \cdot |F|)$. Note that the proposed cyclic sampling scheme (Sect. 3.5) implies $I \to \nu$ for all cases in this paragraph.

Multi-class. Given a fixed number of samples s, ReliefE traverses each of the classes |T|, and for each one performs the sampling. The adaptive neighbor selection scheme does not cost any additional time w.r.t. |I|, as the distances are already computed. The feature update step requires $\mathcal{O}(|F|)$ operations, for each neighbor. The complexity of the original, re-implemented ReliefF is thus $\mathcal{O}(|I| \cdot |F| \cdot s)$ (Robnik-Šikonja and Kononenko 2003). The absMean update does not change this complexity, however, when adaptive scoring is considered, distances to the class members need to be sorted. We re-use the sorted indices of top neighbors to obtain closest distances, thus no additional time is spent on sorting. If ReliefE is considered, the complexity needs to be adapted for input dimension estimation, as well as lower dimension in which distances are computed. The final complexity is thus: $\mathcal{O}(|\nu|^2 \cdot |F| + |I| \cdot d \cdot s)$, where d is the dimensionality of the embedding. Assuming the "empirical complexity" from the previous paragraph holds, the multi-class complexity can also be stated as $\mathcal{O}(|\nu|^{1.12} \cdot |F| + |I| \cdot d \cdot s)$

Multi-label. The complexity of multi-label classification needs to additionally account for the distances computed between the target instances. Effectively, $\mathcal{O}(|\nu|^2|F| + s \cdot (|I| \cdot d_F + k \cdot d_T))$ operations are required, where d_T and d_F correspond to embedding dimensions of the input and output space – for each sample, first distances need to be computed between the instances to that sample within the input space (|I|). Once top k nearest instances are identified, the distances of the target

	instance to these k other target instances are computed in d_T space.
Down-stream ranking.	Commonly, Relief algorithms operate in the raw feature space, however, as ReliefE operates via embedding-based distance computation, we consider the option that embeddings are *pre-computed*. This is possible due the fact that many contemporary embedding algorithms *refine* the representation, once the new data is obtained, and do not (necessarily) re-compute the embedding for each new instance. In this case, the initial complexity of $\mathcal{O}(\lvert\nu\rvert^2 \cdot \lvert F\rvert + \lvert I\rvert \cdot d \cdot s)$ reduces to $\mathcal{O}(\lvert I\rvert \cdot d \cdot s)$.

3.8.2 Space complexity

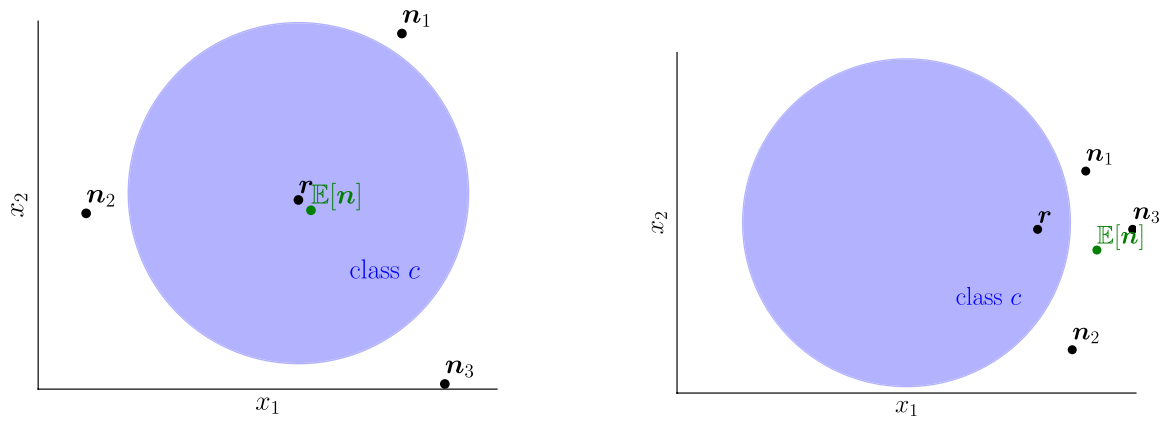
The proposed implementation of ReliefE in comparison with, e.g., state-of-the-art Python-based implementations (as found in Urbanowicz et al. 2018) operates easily in very high-dimensional, sparse vector spaces. In practice, we adopt the CSR formalism for matrix representation. Here, a sparse matrix is stored as three arrays, a data array, a pointer array and an index array. All three code for non-zero entries in the input space. Note that such representation is not optimal for dense matrices, as it results in some (minor) space overhead. This design decision means that every computationally-intensive operation that is part of ReliefE was re-written with handcrafted CSR-friendly, Numba-compilable methods. More formally, let nnz correspond to the number of non-zero elements in a given matrix. The space complexity of ReliefE can thus be stated as: $\mathcal{O}(\max(\text{nnz}(\boldsymbol{F}), \text{nnz}(\boldsymbol{E_F})) + \max(\text{nnz}(\boldsymbol{T}), \text{nnz}(\boldsymbol{E_T})))$.

The complexity thus depends on the relation between the embedded space and the input space, which can be very context-dependent, however very low-dimensional embeddings normally do not result in space overhead, and neither do highly sparse input matrices. More formally, if $\text{nnz}(\boldsymbol{F}) \geq |I| \cdot d$ or $\text{nnz}(\boldsymbol{T}) \geq |I| \cdot d$,

the embedded space will require less (or equal) memory. Note that $\boldsymbol{T}$, corresponding to a potentially very sparse output space, is similarly considered as a sparse matrix, meaning that classification problems with very high-dimensional target spaces can also be considered, which is to our knowledge one of the first such Python-based, user-friendly implementations. As dimensionality estimation only requires the two closest neighbors, we do not keep all others in memory, the space complexity becomes linear, i.e., $\mathcal{O}(|I|)$ (in fact, exactly $2 \cdot |I|$). We empirically discovered that UMAP's memory requirements are the main space bottleneck, and, based on the evaluation on the larger data sets, require $\mathcal{O}(|I|^2)$ (empirical) space. Such complexity potentially arises from the dense computational graph derived by UMAP. This observation led us to introduce the representative (cyclic) sampling scheme, which reduced this complexity to $\mathcal{O}(|\nu|^2)$, making ReliefE executable even on an off-the-shelf computer (Lenovo Carbon X1). Note that the number of samples is lower-bounded by the number of classes or unique label sets.

3.8.3 absMean update step and its implications

Compared to the original ReliefF, one of the proposed modifications implemented in ReliefE is the comparison of a given instance directly to the average nearest neighbor. We believe that this approach is advantageous in two ways.



(a) r is in the center of its class **(b)** r is on the border of its class

Fig. 3 Updating weights with the absMean approach. The $n_{1,2,3}$ represent the instance **r**'s neighbors

First, as shown in Fig. 3a, if a sample $\boldsymbol{r}$ that is far away from the class border is chosen, we cannot capture the local structure of the data in the other classes, so such samples $\boldsymbol{r}$ should not influence the updates considerably. This is not the case in the standard ReliefF, since the differences in feature values are necessarily large. This is overcome by computing the average neighbor first, and then updating the weights.

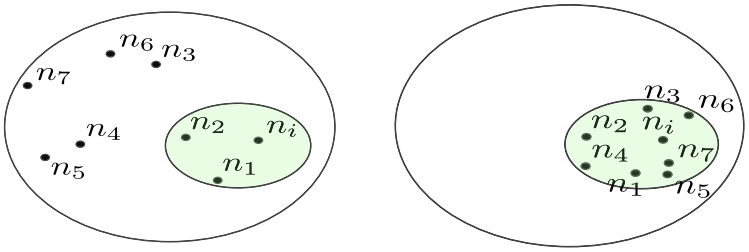
Second, when the sample $\boldsymbol{r}$ is close to the border (Fig. 3b), averaging neighbors results in correctly detecting that the general direction of the neighbors should be perpendicular to the class borders when the number of samples goes to infinity. For example, in the situation depicted in Fig. 3b, only n_1 should be rewarded. Again, computing the mean neighbor $\mathbb{E}(\boldsymbol{n})$ first, brings us closer to the optimal direction. The reduction of noise can be also proven by using the triangle inequality, $\frac{1}{k}\sum_{j=1}^{k}|n_i^0 - n_i^j| \geq |n_i^0 - \frac{1}{k}\sum_{j=1}^{k} n_i^j|$, from which it directly follows that this approach results in smaller weight updates.

3.8.4 Adaptive neighbor selection and its behavior

The considered adaptive neighbor selection attempts to reduce the number of hyperparameters by one (k), potentially saving $\mathcal{O}(k)$ optimization iterations, should this parameter be tuned. Furthermore, by considering neighbors, potentially relevant for a given instance, less noise is considered during the weight update step. For example, assume $k = 7$, with only three other instances very close and the remaining four much further (by a large margin). The latter 4 instances will impact the weight update significantly, as the average distance will be heavily biased towards their mean, and thus potentially not representative of the close neighborhood of a given instance that naturally appears in the data. A visualization in such a situation is shown in Fig. 4.

In both panels ((a) and (b)), the outer circle represents the neighbourhood for a hard-coded value of k. In Fig. 4a, very distant instances are also considered for the update (e.g., from n_3 onward) and the adaptive estimation only selects the closest neighbors (green). However, in Fig. 4b, all instances are very close, thus the value of k is equal to the automatically selected choice.

This follows the intuition behind the Relief family of algorithms, where an instance is compared to its slight perturbations. Another downside of having k fixed, is that taking into account more distant nearest neighbors would (on average), increase the importance of more



(a) Dispersed neighborhood **(b)** Local neighborhood

Fig. 4 Adaptive neighbor selection. The $n_{1,2,3,\ldots}$ represent neighbors

noisy features, since the distance values directly influence the importance. Irregularities in distance distribution were shown to hold for many real-life data sets, see for example the assumptions and their implementation in Dong et al. (2011). Finally, as ReliefE operates in a latent, low-dimensional space, obtained by instance similarity comparison, comparison to the closest instances only is potentially meaningful.

3.8.5 Parallelism aspects

The proposed implementation exploits the Numba framework for just-in-time compilation (Lam et al. 2015). Numba offers parallelism at the level of individual methods that get compiled, meaning that the proposed implementation offers parallelism at the level of weight updates. During compilation, parts of the code that are sensible to compile get detected *automatically*. Many operations such as scalar-vector addition and similar can easily be parallelized. With auto-parallelization, Numba attempts to identify such operations in the ReliefE weight update step, and fuses adjacent ones together, to form one or more kernels that are automatically run in parallel. In practice, we observed that such auto-parallelism does not necessarily offer superior performance in terms of speed. However, it represents an elegant, array-level parallelism detection which, when improved/updated, shall speed up the execution time even more. We omit the discussion regarding different spaces considered during ReliefE to "Appendix 1".

3.8.6 How powerful is ReliefE?

Throughout this paper, we propose and demonstrate the utility of ReliefE when tabular data is considered. However, as ReliefE requires merely the *representations* of instances (training or target), the proposed approach generalizes well beyond tabular data with a single adaptation: the embedding method needs to be suitable for the considered data type. For example, if an instance is described by an ordered list of graphs, the plethora of graph embedding methods (Goyal and Ferrara 2018; Mežnar et al. 2020) could be used to prioritize the graphs based on their (learned) representations. Similarly, ReliefE could be adapted for learning in the context of relational data bases, via Wordification (Perovšek et al. 2015) and other propositionalization-like algorithms.

Table 2 The properties of the considered multi-class classification data sets

Data set	Instances	Features	Classes	Proportion of non-zero entries
chess (Shapiro 1984)	3196	38	2	0.726558
biodeg-p2-discrete (Džeroski et al. 1999)	328	61	4	0.107357
optdigits (Alpaydin and Kaynak 1998)	5620	62	10	0.528117
madelon (Guyon et al. 2005)	2000	500	2	0.999999
php88ZB4Q (Anguita et al. 2013)	10299	561	6	0.999860
pd_speech_features (Sakar et al. 2013)	756	753	2	0.995294
dlbcl (Armstrong et al. 2002)	77	7070	2	0.997388
tumors C (Pomeroy et al. 2002)	60	7129	2	0.995722
AP_Ovary_Kidney (Stiglic and Kokol 2010)	458	10935	2	1.000000
ohscal.wc (Han and Karypis 2000)	11162	11465	10	0.005270
genes (Weinstein et al. 2013)	801	20531	5	0.857824

The last column denotes the proportion of non-zero elements in the data table

4 Empirical evaluation setting

Our empirical evaluation of ReliefE consists of many sub-studies, and can be summarized as follows. First, we discuss the evaluation of ReliefE against state-of-the-art ranking algorithms on eight multi-class classification data sets. Next, we present the empirical evaluation setup where ReliefE's capabilities are shown on nine multi-label classification data sets. Finally, we conducted a series of experiments where we investigated in more detail the convergence and time performance. We conclude this section by describing the Bayesian and frequentist approaches, to aid understanding of the results.

4.1 Multi-class classification data sets

Multi-class classification remains one of the most widely adopted forms of learning. Here, the input space is associated with a single, integer-valued vector, where each instance can belong to one of the many possible classes. In this work, we consider a wide spectrum of data sets, summarized in Table 2.

The data sets are from multiple domains, incl. biological, social and other domains (e.g., chess). The data sets are of different dimensions, in terms of the numbers of rows and also columns.

4.2 Multi-label classification data sets

Feature ranking for multi-label classification remains an active research area. Many of the approaches considered in the previous section (multi-class classification) are not able to handle the multi-label setting, where a single instance can belong to many classes simultaneously. Such a setting, for example, naturally emerges when gene

Table 3 The properties of the considered MLC data sets

Data set	Instances	Features	Classes	Proportion of non-zero entries
delicious (Tsoumakas et al. 2008)	16105	1000	983	0.500000
imdb (imdb dataset 2010)	120919	1001	28	0.019363
medical (Pestian et al. 2007)	978	2898	45	0.500000
bibtex (Katakis et al. 2008)	7395	3672	159	0.500000
Education1 (Ueda and Saito 2003)	12030	27534	33	0.004059
Health1 (Ueda and Saito 2003)	9205	30605	32	0.003555
Entertainment1 (Ueda and Saito 2003)	12730	32001	21	0.004552
Science1 (Ueda and Saito 2003)	6428	37187	40	0.004659
Social1 (Ueda and Saito 2003)	12111	52350	39	0.002949

The last column denotes the proportion of non-zero elements in the data table

function prediction is considered—a single gene is associated with many functions and contexts. The considered multi-label data sets are summarized in Table 3.

Similarly to the multi-class setting, we selected data sets from various domains to maintain diversity. Note that multiple repetitions of 10 fold cross validation were needed to perform Bayesian comparisons.

4.3 Additional experiments and statistical evaluation of results

For MCC, logistic regression with its default parameters was used as the learner. The first reason for this choice is the fact that this very learner is commonly used to evaluate the quality of a given data representation (in our case a subset of the feature space), and is known to be sensitive to noisy features. The second reason is computational: With all repetitions required for Bayesian analysis, additional grid search would be out of reach as it could further increase the computational time beyond reasonable capabilities. For the MLC setting, the default random forest parametrization was used, as it has been previously shown to perform competitively/well in such setting. Throughout the experiments, we set the regularization term of logistic regression (C) to one, the default value (Pedregosa et al. 2011). For multi-label classification, we considered the RandomForest classifier with default settings as set in Pedregosa et al. (2011).

As we consider either multi-class or multi-label problems, we compute either *relative* F1 or micro-averaged relative F1 scores, defined as:

$$\mathrm{rF1}(f) = \frac{\mathrm{F1}_f}{\mathrm{F1}_{f=|F|}},$$

where F1 is the harmonic mean of precision and recall, and f is the number of features. The macro rF1 is defined in the same fashion. Considering relative performance offers direct insights into how performant a given ranking is with how many top-ranked features. Note that by considering relative performance, it can be directly observed when the feature ranking algorithm identifies a ranking that outperforms the situation where all features are

considered—a reasonable baseline. We performed ten fold stratified cross-validation ten times, as required for the statistical analysis discussed next.

In order to summarize the overall performance of a given ranking, we believe taking into account the ranking's quality over all possible values of top f features needs to be considered. Hence, we introduce the area under rF1 (AUrF1), i.e., the integral of rF1 normalized by the number of considered top f rankings (to be more comparative across data sets), where we numerically integrate with the Simpson's method.

Recent criticisms of the frequentist non-parametric comparison of multiple classifiers (Demšar 2006) has given rise to a novel spectrum of Bayesian t-tests, that directly offer insight into a probability space corresponding to the differences in algorithm performance (Benavoli et al. 2017). In this work we adopt the hierarchical t-test, which is capable of comparing pairs of classifiers. The hierarchical Bayesian t-test is used to assess the probability of observing a given difference in performance between a pair of classifiers. As noted by Benavoli et al. (2017), it requires that e.g., ten repetitions of ten fold cross validation need to be considered in order to reliably fit a hierarchical model. The approach attempts to model the probability of observing a given difference in performance between a pair of classifiers, which can be in favor of either of the classifiers or undetermined—practically equivalent (rope region). The plotted results are given in the form of triangular schemes, where each point represents a sample from the posterior distribution. Such samples, when aggregated, directly represent a probability of observing a given state (in this case difference between the classifiers). We set the rope region to 5%—if the difference in quality between two rankings is less than 5%, they are considered equal. The remaining setting is the same as in the original paper (Benavoli et al. 2017), we compare the top ranking for each fold. For a given pair of ranking algorithms, the pairwise Bayesian tests were performed on the data sets common to both algorithms. Finally, results of time performance are presented in computation time (in seconds) diagrams with standard deviations. Such a comparison is not necessarily informative/useful when multiple classifiers are simultaneously considered, thus we also offer the results in the form of average rank diagrams (Demšar 2006). We believe that having both local and global insights into the relations between classifiers their differences are easier to study, even though looking at the classifier ranks alone can be misleading (Benavoli et al. 2017).

4.4 Considered implementations and baselines

We next discuss the implementations considered. For multi-class classification, the considered Relief variants were MultiSURF, MultiSURFstar, ReliefF, all from the scikit-rebate library (Urbanowicz et al. 2018). We also used RandomForest (RF)-based importances (Genie3) and Mutual information (MI)-based ones (Pedregosa et al. 2011). The multi-class Relief variants that are the original contribution of this work include: ReliefE, ReliefE-absMean, ReliefE-adaptive and ReliefE-absMean-adaptive. The suffix *adaptive* denotes the use of an adaptive threshold and *absMean* the use of absMean update step.

Multi-label classification is not supported (at all) in scikit-rebate (Urbanowicz et al. 2018), and thus we considered the multi-label variants of ReliefE and ReliefF (re-implemented in this work with Numba) with all of the possible distances given in Table 1. We emphasize that when multi-label distances are considered, only the cosine and hyperbolic

Table 4 Computational complexity of feature importance estimation

Algorithm	Time complexity	Space complexity
ReliefE	$\mathcal{O}(\lvert\nu\rvert^2 \cdot \lvert F\rvert + \lvert I\rvert \cdot d \cdot s)$	$\mathcal{O}(\lvert\nu\rvert^2)$
ReliefF	$\mathcal{O}(\lvert F\rvert \cdot \lvert I\rvert^2)$	$\mathcal{O}(\lvert F\rvert)$
Random Forest	$\mathcal{O}(t \cdot \lvert F\rvert \cdot \lvert I\rvert \cdot \log^2 \lvert I\rvert)$	$\mathcal{O}(\lvert I\rvert + \lvert F\rvert)$
Mutual Information	$\mathcal{O}(\lvert F\rvert \cdot \lvert I\rvert)$	$\mathcal{O}(\lvert I\rvert)$

For Relief algorithms, we used $s = \lvert I\rvert$. The t corresponds to the number of trees

distances operate on target space embeddings (the other distances do not). The computation of these distances is also more efficient.

Note that all versions of ReliefF, implemented or re-implemented in this work,[3] natively operate on sparse spaces, which is on its own a contribution of this work. In terms of sparsification, we set the sparsification threshold to 0.15, meaning that if a matrix's density is higher than 15%, it is sparsified with the proposed procedure (there are many of such matrices amongst the considered data sets). Detailed results of investigating the ablation of the considered data sets' (induced) sparsities are given in "Appendix 2". Similarly, the behavior of the adaptive k statistic was also studied in more detail in "Appendix 3". Further, ν (the sample for intrinsic dimension estimation) was set to 2048. The dimension number was set so that the algorithm runs normally on an off-the-shelf-computer (Lenovo Carbon X1) even for larger data sets. Thus, if a given data set consisted of more than 2048 instances, a representative subset of 2048 instances was considered for estimating the intrinsic dimension and consequent embedding. The UMAP's setting is left to its defaults, with the dimension being set to the estimated one.[4] The value of k is set to 15 for our implementation for ReliefF, and left at its defaults for the baselines. The time and space complexity of the baselines and ReliefE are summarized in Table 4. Note that, even though ReliefF (and its other variants') space complexity is linear w.r.t $|F|$, their implementations, should they not consider the sparse input structure, in fact require $\mathcal{O}(|I| \cdot |F|)$ space (as found, e.g., in Urbanowicz et al. 2018).

Finally, the considered experiments for multi-label classification consider both Euclidean embeddings, as well as non-Euclidean ones (Poincaré ball).

5 Results

This section presents the results of the empirical evaluation. We begin by discussing the performance comparisons for the task of multi-class classification. We follow on by discussing the results of the experiments on multi-label classification tasks. Finally, we present additional investigations of ReliefE's behavior.

[3] Implementation's official repository is https://github.com/SkBlaz/reliefe.

[4] Extensive evaluation of UMAP's capabilities w.r.t. the proposed implementations is beyond the scope of this paper, and is left for further work.

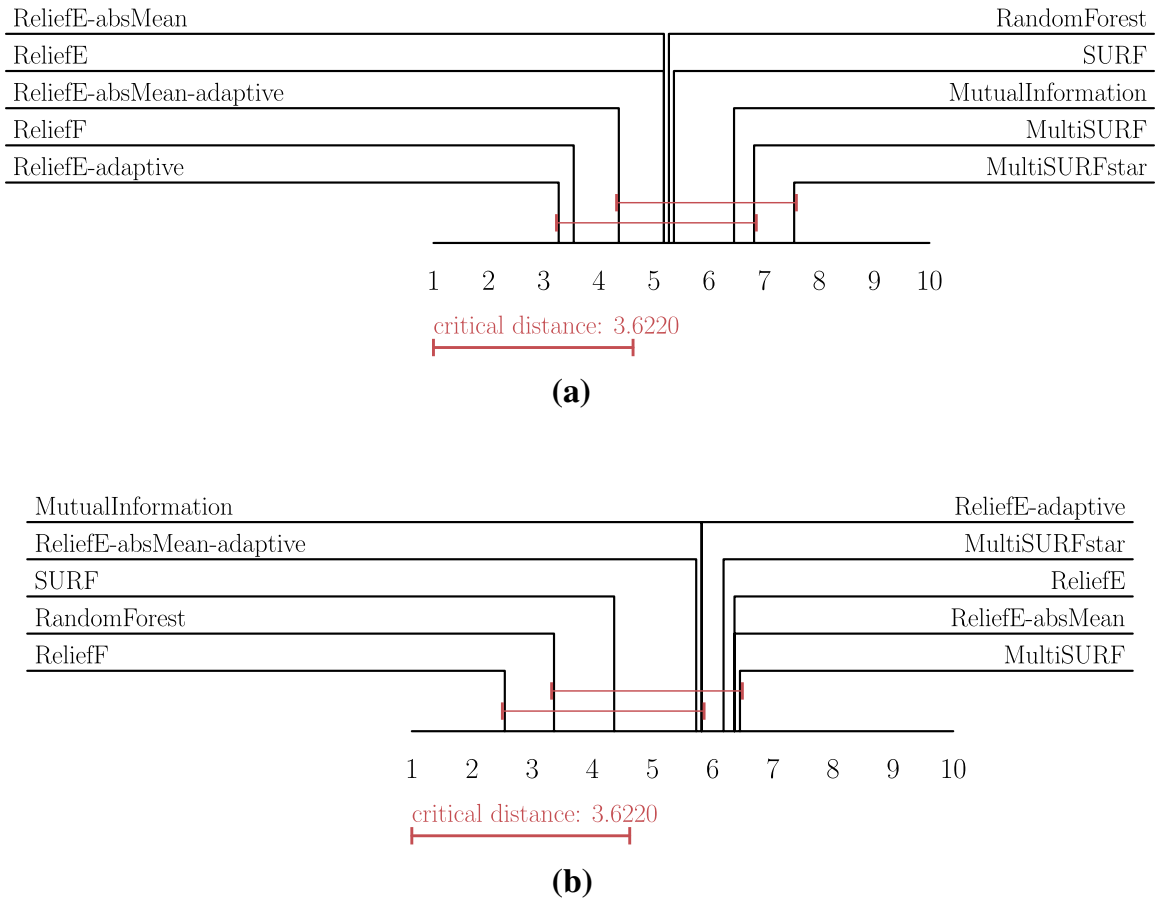


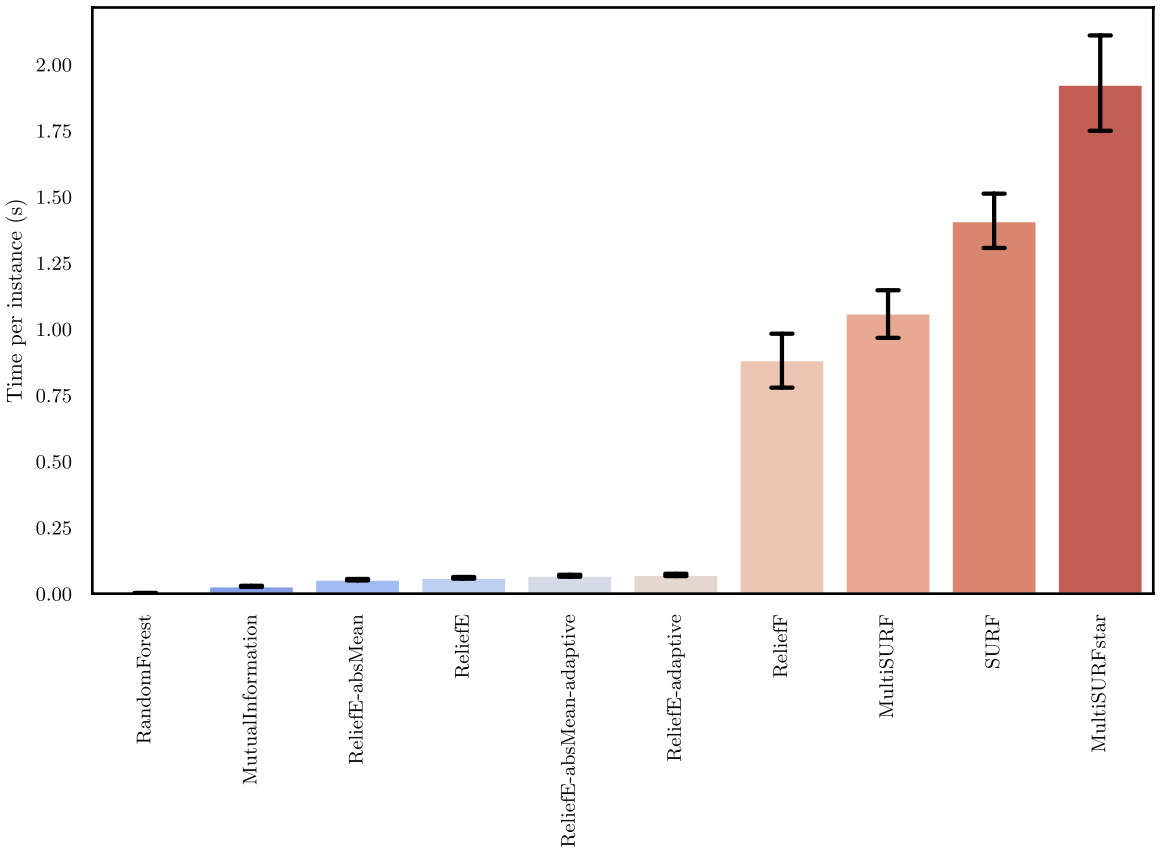
Fig. 5 Max (**a**) and mean (**b**) F1 scores across all feature rankings

5.1 Multi-class classification

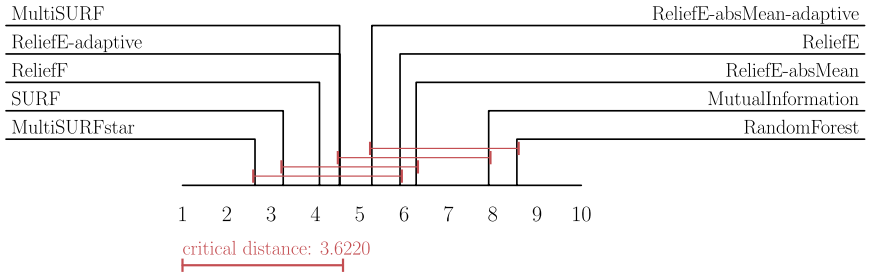
We first present two average rank diagrams depicting the relative performance on the different ranking methods for MCC in terms of the quality of the produced rankings, as measured by the corresponding average and maximum F1 scores (Fig. 5a, b , respectively). The diagrams include critical distances, representing the minimum differences in performance that are statistically significant. It can be observed that the ReliefE variants yield the best performing rankings (with lowest average ranks, Fig. 5a), but there are not many such rankings (Fig. 5b). The AUrF1 values ("Appendix 4") indicate that the performances of the top 5 feature ranking algorithms are highly similar (within the confidence interval).

We next present the mean time consumption averaged across data sets. Consistently slower SURF variants of ReliefF can be observed in the rightmost part of Fig. 6a. The average rank diagram is shown in Fig. 6b.

Additional analysis of the proportions of time spent at different parts of the algorithm is presented in "Appendix 5", showing that most time is spent on feature weight updates. Average rank diagrams comparing the rankings in terms of the top 50 and 100 features are given in "Appendix 6".



(a) Average absolute running times with standard deviations (in seconds).



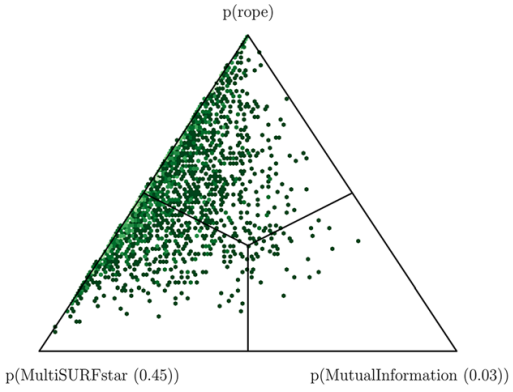
(b) Average rank diagram (times).

Fig. 6 Speed comparison of ranking approaches for MCC. Absolute running times (**a**) show that ReliefE variants perform an order of magnitude (or more) faster. Relative running times are given in terms of average ranks (**b**), where lower ranks mean worse performance, i.e., longer running times

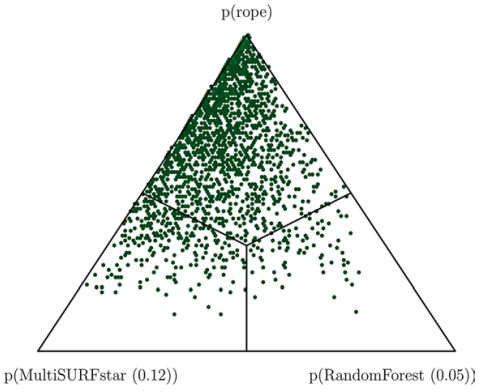
5.2 Bayesian ranking comparison of ranking approaches for MCC

In this section, we present selected Bayesian pairwise comparisons of classifiers' performance. Previously determined relationships, such as the dominance of the SURF branch of algorithms over mutual information were confirmed, and further extended by adding comparisons with the proposed ReliefE branch of algorithms. The comparisons are presented in Figs. 7 and 8.

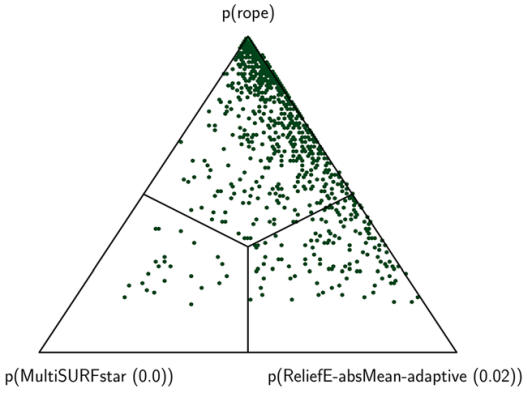
Each diagram has three main regions (parts of the pyramid). The two bottom regions correspond to the samples associated with the dominance of each of the two algorithms



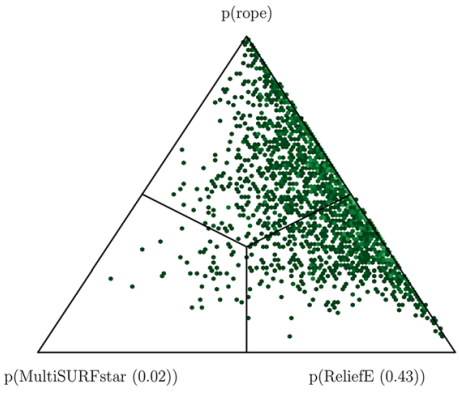
(a) MultiSURFstar vs. MI



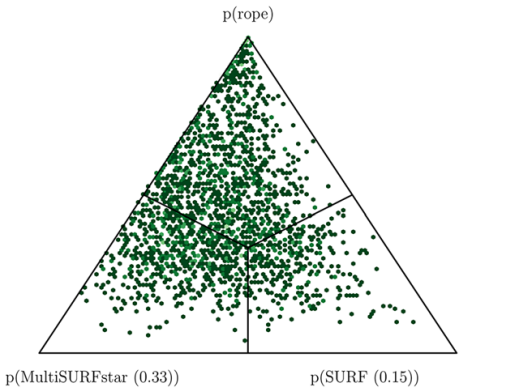
(b) MultiSURFStar vs. RF



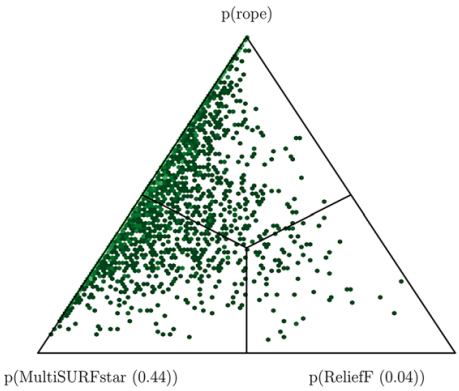
(c) MultiSURFstar vs. ReliefE-absMean-adaptive



(d) MultiSURFstar vs. ReliefE

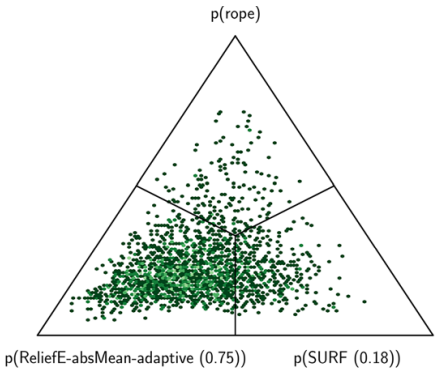


(e) MultiSURFstar vs. SURF

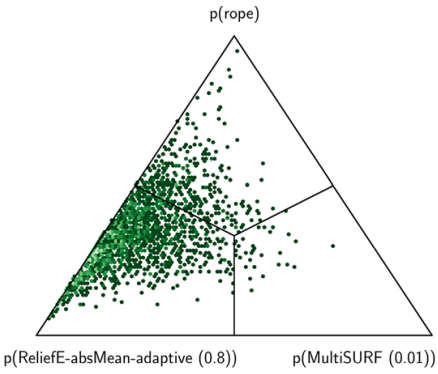


(f) MultiSURFstar vs. ReliefF

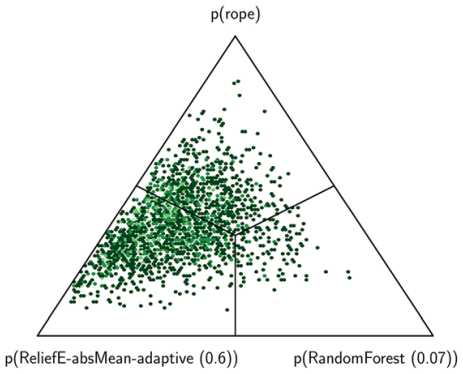
Fig. 7 Bayesian comparisons of performance (ranking qualities) between MultiSURFstar and other feature ranking methods for MCC



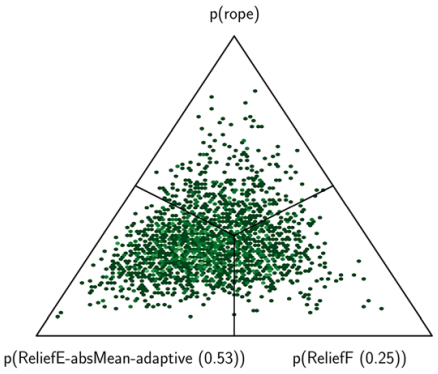
(a) ReliefE-absMean-adaptive vs. SURF



(b) ReliefE-absMean-adaptive vs. MultiSURF



(c) ReliefE-absMean-adaptive vs. RandomForest



(d) ReliefE-absMean-adaptive vs. ReliefF

Fig. 8 Bayesian comparisons of performance (ranking qualities) between ReliefE-absMean-adaptive and other feature ranking methods for MCC

compared, and the rope region to the difference space, where the winner is not clearly defined. The probability density directly corresponds to the density of dots in the diagram, thus, the part of the diagram with the highest density implies the most probable situation. Individual (posterior) probabilities are also shown next to each diagram, and denote the probabilities of one algorithm outperforming the other or the algorithms being of similar performance.

The key results of such pairwise comparisons can be summarized as follows. Very few comparisons yield clear winners. In the majority of the cases, when the most competitive methods are considered, less than 50% probability that one of the ranking algorithms dominates is observed, giving no strong evidence for dominant ranking algorithms. This is the case also for the diagrams in Fig. 7.

The visualizations in Fig. 8 show that ReliefE-absMean-adaptive, the implementation proposed in this work, performs on par, or better than many existing, well established approaches such as MultiSURF and RandomForest-based rankings. However, we observe, in the second part of Fig. 8, that ReliefE-absMean-adaptive offers small, albeit incremental win rate when compared against the other methods. With the highest probability (80%), we can claim ReliefE's dominance against MultiSURF, however, the observed probability

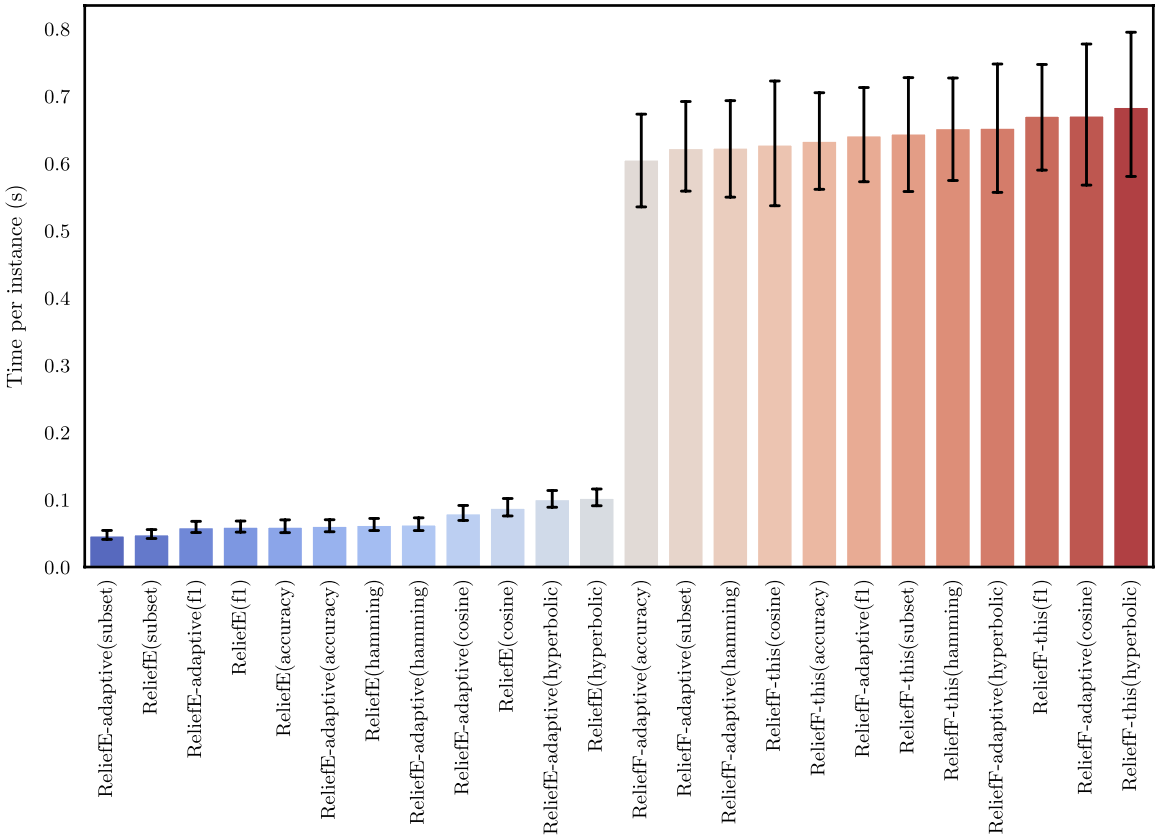


Fig. 9 Running times for the MLC variants of ReliefE (and ReliefF, reimplemented in this work and denoted ReliefF-this)

ratio does not suffice for a significant difference with $> 95\%$ probability (the commonly considered convention). To further study the algorithm performance, we visualize the top f features—rF1 curves and discuss the selected examples—such figures showing in detail the ranking performance of the different algorithms for the selected data sets are given in "Appendix 7". Overall, considering the different statistical approaches to evaluating ReliefE's performance, the results indicate that the method has similar performance to its competitors, but offers up to two orders of magnitude faster ranking computation, which also confirms the theoretical findings from Sect. 3.8.

5.3 Multi-label classification

We next present the results of feature ranking for multi-label classification. For readability purposes, we present the average rank diagrams in "Appendix 8". The time required for the execution of various distance-ranking algorithm combinations is shown in Fig. 9.

The differences in the execution times are apparent. The ReliefE branch (blue) offers more than an order of magnitude faster ranking computation.

The AUrF1 scores, averaged across data sets are shown in Fig. 10.

The best performing ReliefF variants for multi-label classification do not embed the input space. However, the top performant variant employs Euclidean embeddings of the target space, where the distances are computed based on the cosine similarity score. This result indicates multi-label classification can benefit from embedding-based approaches. A

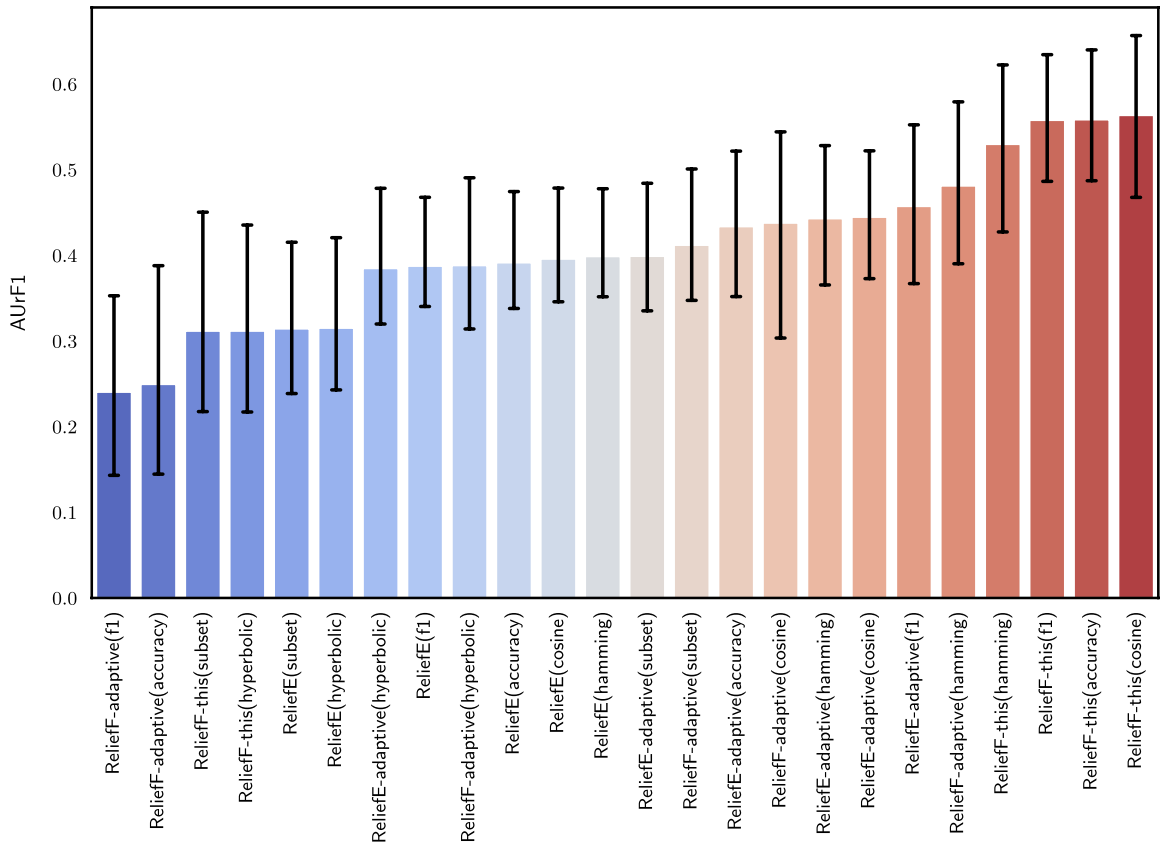


Fig. 10 Area under the relative F1 for different ranking approaches in the context of multi-label classification

case study, where the behavior of various ReliefE variants for MLC is considered in more detail can be found in "Appendix 9".

5.4 Relations between ranking algorithms

We employ the FUJI score, a recently introduced scale-free comparison of ordered positive real-valued lists, to study how different feature ranking algorithms relate to each other. This study employs the same methodology as discussed in Petković et al. (2020) and Škrlj et al. (2020). The considered FUJI scores can, apart from the ranking, also take into account the differences between the elements that are being compared—this is not possible by using, e.g., the Jaccard score. We compare pairs of curves comprised of (rF1,top f) tuples, thus effectively comparing the *shape* of the rankings' performance. The results of these comparisons are shown in Fig. 11 for multi-class classification and in Fig. 12 for multi-label classification. The most apparent pattern that emerges when these comparisons is that embedding-based rankings (ReliefE variants) tend to give very similar rankings. This holds for both multi-class and multi-label classification rankings.

5.5 Convergence to the final ranking

Note that in all the examples up to this point, the number of iterations via which the weights corresponding to feature importances were updated was equal to the number of instances (hence the quadratic complexity). Having shown that this setting already offers

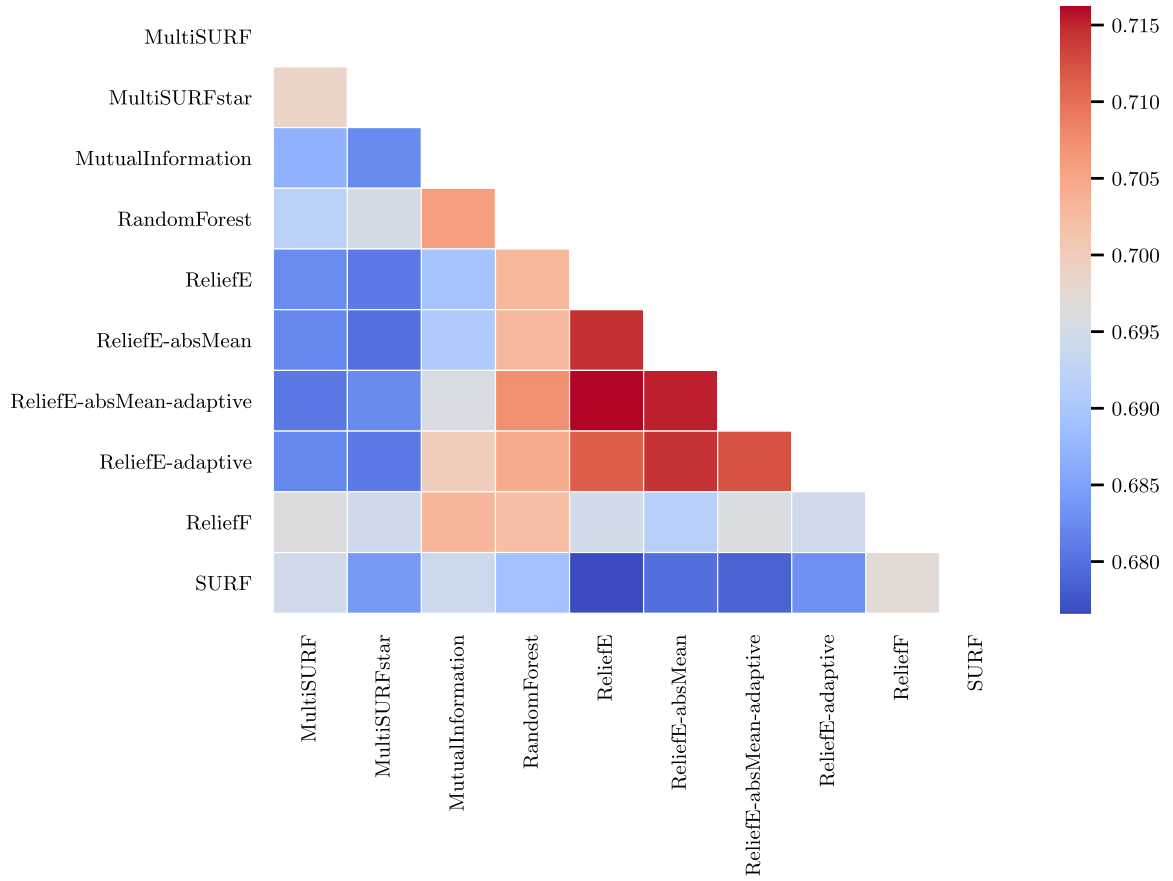


Fig. 11 AUFUJI scores for multi-class rankings. Higher numbers (red colors) mean higher similarity between rankings (Color figure online)

state-of-the-art performance, we further explored how redundant is the iteration process, i.e., what is the minimum number of iterations needed to obtain a similar ranking. We investigated this question on MCC datasets following the approach described below.

For each number of considered iterations, we conducted 100 logistic regression runs building models with up to 100 top-ranked features. We computed the AUrF1 and inspected the curve induced by the obtained series of (top f, AUrF1) tuples. We conducted these experiments for the DLBCL, Tumors C, Biodeg-discrete and chess data sets, with the results shown in Fig. 13. We compared ReliefE-absMean-adaptive with ReliefF as implemented in this work, evaluating each iteration with three-fold cross validation (same splits).

It can be observed in Fig. 13a that the convergence is slower with the ReliefE-absMean-adaptive variant, however, once the performance is achieved, it is no longer impacted by additional iterations. This does not appear to be the case with ReliefF, where a decrease is observed when 32 iterations are considered. Overall, ReliefE-absMean-adaptive offers state-of-the-art performance already after four iterations. A similar situation is observed in the case of Biodeg in Fig. 13c. We also observed that on the Tumors C data set (Fig. 13d), ReliefE-absMean-adaptive was consistently outperformed by ReliefF. Being very high-dimensional, and with only tens of instances, this data set's intrinsic dimension is most likely under-estimated, yielding feature ranking based on representations that loose too much information. The ReliefE branch of algorithms is highly dependent on the underlying embeddings, where construction of high quality embeddings in such data scarce scenarios remains a lively research area on its own.

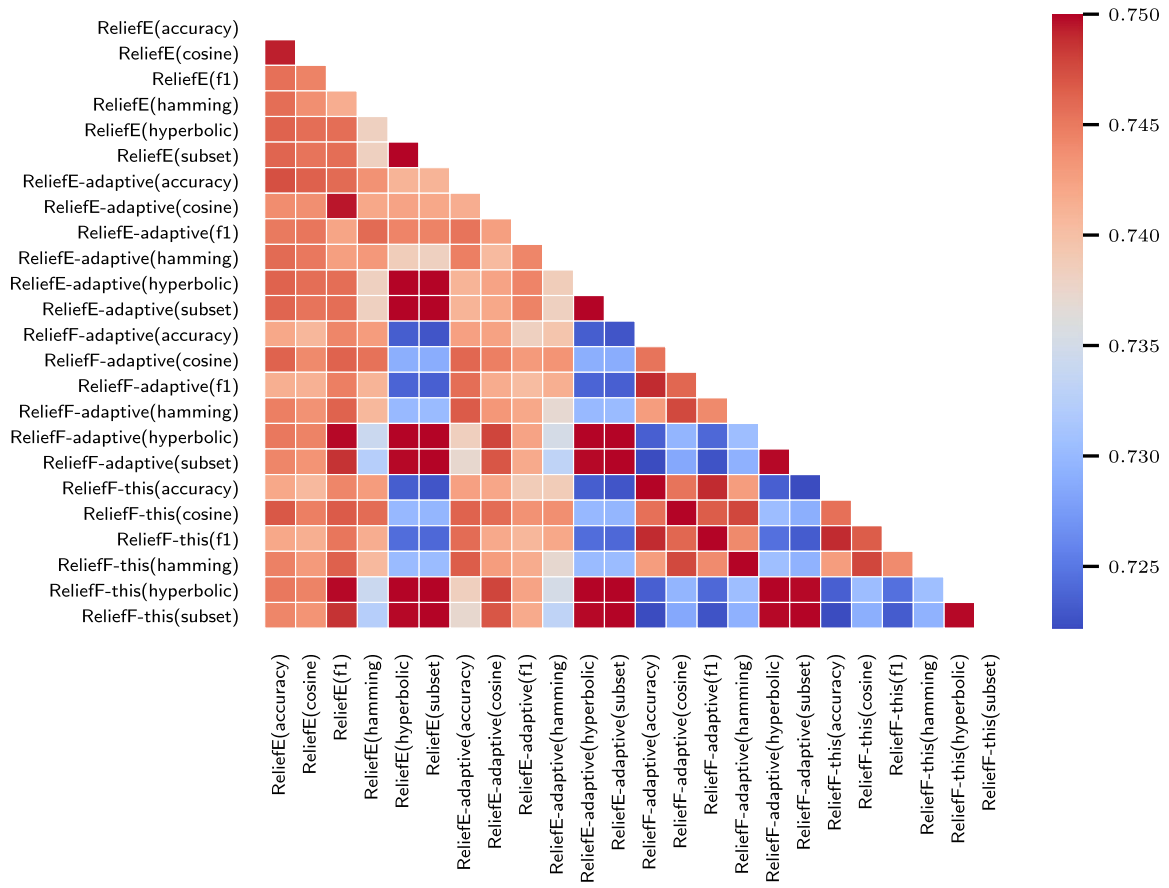
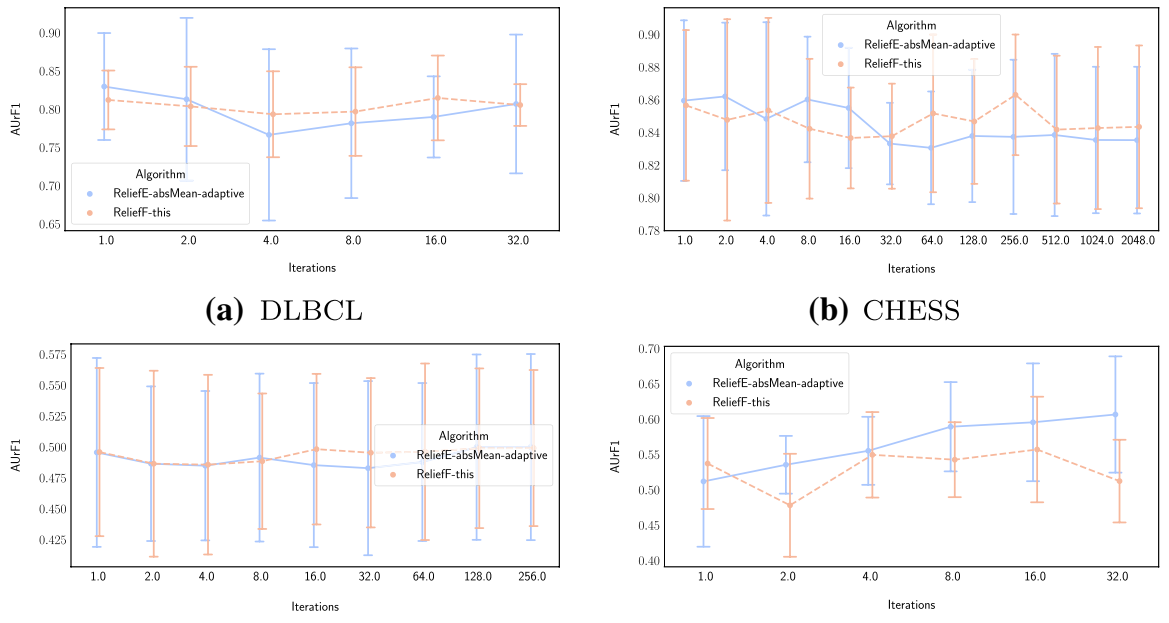


Fig. 12 AUFUJI scores for multi-label rankings. The red block of cells in the upper left part of the triangle corresponds to various variants of ReliefE (Color figure online)



(a) DLBCL **(b)** CHESS

(c) Biodeg-p2-discrete **(d)** Tumors C

Fig. 13 Impact of the number of ReliefF iterations on ranking quality

Potential speedups by decreasing the number of iterations will be explored in further work. The performance on the chess data set, however, remains consistent for both algorithms—this is a low-dimensional data set, where feature importance estimation via embedded space does not offer notable performance improvements, both with respect to top F1 and computation time.

5.6 Relevant negative results

Even though the paper proposes a promising Relief variant, capable of operating in high-dimensional sparse spaces, many intermediary steps did not perform as expected, and are summarized below:

1. Due to pointer-based storage, using sparse matrix algebra can result in additional overhead which can be significant in large, dense data sets.
2. Running UMAP with spectral decomposition resulted in an unexpected memory overhead. We circumvented this issue with ν, however, the original implementation, once adapted for large scale embedding, could offer an alternative that is more native to UMAP's routines.
3. Employment of Numba's parallel capabilities led to somewhat mixed results. On one hand, trivially parallel routines such as independent looping and similar could easily be adapted to run in parallel, however, when the parallel decorator was employed over the whole ReliefE weight update step, even though all cores were utilized, no notable speedups were observed. Additional study of the intricacies of such decorator-based parallelism is left for further work.
4. When validating our and scikit-rebate's implementations against Weka's ReliefF, it turned out that ReliefF, as implemented in scikit-rebate differs with a somewhat negative effect on performance (as shown in this paper).
5. We did not experiment with detailed typing of the most time-consuming methods, however we believe some of the routines could be, this way, made even faster.
6. The intrinsic dimension algorithm (Algorithm 2) appears to *underestimate* the real dimension, leading to poorer performance in some cases.
7. Embedding target instances in hyperbolic space either works well, or does not work at all. We believe the observed performances are due to the intrinsic geometry of the data, which we will explore in further work.

We next discuss some of the general observations and their implications.

6 Discussion

In this work, we considered extensions of the original ReliefF approach with embedding-based distance computations to both multi-class and multi-label classification settings. We observed that, especially in MLC, embedding the target space can contribute both to lower running time and improve a classifier's performance. The distance that showed the most promising results was based on the cosine similarity, which is widely used when considering embedding-based learning and exploration. The main contribution of this work, the ReliefE ranking approach, is capable of operating via embeddings of both input and output spaces (e.g., in multi-label classification).

In this section, we comment on the obtained results and discuss further implications of ReliefE. We first observe that adaptive neighbor selection empirically performs very similarly to implementations where neighbor selection is hard-coded. This positive results indicate that one hyperparameter less needs to be tuned, should the user not have the computational resources for extensive grid searches. Further, the simple adaptation of the update step to take into account the distance to the mean of the neighbors similarly offered competitive results. One of the possible reasons for such performance is the potential cancellation of noise, as with averaging, especially in the embedding space, a joint representation is obtained that can also carry some information regarding the semantic similarity amongst the neighbors.

Within the proposed ReliefE approach, we also explored how data sparsification can be leveraged to further speed up feature ranking in high-dimensional settings. The sparsification procedure was targeted at larger, higher-dimensional data sets and did not affect smaller data sets as much.

In terms of multi-label classification performance, we observed that the classic adaptation of ReliefF with the proposed adaptive distance and the hamming loss was amongst the best performing options. Interestingly, the variant which used the cosine distance on the target space embeddings was also amongst the top three best performing solutions, indicating that multi-label classification potentially benefits more by considering only the embeddings of the target space instances (and not of instances in the feature space). Similarly, the absMean variant of ReliefF was also amongst the top five algorithms performed, indicating that this aggregation scheme is competitive to the widely accepted averaging followed by the absolute value computation step. The best variant of ReliefE that considered both feature and target space embeddings is ranked poorly, indicating that by embedding the feature space, performance is lost (albeit significant speedups can be obtained): This hints at a trade-off between performance and ranking quality. Of the remaining metrics, the subset and hyperbolic distances were amongst the worst performing ones, indicating that hyperbolic embeddings operate well in rather limited settings, possibly where a hierarchical structure of the target space can be observed.

This work is also one of the first (to our knowledge) to compare the performance curves of different ranking algorithms with the Fuzzy Jaccard Index. We observe that embedding-based algorithms proposed in this work behave very similarly, for both multi-label and multi-class classification. Especially in MLC, two consistent patterns emerge. All ReliefE variants are shown to be very similar to one another (red block in Fig. 12). However, also the hyperbolic and subset versions of ReliefF appear to behave similarly to the embedding-based ones, even though the input space was not embedded in these cases. For multi-label classification (Fig. 11), the ReliefE variants again emerge as the most similar (to one another). However, similarly to the MLC comparison, versions of the adapted ReliefF, as implemented in this work, were shown to yield similar performance curves to ReliefE-based variants.

Following the results of ablation studies, we believe further speedups could be obtained by considering fewer iterations. Current experiments indicate that potentially quadratic speedup could be obtained, as adequate performance was already observed after $\sqrt{|I|}$ iterations in some cases. Further, the number of iterations could also be adapted dynamically, by monitoring the feature ranking scores and detecting convergence before all iterations are carried out.

When studying individual data sets, e.g., DLBCL and opt-digits, we observe that ReliefE offers superior performance at a fraction of the computation time required by

the other methods, indicating that the development of approaches based on ideas introduced in this paper is a sensible research avenue.

In this work, we have evaluated feature rankings based on classification performance obtained by robust learners, such as logistic regression, which have not been fine-tuned. The purpose of such evaluation was to emphasize the effect of feature ranking. However, extensive studies of the interplay between regularization regimes (e.g., L1 vs. L2) and ranking performance could also offer interesting insights into the robustness of rankings, and further, their purpose. For example, a L1 regularized learner could automatically discard large parts of the feature space: Although this would be considered as feature selection (and not ranking), it would potentially offer similar results. We leave this type of experiments for further work.

Similarly, the Bayesian comparisons, involving mostly a state-of-the-art feature ranker MultiSURFstar and the proposed ReliefE algorithm(s), indicate that ReliefE is competitive and many times outperforms MultiSURFstar, even in a probabilistic sense. For example, the probability that ReliefE-absMean-adaptive outperforms MultiSURFstar is more than 30%, with most of the remaining probability density lying in the equal performance (rope) region.

Finally, we discuss several potentially interesting future empirical studies that would represent a non-trivial extension of the proposed work. Detailed analysis of the algorithms' performance with respect to various properties of the data sets could offer additional insights into when to use what type of ranking. We believe that meta-learning could be a promising research venue, as by linking the data sets' properties with suitable algorithms could largely benefit situations where embedding-based ranking is not the best option. Overall, if one optimizes for efficient performance on large, contemporary data sets, ReliefE offers a computationally efficient approach, that could serve as a first step to further study where to invest the remaining computational resources, and whether feature ranking is a sensible approach at all (it might not be for, e.g., image-based data). Similarly, understanding whether the choice of the distance score can be further *transferred* between similar data sets also represents an interesting research direction worth of further study. Overall, the proposed paper provides an empirical, as well as a theoretical foundation for potentially more involved embedding schemes, such as e.g., (variational) autoencoder-based ones. We believe that a relevant factor influencing ReliefE's performance is the quality of the *learned* representation, indicating that another promising research venue could be the investigation of different embedding approaches (this work explores different distances within a single embedding approach, but does not consider different embedding approaches).

7 Conclusions and further work

In this paper, we proposed one of the first embedding-based Relief implementations with both theoretical and practical grounding. We explored whether embedding the input, but also the output space onto a Riemannian manifold prior to feature ranking yields better rankings. The results indicate that, while being significantly faster, embedding-based ranking methods do not consistently outperform the ones that do not use the embeddings. However, we show that they are indeed consistently faster than all other Relief-based ranking approaches.

We also show that for multi-label classification, where additional complexity arises due to multiple label co-presence, ReliefE can offer more stable, and on data sets like Delicious, better performance. Further, we demonstrate that embedding the target label space is beneficial for the final ranking's quality in a multi-label setting. The proposed adaptive neighbor estimation procedure could be further developed in terms of the neighborhood dependence with respect to a given metric. Similarly, the current implementation potentially over-estimates the neighborhood size, which could be due to the nature of the embedded space or the method's bias. Both possibilities are to be explored in future work.

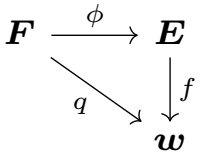
We believe that comparison of feature ranking algorithms should also be considered at the level of their properties and not only their performance. In this work, we show that embedding-based ranking gives rise to a fundamentally different type of rankings, which we believe are worthy of being studied further. To our knowledge, we are the first to perform such large-scale comparisons of a long list of ranking approaches (using, e.g., different similarities in MLC) and take into account also the actual values of importance scores of rankings (through the FUJI score), and not only the feature order.

We also observe that the variants of the original ReliefF, as re-implemented in this work, already offer superior performance to, e.g., the SURF branch of algorithms, indicating that their scikit-rebate implementations have some limitations in terms of numeric stability (and are not adapted at all to handle sparsity).

As further work, we believe the study of non-Euclidean spaces could yield many novel insights, as the target space is frequently of hierarchical nature, implying Euclidean geometry is not sufficiently good for its representation. In this work, we show initial results for embedding on a hyperboloid (Pincaré ball model). However, Lorenzian geometry can also be considered.

Appendix 1: Theoretical considerations of embedding spaces

As many of the recently introduced embedding-based methods tend to replace earlier methods, whilst maintaining the representation power, we believe the comparison of the two mappings, when considered, can be represented as a simple diagram. The example, considered to represent ReliefE's mapping compared to, e.g., that of the standard ReliefF's can be represented as



Here, the initial, real valued feature matrix $\boldsymbol{F}$ is either directly (q), or indirectly (ϕ and f) mapped to the output weight vector $\boldsymbol{w}$. Note that $q : \mathbb{R}^{|I|\times|F|} \rightarrow \mathbb{R}^{|F|}$, $\phi : \mathbb{R}^{|I|\times|F|} \rightarrow \mathbb{R}^{|I|\times d}$ and $f : \mathbb{R}^{|I|\times d} \rightarrow \mathbb{R}^{|F|}$. ReliefE operates under the assumption that the initial ranking can be retrieved via latent space $\boldsymbol{E}$ in two steps (ϕ and f).

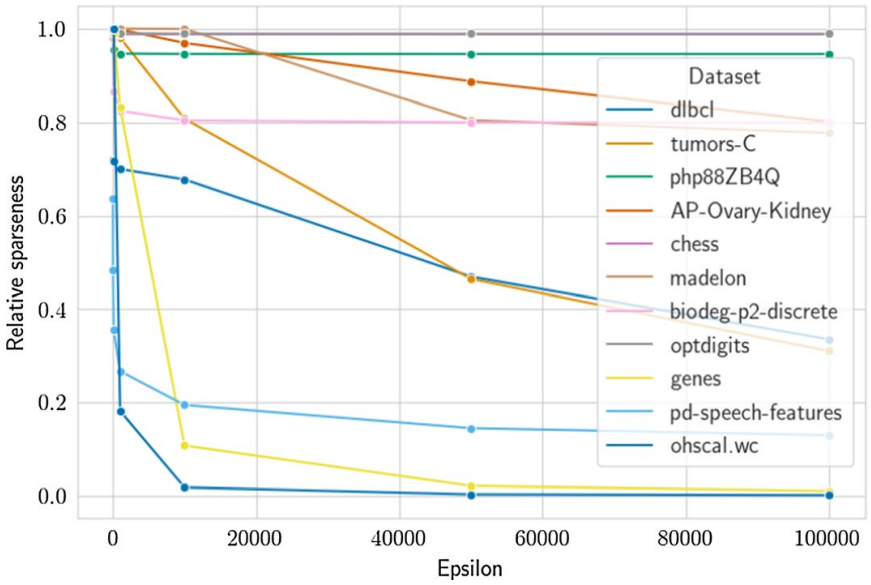


Fig. 14 Dependence of sparsification on approximation error allowed

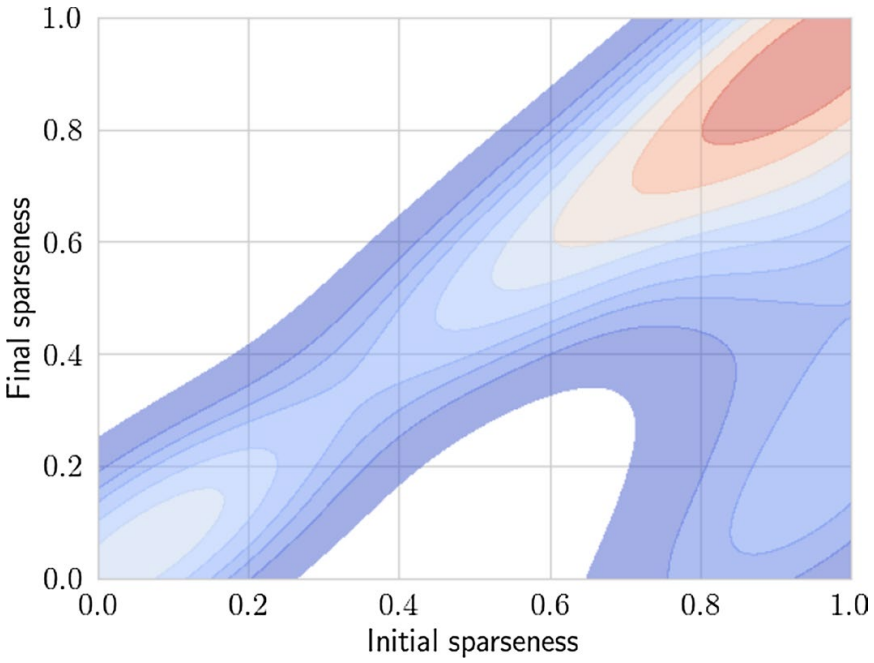


Fig. 15 Kernel density estimation of the relation between the initial and final sparsity

Appendix 2: Ablation study of data sparsity

The considered sparsification procedure is dependent on the parameter *epsilon*, i.e., the approximation error. The study of how different error thresholds impact the final sparsification result is shown in Fig. 14. It can be observed that most of the data sets only get sparsified after a rather large epsilon is permitted. The second ablation explores the relation between the initial data sparsity and the final sparsity, i.e., the sparsity of a given data set after the conducted sparsification procedure. The result is shown as a kernel density plot in Fig. 15.

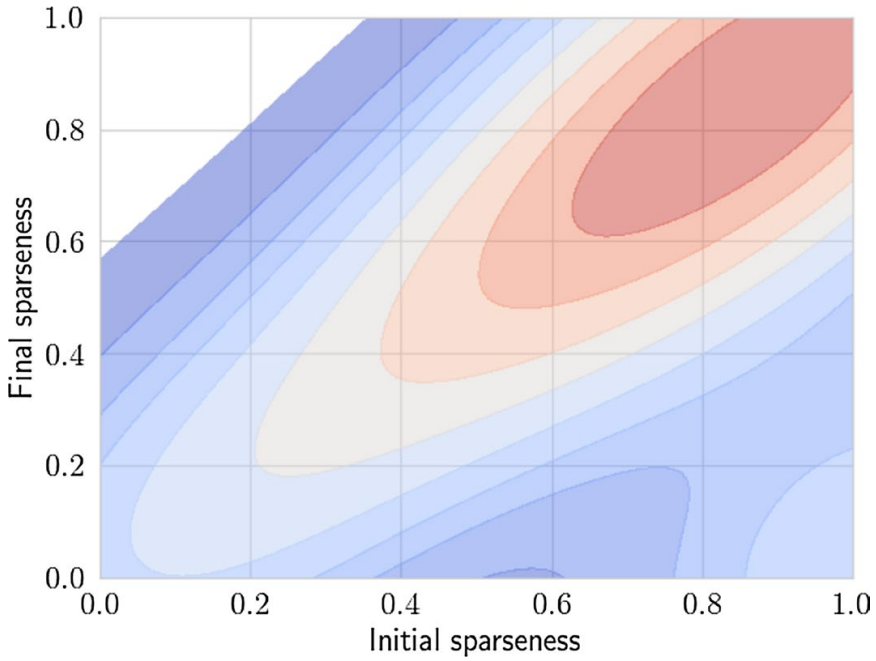


Fig. 16 Kernel density estimation – estimated epsilon values

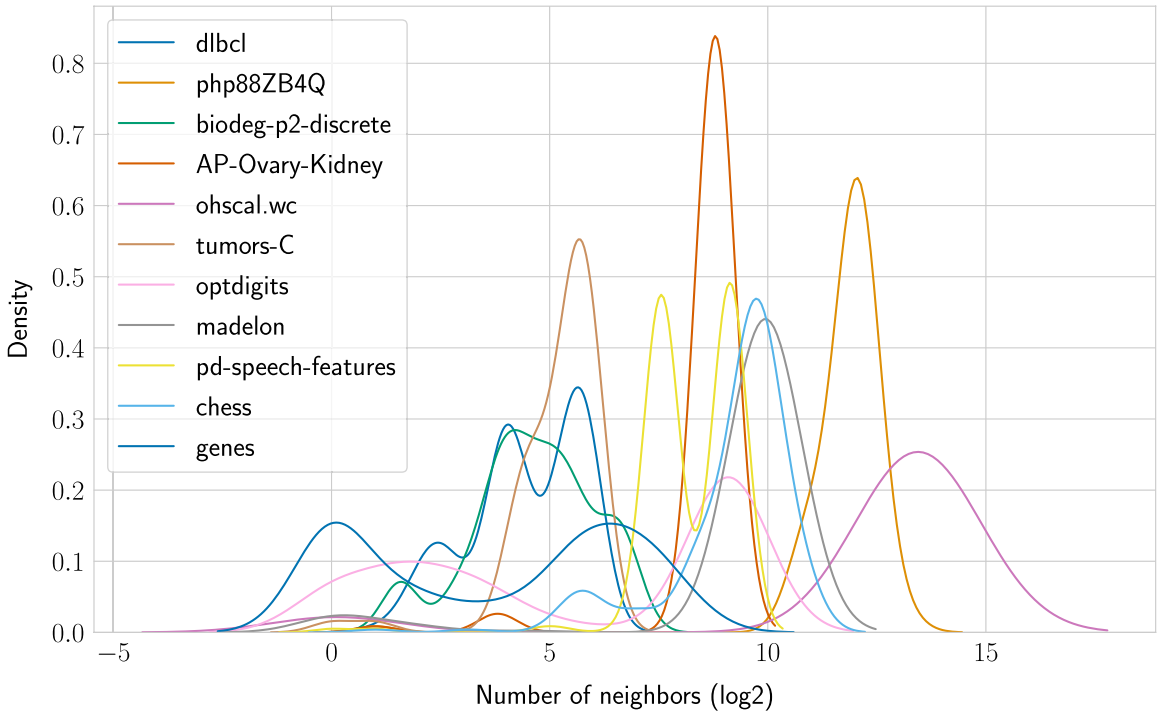


Fig. 17 Density of estimated k values for 100 iterations of ReliefE

The observed result indicates that when the data set is sparse to begin with, the result will be, as expected, similarly sparse. However, the vertical density at the rightmost part of the figure demonstrates that the sparsification procedure indeed yields sparser data, albeit not in all cases. A similar visualization can be produced for the space based on estimated epsilon values, shown in Fig. 16.

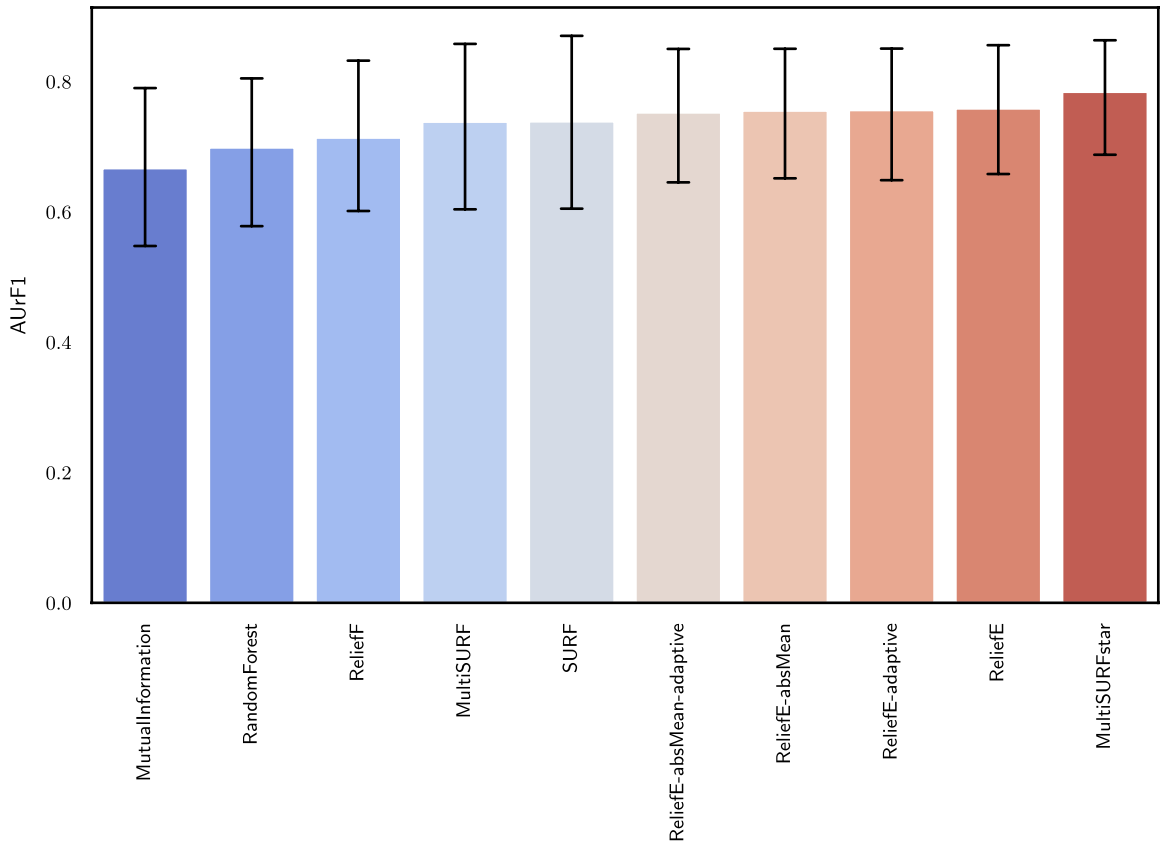


Fig. 18 Area under the relative F1 curve for multi-class classification. All ranking approaches perform similarly with no notable differences. More insights into the relative performance of ranking algorithms are provided by Bayesian tests and FUJI-based comparisons of performance curves

The considered estimate yields a similar landscape sparseness to that obtained via grid-search (Fig. 15), indicating decrease in most cases. However, there are examples where a given data set's density was substantially lowered, such as for example pd-speech-features and biodeg-p2-discrete. The results indicate that the considered estimate could be further relaxed, albeit at the cost of worse approximation of the input matrix, which could negatively impact the final performance.

Appendix 3: Adaptive *k* distributions

In this ablation study, we visualized the distributions of the neighborhoods across all considered MCC data sets. This plot demonstrates that for different data sets, differently sized neighborhoods were identified by the proposed heuristic (Fig. 17).

Appendix 4: Area under the rF1

The AUrF1 scores, averaged across data sets are shown in Fig. 18. It can be observed that the first 5 rankings behave very similarly w.r.t. this measure. Thus, we emphasize other types of comparison, where the differences are more apparent.

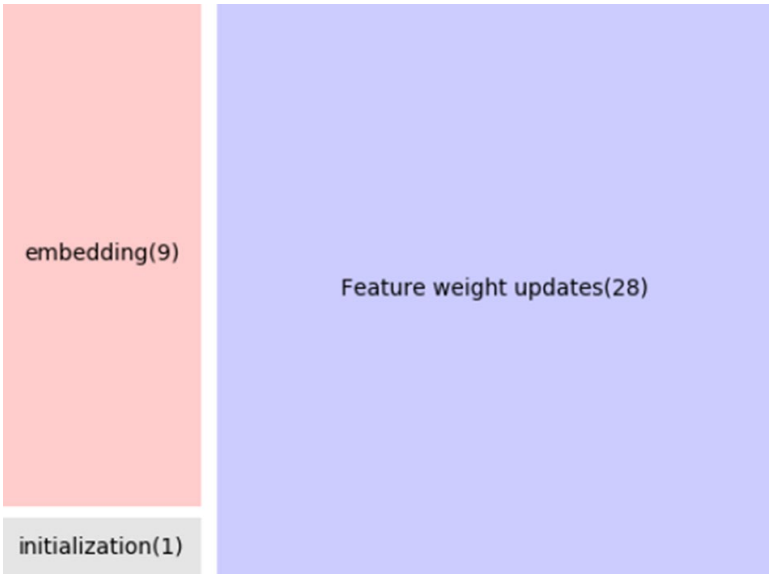


Fig. 19 Proportions of time spent on different methods within ReliefE on the p53 dataset. Majority of time is spent on weight update steps (as expected)

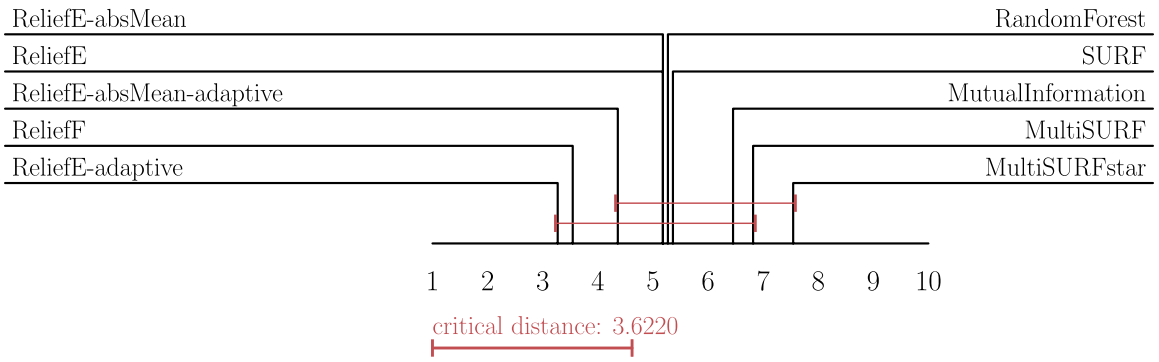


Fig. 20 Max first 100 features. Similarly to the situation with 50 top features, the ReliefE variants, including the adaptive one, perform well for the multi-class classification task

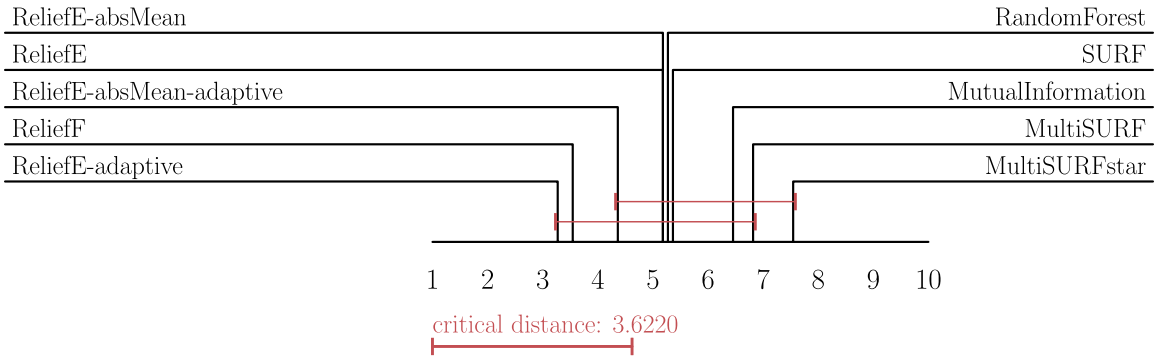


Fig. 21 Max first 50 features. The performance if considering only first 50 features. The adaptive version of ReliefE performs on average the best in this scenario

Appendix 5: Detailed analysis of running time

We additionally studied how different parts of ReliefE impact the total running time. For p53, one of the largest considered data sets, we visualize the proportions in Fig. 19.

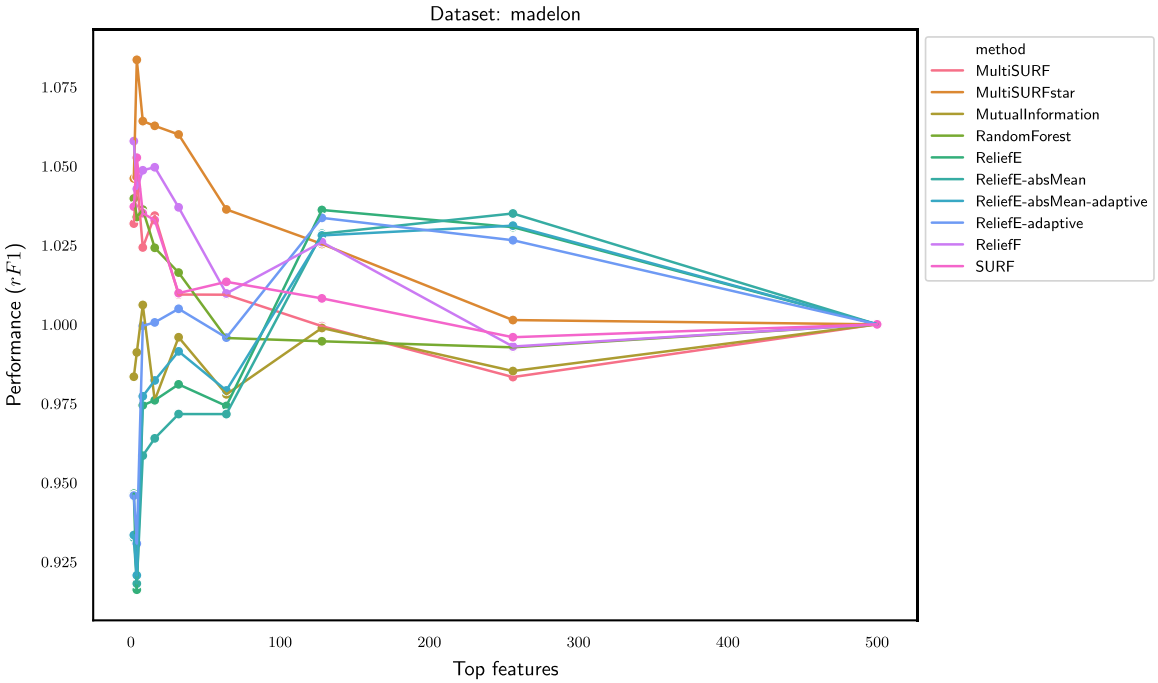


Fig. 22 Madelon performance curves. This result indicates that there exist situations where initially better rankings are obtained via e.g., the SURF branch of the algorithms, however, when considering more features, ReliefE variants are the only ones that find rankings which perform well

Appendix 6: Multi-class classification, additional rank diagrams

This appendix includes additional ablation studies in the form of critical distance diagrams presenting the performance for multi-class ranking (Figs. 20 and 21).

Appendix 7: Multi-class classification—case study with Madelon, DLBCL and genes

This section contains feature rankings, visualized for the Madelon data set, where either the ReliefE or a variant of ReliefF equiped with one of the proposed heuristics shows different behavior (better performance) (Fig. 22.) The visualized performances offer insights into behavior of the algorithms. For example, the ReliefF branch of adaptations (and vanilla ReliefF) peak at less than a hundred features, however, another performance peak where feature ranking is sensible ($rF1 > 1$) is around 250 features, where ReliefE-type algorithms are consistently amongst the best-performing ones.

The power of ReliefE is apparent when considering DLBCL data set (very high dimensional with not many instances). Results are shown in Fig. 23. Finally, the results for the genes data set are shown in Fig. 24. Note how the more time expensive SURF variants were not able to finish in dedicated time. Further, ReliefF is notably worse, requiring more information to detect the relevant signal. On the other hand ReliefE variants perform consistently well.

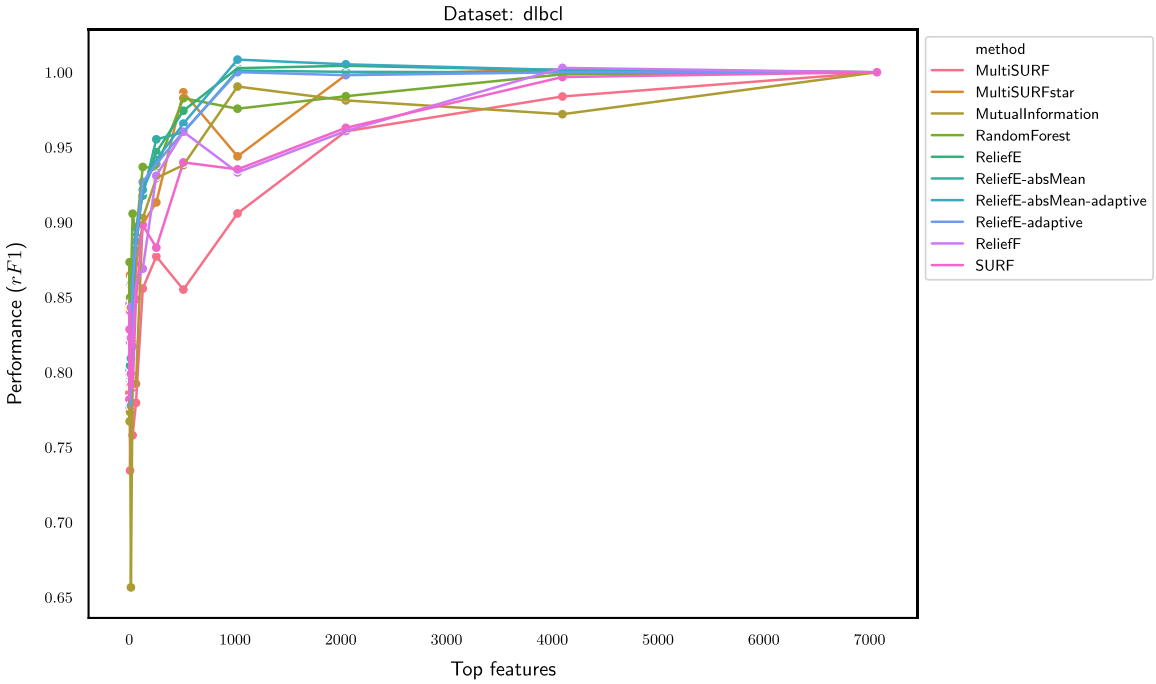


Fig. 23 DLBCL performance curves. The DLBCL is a very high-dimensional data set and reflects the ReliefE's capability to operate with high-dimensional feature spaces

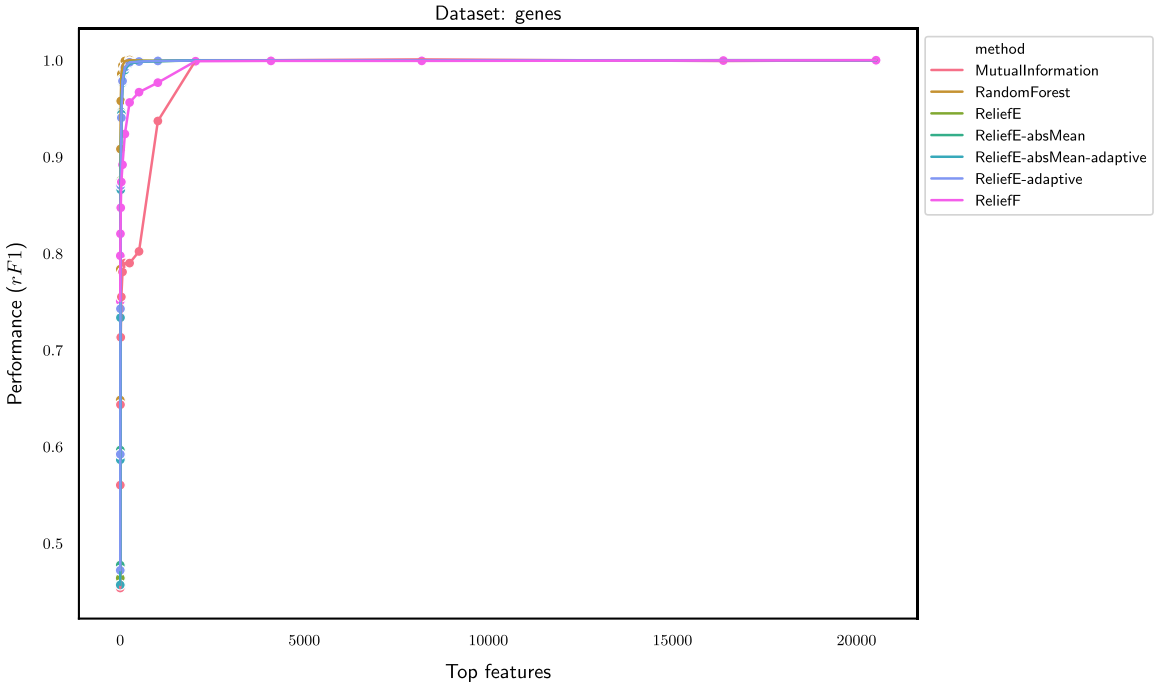


Fig. 24 Genes performance curves. Compared to mutual information (myopic)-based rankings ReliefE performs consistently better (steeper curve at the beginning in the first around hundred features

Appendix 8: Multi-label classification—additional rank diagrams

In this section, we present the average rank diagrams that offer insights into global distribution of the performances when multi-label classification setting is considered (Figs. 25 and 26).

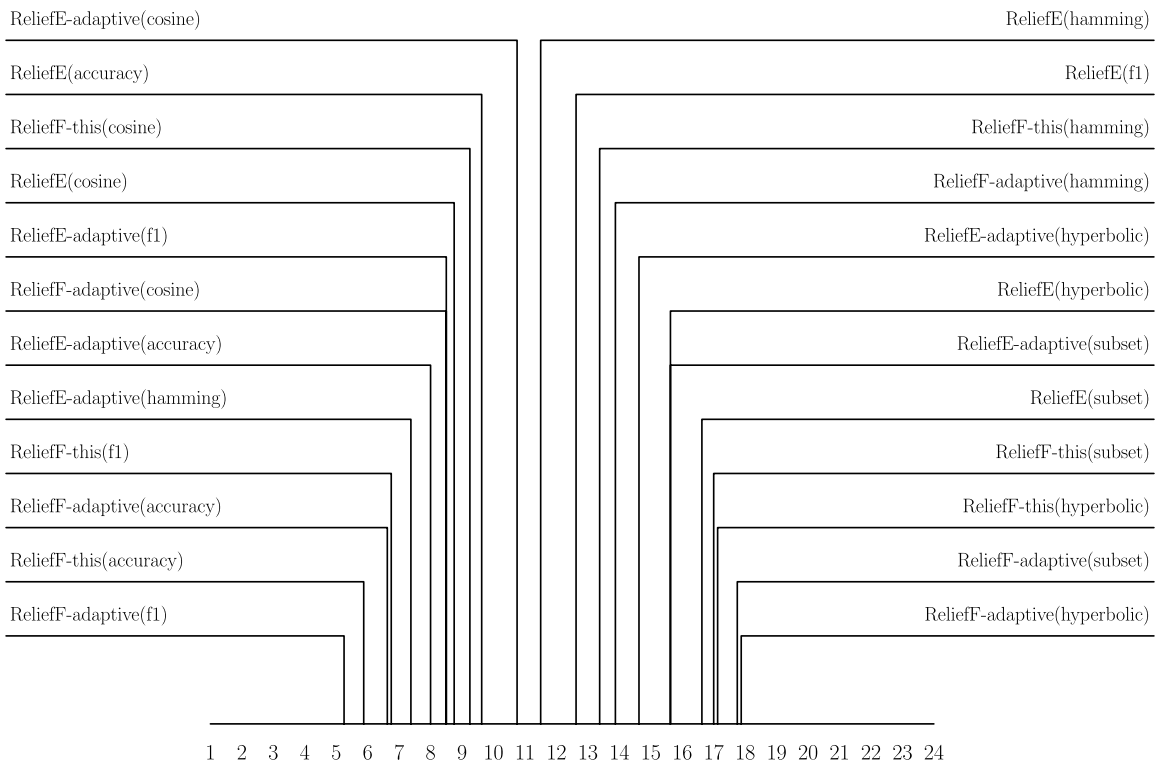


Fig. 25 Max (upper) F1 scores for top 10 highly-ranked features. The results indicate that the adaptive threshold step impacts the placing of the top features (adaptive) version of ReliefF

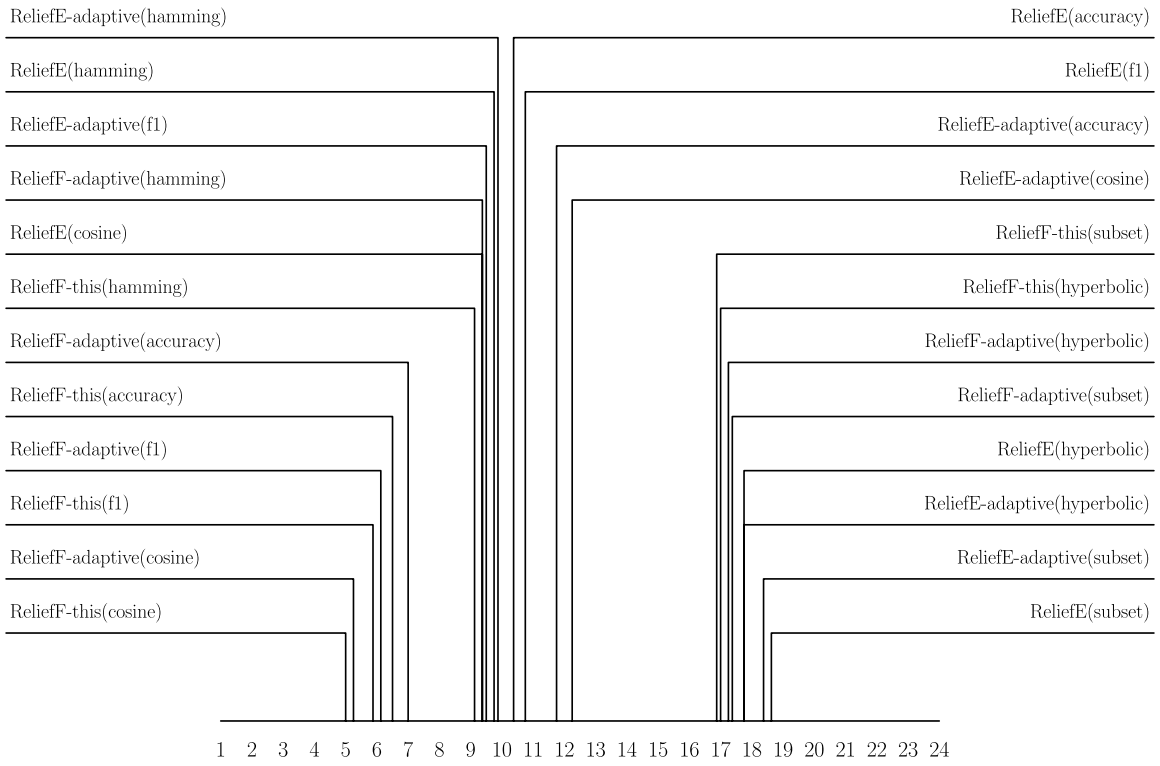


Fig. 26 Mean (upper) F1 scores for top 10 highly-ranked features. The average rank diagrams confirm the finding that if the target space is embedded via cosine distance, the MLC ReliefF variant performs the best

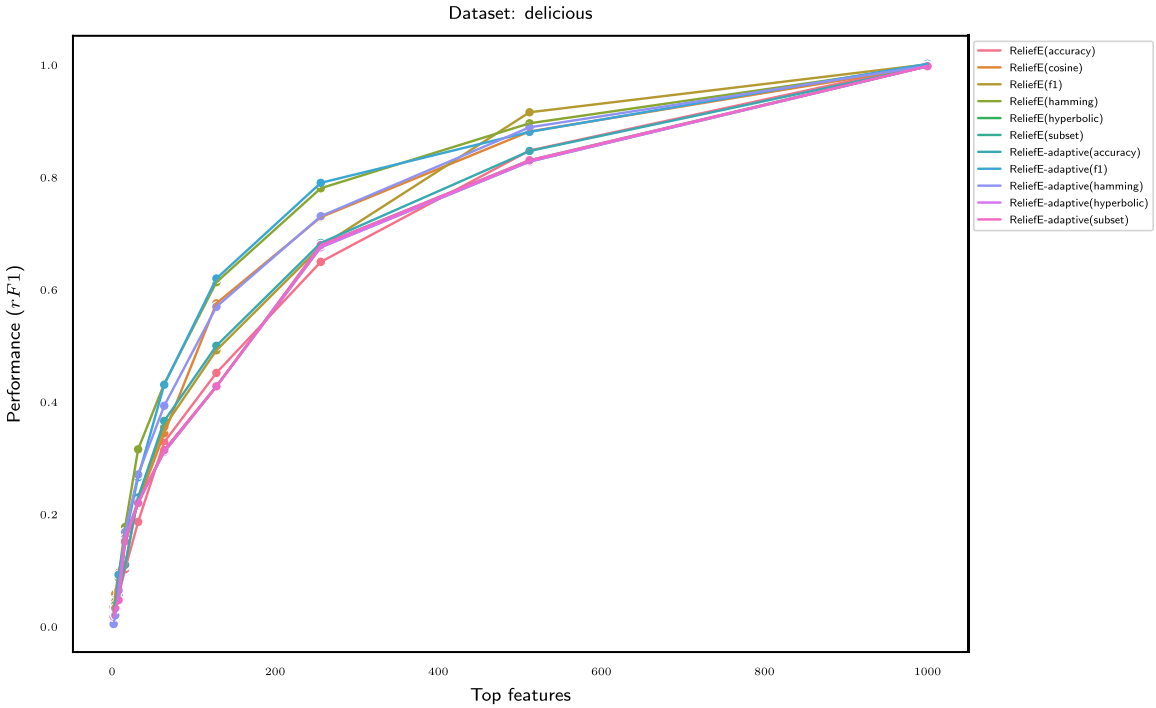


Fig. 27 Delicious performance curves. The ReliefE variants ReliefE-adaptive(f1) and ReliefE(hamming) perform consistently better up to 250 features

Appendix 9: Multi-label classification—case study with Delicious

We study in more detail the performance on the Delicious data set, as it offers interesting insights into the algorithms' performances (Fig. 27). The algorithms' performances are overall consistent. Note how cosine-based embeddings of the target space emerge as the best option (orange line), indicating embedding-based distances amongst the target instances can already offer competitive performance.

Acknowledgements We would like to acknowledge the Slovenian Research Agency (ARRS) for funding the first and the last author (BŠ, MP) through young researcher grants and supporting other authors (SD, NL) through the research program *Knowledge Technologies* (P2-0103) and the research project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078). This research was also partially supported by TAILOR (a project funded by the EU Horizon 2020 research and innovation programme under GA No. 952215) and AI4EU (GA No. 825619). We would also like to thank the administrators of the SLING supercomputing environment for the computing resources which made the empirical part of this study possible.

References

Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika, 34*(4), 369–374.

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. (2013). A public domain dataset for human activity recognition using smartphones. ESANN.

Armstrong, S. A., Staunton, J. E., Silverman, L. B., Pieters, R., den Boer, M. L., Minden, M. D., et al. (2002). Mll translocations specify a distinct gene expression profile that distinguishes a unique leukemia. *Nature Genetics, 30*(1), 41–47.

Arora, S., Hazan, E., & Kale, S. (2006). A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (pp. 272–279). Springer.

Balasubramanian, M., & Schwartz, E. L. (2002). The isomap algorithm and topological stability. *Science, 295*(5552), 7–7.

Benavoli, A., Corani, G., Demšar, J., & Zaffalon, M. (2017). Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis. *The Journal of Machine Learning Research, 18*(1), 2653–2688.

Breskvar, M., Kocev, D., & Dzeroski, S. (2018). Ensembles for multi-target regression with random output selections. *Machine Learning, 107*(11), 1673–1709. https://doi.org/10.1007/s10994-018-5744-y.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine, 34*(4), 18–42.

Cao, J., Spielmann, M., Qiu, X., Huang, X., Ibrahim, D. M., Hill, A. J., et al. (2019). The single-cell transcriptional landscape of mammalian organogenesis. *Nature, 566*(7745), 496–502. https://doi.org/10.1038/s41586-019-0969-x.

Connor, J. T., Martin, R. D., & Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks, 5*(2), 240–254.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*(Jan), 1–30.

Dong, W., Moses, C., & Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web* (pp. 577–586).

Džeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., & Van Laer, W. (1999). Experiments in predicting biodegradability. In textitInternational conference on inductive logic programming (pp. 80–91). Springer.

Eppstein, M. J., & Haake, P. (2008). Very large scale relieff for genome-wide association analysis. In *2008 IEEE symposium on computational intelligence in bioinformatics and computational biology* (pp. 112–119). IEEE.

Facco, E., d'Errico, M., Rodriguez, A., & Laio, A. (2017). Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports, 7*(1), 1–8.

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems, 151,* 78–94.

Granizo-Mackenzie, D., & Moore, J. H. (2013). Multiple threshold spatially uniform relieff for the genetic analysis of complex human diseases. In *European conference on evolutionary computation, machine learning and data mining in bioinformatics* (pp. 1–10). Springer.

Greene, C. S., Himmelstein, D. S., Kiralis, J., & Moore, J. H. (2010). The informative extremes: Using both nearest and farthest individuals can improve relief algorithms in the domain of human genetics. In *European conference on evolutionary computation, machine learning and data mining in bioinformatics* (pp. 182–193). Springer.

Greene, C. S., Penrod, N. M., Kiralis, J., & Moore, J. H. (2009). Spatially uniform relieff (surf) for computationally-efficient filtering of gene-gene interactions. *BioData Mining, 2*(1), 5.

Guyon, I., Gunn, S., Ben-Hur, A., & Dror, G. (2005). Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems* (pp. 545–552).

Han, E. H. S., & Karypis, G. (2000). Centroid-based document classification: Analysis and experimental results. In D. A. Zighed, J. Komorowski, & J. Żytkow (Eds.), *Principles of data mining and knowledge discovery* (pp. 424–431). Berlin: Springer.

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory, 14*(1), 55–63. https://doi.org/10.1109/TIT.1968.1054102.

Imdb dataset. (2010). https://sourceforge.net/projects/meka/files/Datasets/IMDB-F.arff/download.

Katakis, I., Tsoumakas, G., & Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD 2008 discovery challenge*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
Kira, K., Rendell, L. A., et al. (1992). The feature selection problem: Traditional methods and a new algorithm. *AAAI, 2,* 129–134.
Lam, S.K., Pitrou, A., & Seibert, S. (2015). Numba: A llvm-based python jit compiler. In *Proceedings of the second workshop on the LLVM compiler infrastructure in HPC* (pp. 1–6).
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444.
Liu, T., Moore, A. W., Yang, K., & Gray, A. G. (2005). An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems* (pp. 825–832).
Maaten, L. .v.d., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research, 9*(Nov), 2579–2605.
Masci, J., Boscaini, D., Bronstein, M., & Vandergheynst, P. (2015). Geodesic convolutional neural networks on Riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 37–45).
McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426.
McInnes, L., Healy, J., Saul, N., & Grossberger, L. (2018). Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software, 3*(29), 861.
Mežnar, S., Lavrač, N., & Škrlj, B. (2020). Snore: Scalable unsupervised learning of symbolic node representations. *IEEE Access, 8,* 212568–212588. https://doi.org/10.1109/ACCESS.2020.3039541.
Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K.Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 26, pp. 3111–3119). Curran Associates, Inc.
Moore, J. H., & White, B. C. (2007). Tuning relieff for genome-wide genetic analysis. In *European conference on evolutionary computation, machine learning and data mining in bioinformatics* (pp. 166–175). Springer.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research, 12*(Oct), 2825–2830.
Perovšek, M., Vavpetič, A., Kranjc, J., Cestnik, B., & Lavrač, N. (2015). Wordification: Propositionalization by unfolding relational data into bags of words. *Expert Systems with Applications, 42*(17–18), 6442–6456.
Pestian, J. P., Brew, C., Matykiewicz, P., Hovermale, D. J., Johnson, N., Bretonnel Cohen, K., & Duch, W. (2007). A shared task involving multi-label classification of clinical free text. In *Proceedings of the workshop on BioNLP 2007: Biological, translational, and clinical language processing (BioNLP'07)* (pp. 97–104).
Petković, M., Kocev, D., & Džeroski, S. (2018). Feature ranking with relief for multi-label classification: Does distance matter? In L. Soldatova, J. Vanschoren, G. Papadopoulos, & M. Ceci (Eds.), *Discovery science* (pp. 51–65). Cham: Springer International Publishing.
Petković, M., Slavkov, I., Kocev, D., & Džeroski, S. (2021). Biomarker discovery by feature ranking: Evaluation on a case study of embryonal tumors. *Computers in Biology and Medicine, 128,* 104143. https://doi.org/10.1016/j.compbiomed.2020.104143.
Petković, M., Škrlj, B., Kocev, D., & Simidjievski, N. (2020). Fuzzy Jaccard index: A robust comparison of ordered lists. https://arxiv.org/abs/2008.02216
Pomeroy, S., Tamayo, P., Gaasenbeek, M., Sturla, L. M., Angelo, M., McLaughlin, M., et al. (2002). Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature, 415,* 436–42. https://doi.org/10.1038/415436a.
Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., et al. (2018). A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR), 51*(5), 1–36.
Robnik-Šikonja, M., & Kononenko, I. (2003). Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning, 53*(1–2), 23–69.
Roweis, S. .T. ., & Saul, L. .K. . (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science, 290*(5500), 2323–2326.
Sakar, B. E., Isenkul, M. E., Sakar, C. O., Sertbas, A., Gurgen, F., Delil, S., et al. (2013). Collection and analysis of a Parkinson speech dataset with multiple types of sound recordings. *IEEE Journal of Biomedical and Health Informatics, 17*(4), 828–834.
Shapiro, A. D. (1984). The role of structured induction in expert systems. *Annexe Thesis Digitisation Project 2018 Block 19.*
Škrlj, B., Džeroski, S., Lavrač, N., & Petkovič, M. (2020). Feature importance estimation with self-attention networks. arXiv preprint arXiv:2002.04464.

Stepišnik, T., & Kocev, D. (2020). Hyperbolic embeddings for hierarchical multi-label classification. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, & Z. W. Raś (Eds.), *Foundations of intelligent systems* (pp. 66–76). Cham: Springer International Publishing.

Stiglic, G., & Kokol, P. (2010). Stability of ranked gene lists in large microarray analysis studies. *BioMed Research International, 2010*.

Stokes, M. E., & Visweswaran, S. (2012). Application of a spatially-weighted relief algorithm for ranking genetic predictors of disease. *BioData Mining, 5*(1), 20.

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. In *ECML/PKDD 2008 workshop on mining multidimensional data (MMD'08)*.

Ueda, N., & Saito, K. (2003). Parametric mixture models for multi-labeled text. In *Advances in neural information processing systems* (Vol. 15, pp. 721–728). MIT Press.

Urbanowicz, R. J., Olson, R. S., Schmitt, P., Meeker, M., & Moore, J. H. (2018). Benchmarking relief-based feature selection methods for bioinformatics data mining. *Journal of Biomedical Informatics, 85,* 168–188.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Weinstein, J. N., Collisson, E. A., Mills, G. B., Shaw, K. R. M., Ozenberger, B. A., Ellrott, K., et al. (2013). The cancer genome atlas pan-cancer analysis project. *Nature Genetics, 45*(10), 1113.

Chapter 6

Related Contributions

> The goal of Computer Science is to build something that will last at least until we have finished building it.
>
> William C. Brown

The following set of related contributions impacted the development of the core thesis contributions, even if not fully focusing on the common thesis thread of *neuro-symbolic* learning. The scope of individual contributions (dots) is shown in Figure 6.1. Each related contribution is described next.

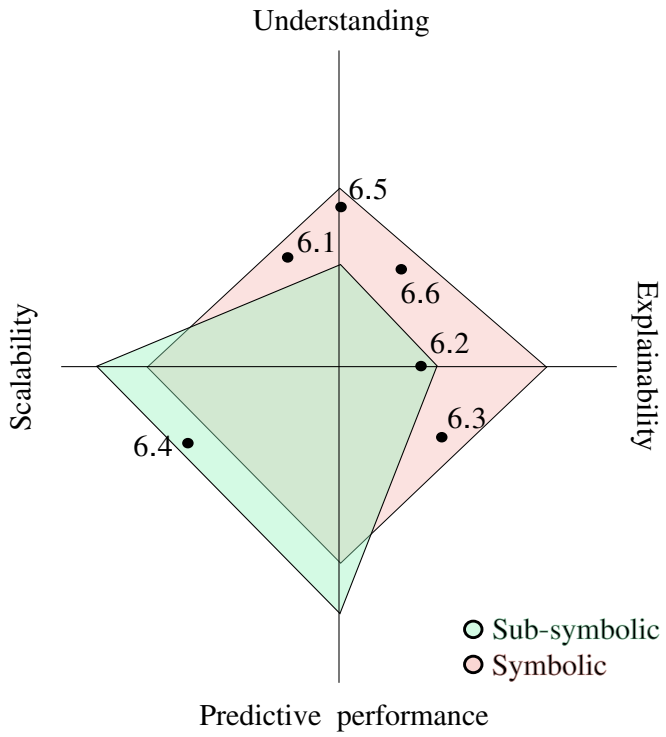
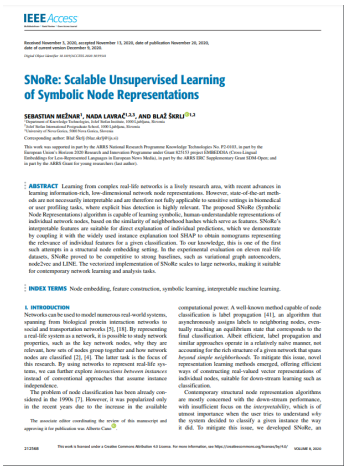


Figure 6.1: Schematic overview of the relations between the symbolic and sub-symbolic learning paradigms, with the placement of the related thesis contributions, presented by the labeled dots corresponding to the sections of this chapter where these contributions are described.

6.1 Symbolic Node Embedding

The contribution explored the notion of *symbolic node representation learning*. It is partially related to the core contribution DNR (Škrlj et al., 2021), as the goal was, similarly, to obtain node representations. The main difference, however, was, that this paper explored how one can obtain *sparse* node representations exclusively in a symbolic manner. As part of this paper, we also explored the extent to which the sparse and dense node representations are equivalent in terms of space use and proposed a simple heuristic for determining the dimension of symbolic space with similar computational requirements to its dense counterparts. The key hypothesis addressed in this work was whether symbolic embeddings are competitive to conventional, e.g., node2vec-based ones. Should this be the case, a substantially more interpretable node representation is possible, with implications for understanding the model behaviour at the individual prediction level. In the context of this thesis, the tool addresses the scalability and performance hypotheses, even though it is purely symbolic.


IEEE Access

SNoRe: Scalable Unsupervised Learning of Symbolic Node Representations

SEBASTIAN MEŽNAR[1], NADA LAVRAČ[1,2,3], AND BLAŽ ŠKRLJ[1,3]

Mežnar, S., Lavrač, N., & Škrlj, B. (2020). SNoRe: scalable Unsupervised Learning of Symbolic Node Representations. *IEEE Access*, *8*, 212568–212588. https://doi.org/10.1109/ACCESS.2020.3039541

6.2 Semantic Reasoning from Embedding-based Communities

The following publication represents our endeavor towards better understanding of how network node communities can be a side product of pre-trained embeddings and what are the implications of this approach, but also whether existing methodology from the field of ILP, namely the semantic reasoner Hedwig (Vavpetič et al., 2013), could be adopted to *better understand* the detected communities. Instead of inspecting the term sets present in a given community, the resulting semantic rules for some communities yielded term conjuncts, offering insight into the simultaneous occurrence of, e.g., multiple biological processes. One of the key contributions of this paper is an efficient implementation of the now well known NetMF algorithm, which joins multiple node embedding algorithms within the joint framework of *matrix factorization*. This work, not focusing on the neuro-symbolic paradigm by design, addresses first the issue of community detection, and subsequently, explanations of these communities. It is, to our knowledge, one of the first attempts to derive semantic rules for a given community – the main hypothesis addressed is how symbolic rule discovery can be used to better understand a given node grouping (community), hence not focusing on the overall approach's scalability, but showing competitive predictive performance.


Machine Learning (2020) 109:2161–2193

Embedding-based Silhouette community detection

Blaž Škrlj · Jan Kralj · Nada Lavrač

Škrlj, B., Kralj, J., & Lavrač, N. (2020). Embedding-based silhouette community detection. *Machine Learning*, *109*(11), 2161–2193. https://doi.org/10.1007/s10994-020-05882-8

6.3 Semantic Feature Construction with tax2vec

This contribution was one of the first contributions of the thesis' author, focusing on *feature construction* for the task of document classification. Even though a given document corpus can already include sufficient information for creating, e.g., word-based features which solve a given classification task, an active research endeavour attempts to go beyond the inclusion of 'raw' textual information, investigating whether *semantic* sources of knowledge are of potential use. The key goal of tax2vec was to explore whether word taxonomy-based features can complement the existing TF-IDF-based ones (or embedding-based ones). In this paper, we also investigated different graph-based heuristics for selection of the representative term set – here, one of the best-performing heuristics was based on the recent idea of graph-based ontology prunning (Kralj et al., 2019). The proposed tax2vec method was implemented as a self-contained library, requiring zero knowledge about parsing/processing, e.g., RDF-encoded semantic data. This tool addresses the explainability hypothesis, as its sole purpose is to construct interpretable, high-level semantic features that aid understanding, while also having the potential to improve few-shot classification.


Computer Speech & Language

tax2vec: Constructing Interpretable Features from Taxonomies for Short Text Classification

Blaž Škrlj, Matej Martinc, Jan Kralj, Nada Lavrač, Senja Pollak

Škrlj, B., Martinc, M., Kralj, J., Lavrač, N., & Pollak, S. (2020). tax2vec: constructing Interpretable Features from Taxonomies for Short Text Classification. *Computer Speech & Language*, 101104. https://doi.org/https://doi.org/10.1016/j.csl.2020.101104

6.4 Contextual Keyword Identification with TNT-KID

The subsequent related publication concerns the task of keyword identification. The task of keyword identification refers to the identification of key tokens or ordered token sets which effectively *summarize* a given document. The contribution (TNT-KID) is a transformer-based solution to this problem. By incorporating a specialized (final) detection head, we transformed the problem of keyword identification into the problem of token tagging. One of the key novelties introduced with TNT-KID was extensive investigation of *knowledge transfer* – by pre-training the language model on related corpora, we demonstrated improved performance. Further, the presented transfer learning capabilities indicate that substantially smaller amounts of data might be required for adequate keyword detection. Note that this approach is supervised, meaning that pre-labelled documents are obligatory for


Natural Language Engineering

ARTICLE

TNT-KID: Transformer-based neural tagger for keyword identification

Matej Martinc, Blaž Škrlj and Senja Pollak

the method's correct function. One of the key features of TNT-KID is its scalability and seamless *domain adaptation*. By fine-tuning the neural network model to a given domain, better keyphrases can be obtained. However, the symbolic learning paradigm offers a promising research direction with regards to the generation of informed prompts based on existing background knowledge sources, offering a potential way to reduce the amount of data needed for adaptation to completely new domains. The author helped with the design of TNT-KID.

> Martinc, M., Škrlj, B., & Pollak, S. (2021). TNT-KID: transformer-based neural tagger for keyword identification. *Natural Language Engineering*, 1–40. https://doi.org/10.1017/S1351324921000127

6.5 On Attention Vectors and Explanations

In this final related publication, we present our approach to interactive *attention visualization*. With the emergence of attention-based language models (foundation models), many research attempts were made to link the attention and its potential for explanation. We developed an online tool termed AttViz (https://attviz.ijs.si/) which facilitates interactive visualization of the neural attention layers. Our contribution offers an interactive (online) tool that enables exploration of the learned attention spaces and potentially helps the researcher understand what was emphasized, but more importantly, whether any artefacts were present during learning. The tool addresses mostly the 'understanding' aspect of learning – it was built to identify potential artefacts identified as relevant by the neural network and other relevant attention patterns.


Exploring Neural Language Models via Analysis of Local and Global Self-Attention Spaces

Blaž Škrlj, Jožef Stefan International Postgraduate School, Jožef Stefan Institute, Slovenia, blaz.skrlj@ijs.si
Shane Sheehan, University of Edinburgh, United Kingdom
Nika Eržen, Jožef Stefan Institute, Slovenia
Marko Robnik-Šikonja, University of Ljubljana, Slovenia
Saturnino Luz, University of Edinburgh, United Kingdom
Senja Pollak, Jožef Stefan Institute, Slovenia

Abstract

Large pretrained language models using the transformer neural network architecture are becoming a dominant methodology for many natural language processing tasks, such as question answering, text classification, word-sense disambiguation, text completion and machine translation. Commonly comprising hundreds of millions of parameters, these models offer state-of-the-art performance, but at the expense of interpretability. The attention mechanism is the main component of transformer networks. We present AttViz, a method for exploration of self-attention in transformer networks, which can help in explanation and debugging of the trained models by analysing associations between word tokens in an input sequence. We show how the existing visualising pipelines can be explored with AttViz, and how novel insights regarding the attention structure heads and their aggregations. We implemented the proposed methods in an online toolkit and an offline library. Using examples from news articles, we demonstrate how AttViz can be used to inspect and potentially better understand what a model has learned.

1 Introduction

Currently the most successful machine learning approaches for text-related tasks predominantly use large *language models*. They are implemented with transformer neural network architecture (Vaswani et al., 2017), extensively pretrained on large text corpora to capture context-dependent meanings of individual tokens (Devlin et al., 2019; Liu et al., 2019; Yang et al., 2019). Even though training of such neural networks with hundreds of millions of parameters is long and expensive (Radford et al., 2019), many pre-trained models have been made freely available. This has created an opportunity to explore how, and why these models perform well on many tasks. One of the main problems with neural network models is their lack of *interpretability*. Even though the models learn the given task well, understanding the reasons behind the predictions, and assessing whether the model is

Online interactive exploration
Offline statistical analysis
Labelled documents
Optional fine-grain model

Figure 1: An overview of AttViz suite. The system consists of two main functional modules supporting online and offline visualizations. The online visualization (http://attviz.ijs.si, first part of the paper) offers direct exploration of token attention across the space of input documents; its purpose is anomaly detection and general inspection of the attention space (of trained models). The offline part of AttViz (second part of the paper) is a Python library that offers computationally more demanding statistical analyses, ranging from visualization of key tokens for each attention head, computation of the attention head properties via FUJI integrals, and inspection of the attention distribution per-token basis.

76

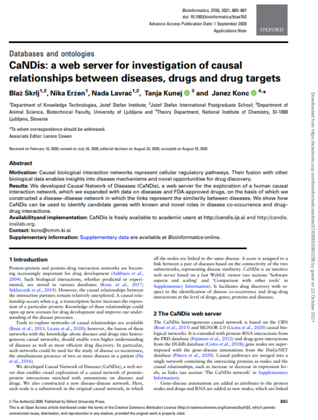
Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation, pages 76–83
April 19, 2021 © Association for Computational Linguistics

> Škrlj, B., Sheehan, S., Eržen, N., Robnik-Šikonja, M., Luz, S., & Pollak, S. (2021). Exploring neural language models via analysis of local and global self-attention spaces. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 76–83. https://aclanthology.org/2021.hackashop-1.11

6.6 Interactive exploration of causal drug-target interactions

The following publication represents our attempt to fuse two layers of biological information that can be of great relevance to researchers from the field of medicinal chemistry. This publication, although not being directly related to neuro-symbolic computing, helped us better understand how different sources of relational (background) knowledge can be processed, fused and utilized in a stand-alone web application. Substantial amounts of time were spent on making the web server as responsive and interactive as possible with the current 3D, webGL-based visualization libraries. The main aim of this project was to demonstrate the extent to which graph-based data fusion can facilitate the exploration of heterogeneous biological networks and the amount/effort required to host such graphs and serve them in real-time. To our knowledge, this was one of the first applications which incorporated the FDA-approved drugs alongside causal biological interactions. Although not directly considering learning, the developed tool illus-

trates how fusing heterogeneous data sources and subsequent presentation (visualization) of such data can offer novel insights into the interplay between drugs and drug targets. The key novelty of this tool was the inclusion of causal interactions, which add an additional layer of certainty to the derived networks (of reactions). In the context of this work, the tool focuses on a better understanding of a given learning process, albeit in this case concerning the users of the tool and not the algorithms as such.



Bioinformatics, 37(6), 2021, 885–887
doi: 10.1093/bioinformatics/btaa762
Advance Access Publication Date: 1 September 2020
Applications Note

Databases and ontologies

CaNDis: a web server for investigation of causal relationships between diseases, drugs and drug targets

Blaž Škrlj[1,2], Nika Eržen[1], Nada Lavrač[1,2], Tanja Kunej[3] and Janez Konc[4,*]

[1]Department of Knowledge Technologies, Jožef Stefan Institute; [2]Jožef Stefan International Postgraduate School; [3]Department of Animal Science, Biotechnical Faculty, University of Ljubljana and [4]Theory Department, National Institute of Chemistry, SI-1000 Ljubljana, Slovenia

*To whom correspondence should be addressed.

Associate Editor: Lenore Cowen

Received on February 10, 2020; revised on July 24, 2020; editorial decision on August 23, 2020; accepted on August 25, 2020

Abstract

Motivation: Causal biological interaction networks represent cellular regulatory pathways. Their fusion with other biological data enables insights into disease mechanisms and novel opportunities for drug discovery.

Results: We developed Causal Network of Diseases (CaNDis), a web server for the exploration of a human causal interaction network, which we expanded with data on diseases and FDA-approved drugs, on the basis of which we constructed a disease–disease network in which the links represent the similarity between diseases. We show how CaNDis can be used to identify candidate genes with known and novel roles in disease co-occurrence and drug–drug interactions.

Availabilityand implementation: CaNDis is freely available to academic users at http://candis.ijs.si and http://candis.insilab.org.

Contact: konc@cmm.ki.si

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Protein–protein and protein–drug interaction networks are becoming increasingly important for drug development (Ashburn *et al.*, 2004). Such biological interactions, whether predicted or experimental, are stored in various databases (Konc *et al.*, 2017; Szklarczyk *et al.*, 2019). However, the causal relationships between the interaction partners remain relatively unexplored. A causal relationship occurs when e.g. a transcription factor increases the expression of a particular protein. Knowledge of these relationships could open up new avenues for drug development and improve our understanding of the disease processes.

Tools to explore networks of causal relationships are available (Boué *et al.*, 2015; Licata *et al.*, 2020); however, the fusion of these networks with the knowledge about diseases and drugs into heterogeneous causal networks, should enable even higher understanding of diseases as well as more efficient drug discovery. In particular, such networks could be used for the study of disease co-occurrence, the simultaneous presence of two or more diseases in a patient (Hu *et al.*, 2016).

We developed Causal Network of Diseases (CaNDis), a web server that enables visual exploration of a causal network of protein–protein interactions enriched with annotations on diseases and drugs. We also constructed a new disease–disease network. Here, each node is a subnetwork in the original causal network, in which all the nodes are linked to the same disease. A score is assigned to a link between a pair of diseases based on the connectivity of the two subnetworks, representing disease similarity. CaNDis is an intuitive web server based on a fast WebGL viewer (see sections 'Software aspects and scaling' and 'Comparison with other tools' in Supplementary Information). It facilitates drug discovery with respect to the identification of disease co-occurrence and drug–drug interactions at the level of drugs, genes, proteins and diseases.

2 The CaNDis web server

The CaNDis heterogeneous causal network is based on the CBN (Boué *et al.*, 2015) and SIGNOR 2.0 (Licata *et al.*, 2020) causal biological networks. It is extended with protein–RNA interactions from the PRD database (Fujimori *et al.*, 2012) and drug–gene interactions from the DGIdb database (Cotto *et al.*, 2018); gene nodes are superimposed with the gene–disease annotations from the DisGeNET database (Piñero *et al.*, 2020). Causal pathways are merged into a single network containing the interacting proteins as nodes and the causal relationships, such as increase or decrease in expression levels, as links (see section 'The CaNDis network' in Supplementary Information).

Gene–disease annotations are added as attributes to the protein nodes and drugs and RNA are added as new nodes, which are linked

 885

Škrlj, B., Eržen, N., Lavrač, N., Kunej, T., & Konc, J. (2020). CaNDis: a web server for investigation of causal relationships between diseases, drugs and drug targets. *Bioinformatics*, *37*(6), 885–887. https://doi.org/10.1093/bioinformatics/btaa762

Chapter 7

Implementations

> The most effective way to do it, is to do it.
>
> Amelia Earhart

In this chapter, we focus on the implementations of the developed methods. Many times overlooked, the implementation aspects of a given project/paper can play a crucial role in the presented idea's *adoption*. We next present the main design decisions we adopted when developing the software packages related to each contribution.

7.1 From Libraries to Web Servers

There exist multiple possible ways of disseminating a newly developed algorithm. The design decision which governs the implementation thus depends on *users*. Given that the purpose of the dissertation was to produce easy-to-use methods for other machine learning practitioners, most developed software was developed in the form of *libraries*. This, however, is not the only possible way to disseminate the results. Should the computation times be rather short, and there is a requirement for interactivity, web-server-based solutions can also serve as a viable alternative, even though it requires the knowledge of a more diverse set of technologies. As the only tool which was developed as a web server was CaNDis (Škrlj, Eržen, et al., 2020), which is not of core relevance to the key focus of this thesis, the subsequent sections focus mostly on the development of simple-to-use Python libraries.

7.2 The Common Format of the Developed Libraries

We next present two of the papers' implementations that illustrate how the implemented libraries are used. Finally, we conclude this section with actual code examples that acquaint the reader with how the developed approaches/algorithms can be used with relatively little effort.

One of the key features of libraries such as *scikit-learn* (Pedregosa et al., 2011) is an intuitive Application Programming Interface (API), which abstracts away the unnecessary method calls/class initializations and offers the user a simple end-point for running, fine-tunning and storing a given machine learning model. For example, the key methods one encounters for most of the *scikit-learn*-compatible algorithms are the following ones.

fit(). This method takes as input either only the input space or the input and the target space, updating a given model's state so that, e.g., the weights correspond to

the association being learned. This method is common, for example, in supervised learning (fit(x, y)), but also in unsupervised learning (fit(x)). There exist special versions suitable for larger data sets (e.g., *partial_fit*), which, however, are used in a similar manner.

predict(). The prediction method is commonly applied on a given instance space (predict(X)), offering the user to obtain a collection of predictions from a given model.

transform(). The transformation method converts the original space into a new space. This method is useful, for example, when converting a list of documents into the corresponding TF-IDF space.

The mentioned three methods are the main tools a machine learning practitioner many times works with. They are normally inherent to a given model class, so that, for example, model.fit(X, Y) is a valid call. Even though, in theory, the three methods could be sufficient to represent most of what one encounters when designing/developing models, when considering, e.g., feature ranking, the model itself has attributes that might be of interest. The chosen API hence also, when needed, supports calls such as model.feature_importances, enabling access to the results of, e.g., the process of feature ranking.

7.3 Examples

This section presents some of the examples which illustrate the simplicity of using the developed approaches. We focus first on autoBOT, followed by DNR. The implementations of the ideas presented in other papers follow the same principles. The two examples were selected as they represent two different levels of complexity (software-wise), albeit both being currently packaged in the form of Python libraries.

7.3.1 An autoML in a few lines

The first example we consider is the implementation of autoBOT[1]. The example in Figure 7.1 represents a simple method which incorporates the key components of each autoML run. First, the data is loaded, followed by the call to the autoBOT. Here, we intentionally omitted all hyperparameters (e.g., evolution length) to demonstrate how simple it can be to experiment with the developed approach. The reader can also observe that predictions can be obtained in a Scikit-like manner (the same API). Additional effort was focused on developing the appropriate documentation, which documents all other functionalities in a systematic manner[2].

7.3.2 Neuro-symbolic Node embedding

The second example concerns *node embedding*. As here, networks are used as inputs, a similar API was implemented (Figure 7.2). Hence, the user can use the *fit*, and *transform* calls to obtain the embeddings of the desired nodes of interest. Note that only the adjacency structure is here used as input (unsupervised node representation learning). The output *embedding* is a matrix $\in \mathbb{R}^{|N| \times d}$.

[1] https://github.com/SkBlaz/autobot
[2] https://skblaz.github.io/autobot/

```
import autoBOTLib
import pandas as pd

def run():

    ## Load example data frame
    dataframe = pd.read_csv("trainData.tsv", sep="\t")
    train_sequences = dataframe['text_a']
    train_targets = dataframe['label']

    ## Evolve the learners
    autoBOTLibObj = autoBOTLib.GAlearner().evolve()

    ## Perform predictions
    dataframe2 = pd.read_csv("testData.tsv", sep="\t")
    test_sequences = dataframe2['text_a']

    predictions = autoBOTLibObj.predict(test_sequences)
```

Figure 7.1: An example of using the autoBOT library.

7.3.3 Overview of the implementations

This section summarizes the implemented approaches alongside their repositories. Most of the work is implemented in a manner similar to the two discussed libraries (DNR, autoBOT), making their use straight-forward for the users that are less programming savvy – at least that was the goal. The implementations are summarised in Table 7.1.

Table 7.1: Overview of the implementations related to this thesis.

Thesis part	Paper	Implementation	Software type
Relational	Deep Node Ranking	https://github.com/SkBlaz/DNR	Library
Relational	Propositionalization and Embeddings	https://github.com/SkBlaz/PropStar	Jupyter notebooks
Texts	autoBOT	https://github.com/SkBlaz/autobot	Library
Tabular	Attention-based feature ranking	https://github.com/SkBlaz/san	Library
Tabular	ReliefE	https://github.com/SkBlaz/reliefe	Library
Other	tax2vec	https://github.com/SkBlaz/tax2vec	Library
Other	AttViz	https://attviz.ijs.si/	Web server
Other	CaNDis	https://candis.ijs.si/	Web server
Other	SNoRe	https://github.com/smeznar/SNoRe	Library
Other	Silhouette Community Detection	https://github.com/SkBlaz/SCD	Library

The implementations also include, if possible (license-wise), the relevant collections of data sets useful for replicating their results. If needed, we also implemented sufficient dockerization, mostly via the Singularity containers (Kurtzer et al., 2017).

```
import dnrlib
import scipy.io

input_matrix = "graphSparse.mat"
mat = scipy.io.loadmat(input_matrix)

adjacency = mat['network']

dnr_class = dnrlib.DNR(device = "cpu",
                       batch_size = 64,
                       algorithm = "DNR",
                       num_epoch = 100,
                       hidden_size = 1,
                       num_pivot_nodes = None,
                       n_components = 256)

embedding = dnr_class.fit_transform(adjacency)
```

Figure 7.2: An example of using the DNR library.

Chapter 8

Conclusions

> A ship in port is safe, but that's not what ships are built for.
>
> Grace Hopper

This final chapter gives an overview and conclusion. It first summarizes the conducted work, the main results and their applications, followed by further work.

8.1 Overview of the Conducted Work

This dissertation focused on the neuro-symbolic learning paradigm with emphasis on the scalability of this type of method. We began by discussing the developed neuro-symbolic methods focusing on relational learning (Chapter 3). The two presented methods offer scalable neuro-symbolic solutions to the problems of instance classification from relational databases (SQL-like databases) and node classification in homogeneous real-life networks. We demonstrated that by adopting the neuro-symbolic paradigm, the proposed approaches offer competitive performance, while scaling better when considering the task of instance classification from relational databases. Furthermore, the proposed node classification algorithm termed Deep Node Ranking (DNR) offered state-of-the-art performance whilst simultaneously offering insights into the learned representations and their properties, which shed new light on the relationship between symbolic and sub-symbolic representations of the (same) entities.

In the second part of the dissertation, we focused on learning from texts (Chapter 4). Here, we proposed (to our knowledge) one of the first neuro-symbolic autoML engines aimed at solving the task of (multi-class) text classification. The proposed autoBOT approach revolves around the idea of *representation evolution* – a collection of representations of the same document is simultaneously re-weighted to identify the appropriate combination, which offers the best performance for a given training set of (labeled) documents. Special care was devoted to making autoBOT easy to use on off-the-shelf hardware. Current results indicate that models with orders of magnitude fewer parameters can offer competitive performance to current state-of-the-art contextual language models. Currently, we believe autoBOT should serve as a strong baseline when assessing both what are the key phrases/aspects of a text that are crucial for obtaining good classifiers (explainability) and obtaining out-of-the-box models that perform better than *ad hoc* baselines commonly used due to their simplicity/speed.

The final part of the dissertation focused on learning from tabular data (Chapter 5). Here, we presented two contributions that address two aspects of neuro-symbolic systems: the explainability when considering neural-only learners and the scalability when using

learned representations of instances for the task of feature ranking. We first demonstrated that by adopting an attention-like layer as the initial layer of a multi-layer deep neural network, if the trained neural network offered good performance, the attention coefficients resembled the ones obtained in a conventional feature ranking. This paper was one of the first to explore the link between neural attention and feature ranking, as this area of research has now gone mainstream and is the subject of discussion at every larger conference. The second contribution enabled us to explore whether – by adopting learned low-dimensional representations of the instances of interest – we could *speed up* the feature ranking process. Current results indicate that this is indeed the case, even though this type of ranking additionally depends on the *representation quality*, which can vary at the data set level.

8.2 Lessons Learned

Behind each contribution presented in this thesis, large amounts of experimentation and learning took place. This chapter focuses on the main lessons learned, i.e. the take-away messages concerning the development of neuro-symbolic methods for different input types. We begin by discussing the algorithmic aspects of representation learning that were relevant when designing algorithms capable of operating with sparse matrices. We continue the discussion by assessing the role of specialized hardware for the developed approaches, including promising new directions which are yet to be explored. We continue the discussion by assessing the development regime, which led us to the current metaheuristic governing the developed autoML's behaviour – the evolution strategies. We further discuss the impact of model sizes on their applicability and conclude with a note on Bayesian hypothesis evaluation, a promising new direction that offers probabilistic bounds, apart from the widely adopted p-based hypothesis (rejection/acceptance).

8.2.1 Implementing Representations

We begin the discussion with the notion of *representation implementation*. This aspect of representation learning, albeit not being the focus of many methods, can play a crucial role when considering the scaling properties of a given method. Note that implementing a representation corresponds to identifying a suitable *implementation* of a given representation with respect to solving a given *down-stream* task. Throughout the presented work, we encounter two main representation types: sparse and dense representations. Implementing dense representations is, as long as there is enough memory at one's disposal, straightforward and is supported in most of the linear algebra-focused computing libraries. The dense representations commonly correspond to low-dimensional real-valued spaces – here, millions or more instances can be stored seamlessly if the dimension is low enough. The limiting factor at some point becomes the bit precision of a given representation's matrix's entries, which can be further fine-tuned (see Section 3.3). In contrast to dense representations, which mostly result as the output of sub-symbolic learning, *sparse* representations are more common when considering symbolic representations. Given the topic of this dissertation, this type of representation was used in most of the presented contributions.

One of the main insights obtained during the development of the presented methods was that the type of sparse structure used could greatly determine the method's performance. Furthermore, a combination of sparse-dense representations, for example, in the case of well-performing solution PropDRM (Lavrač et al., 2020), was, that even though the whole data set could not be represented as a dense matrix, densifying only the mini-batches proved efficient (especially as GPUs could be leveraged this way). Even though existing libraries

many times support the mentioned sparse matrix-holding structures (Virtanen et al., 2020), using such structures implies that the considered piece of computation *can be vectorized*. Even though this is mostly the case, it is possible that many operations are required on the sparse matrices, which are non-trivial to vectorize, leading us to the second option regarding the implementation of sparse matrix-related operations. Contemporary Python frameworks such as, e.g., Numba (Lam, Siu Kwan and Pitrou, Antoine and Seibert, Stanley, 2015) offer transpilation of specifically formatted Python code to machine code, ensuring C-like performance. This approach was very suitable when implementing the ReliefE algorithm, as non-trivial operations are considered during each update step. Overall, the question of using sparse/dense data structures for representing the data of interest was, in most cases, related to the representation type considered: sparse matrices are primarily associated with symbolic, and dense with sub-symbolic representations.

8.2.2 To GPU or not to GPU

We next discuss the presented methods from the viewpoint of the hardware used to carry out the required computation. Contemporary machine learning approaches have profited greatly from being able to exploit the parallelism capabilities offered by GPUs or TPUs. Arguably, these hardware components were of crucial relevance when designing the first super-human vision and language generation systems. However, when considering neuro-symbolic learning, interesting trade-offs regarding the use of GPUs can emerge and are discussed next. Even though GPUs offer an order or more parallel processes, they are constrained both by recursion depth and allowed memory (memory storage is implemented differently from the classic CPU-based setting). These properties led us to explore the trade-offs between using multiple CPU and GPU threads. The developed methods, which can directly exploit the existing GPU-based computing power are autoBOT, PropDRM and DNR. In autoBOT however, only some of the representations (or more precisely transformer classes) can exploit the available GPUs; in the most recent implementation, the representation learners are limited to the ones based on large pre-trained language models. PropDRM, being a feed-forward neural network, generally profits from being able to use GPU-based computing power. One trade-off we observed, however, is that of densification of the representations during the forward pass. Here, should too large batches be considered with (high-dimensional inputs this is quickly the case), the available GPU can be overloaded. Finally, the DNR algorithm similarly can use GPUs for distilling the final representation. We observed a few fold speedups; however, for larger networks the GPU's memory started to become the bottleneck; hence the default DNR implementation operates in a CPU-only manner.

8.2.3 Taming the Evolution

One of the most computationally intensive endeavours of this thesis was the *meta learning* governing the document representation space (autoBOT). As there are no closed-form solutions that would enable efficient derivation of the final set of feature type weights, we resorted to metaheuristic optimization to at least approximately solve this problem. The selected metaheuristic approaches relied on *evolutionary strategies*, a type of genetic algorithms capable of operating with *real valued* solutions ($\in \mathbb{R}^d$).

The development of autoBOT required approximately two years of experimentation to arrive at the current point, which ensures relatively fast convergence and offers state-of-the-art performance. Further, developing an autoML system is prone to longer debugging cycles – for example, when investigating the selection schemes, it was apparent only after tens of generations whether a given scheme outperforms the others, resulting in long ex-

periments that already transcended the capabilities of an off-the-shelf machine. The main lessons learned during experimentation with evolution are the following. First, one should always start with as simple a solution as possible. We initially attempted to explore a *multi-objective* scenario, which also requires models to be compact; however, this type of optimization did not converge well enough to be considered. Furthermore, the solution selection schemes played a pivotal role during the development of autoBOT. Only when we considered the tournament-based selection, we start to observe faster and more consistent convergence. Note that this type of selection is one of the oldest and best-tested selection schemes. In terms of the initialization, we observed a substantial performance improvement when, instead of random initialization, we initialize the type importances to their performance in an internal round of cross-validation. Being normalized to the interval $[0, 1]$, these initial conditions offered adequate solutions out-of-the-box; however, when exposed to evolution, they resulted in an even better final solution. In terms of representation implementation, we considered two main scenarios. In the first one, the representation space was copied for each individual. As proved in our work, this results in linearly higher space complexity compared to a scenario where representations are precomputed and only re-weighted during evolution. At the time of development it was not clear how large a memory bottleneck this copying represents; we thus implemented both versions, only to discover that the one which conducts copying does not scale when considering tens of thousands of features even on a moderately sized data set. Note that by copying the individuals, the implementation is much simpler compared to the one currently used – we had to ensure that sparse matrices are used at each step of the evolution, as even a single unplanned densification could result in too high overhead for an off-the-shelf machine.

8.2.4 On Model Sizes and their Applicability

Next, we discuss the resulting models' sizes for different input data types, the implications of the obtained model sizes, and potentially interesting further developments. Contemporary machine learning models, especially those based on deep neural networks, are larger than simpler symbolic models such as linear regression-based models, decision trees and rules. A useful measure of model complexity, which is mostly resonant with its actual memory footprint, is the number of tunable parameters the final model consists of. Note that these are not *hyperparameters*, i.e. parameters governing the model creation. Contemporary deep neural networks comprise hundreds of millions and sometimes billions of parameters. Such over-parametrization leads to models which are Gigabytes in size, rendering them harder to deploy without specialized infrastructure. As hypothesized, neuro-symbolic approaches commonly require less space, making them potentially more useful in scenarios where the hardware is limited. Furthermore, when the data is very sparse, densifying it in order to, e.g., perform the forward pass, can already be a problematic endeavour. We demonstrated that the neuro-symbolic paradigm offers solutions to some of the problems of this type; by first learning the representations of concepts, which are subsequently processed in a symbolic manner, a given approach can retain most of its performance whilst scaling better. One of the most expensive models produced in terms of space were the results of evolution (text classifiers resulting from autoBOT runs). Here, the models were of various sizes, which directly reflect the properties of the underlying representations. For example, a version of *neuro-symbolic* autoBOT, which also incorporates language-model based (contextual) representations, could be Gigabytes in size. However, its symbolic counterpart was only about 100 Megabytes. This discrepancy can be crucial when deciding which model to implement as part of the final solution for a given classification task.

8.2.5 (Bayesian) Hypothesis Testing

We next discuss the notion of hypothesis testing and its developments in the last years. Benchmarking these methods against a collection of other (mostly) established algorithms is a common (good) practice when developing machine learning methods. However, merely reporting the performance without any additional comparisons is not necessarily enough – the user should also be interested in how, e.g., interchangeable a given pair of methods is and with what *probability* a given model outperforms the other one. In recent years, a paradigm shift related to model evaluation has been taking place. Bayesian model comparisons (Benavoli et al., 2017) are increasingly used to more precisely assess the relations between different algorithms. Even though, to our knowledge, such comparisons are limited to pairs of learning algorithms, they are very useful to determine to what extent the performances are similar/practically equivalent. In this thesis, we adopted this type of hypothesis testing where applicable. However, in most cases, it was complemented with classical (Friedman-Nemenyi) testing, which also remains used in the community. In all cases, we observed that the results of the tests were complementary – if a clear dominance of a given algorithm was observed with the classical approach, similar dominance was also observed as the result of Bayesian hypothesis testing. Furthermore, Bayesian comparisons offer additional context, which helped us better understand the relations between the algorithms' performances. A remaining major drawback of using Bayesian tests remains their large computational complexity if one accounts for the required number of experiment repetitions (the Bayesian model fitting itself is not problematic).

8.3 Further Work

This final section addresses the open problems suitable for further study. As *neuro-symbolic* learning is a relatively young paradigm, there remain interesting research opportunities to understand better the complementarity between symbolic and sub-symbolic learning in general. For example, only in recent years, an emphasis on incorporating computational reasoning with sub-symbolic representation learning has gained considerable attention. By being able to first, in a *self-supervised manner*, produce *representations* of the objects of interest and next reason with/about such objects in a consistent, computationally tractable manner, is an interesting open issue. Even though many existing deep learning-only systems have shown promising performance for the tasks of, e.g., protein folding, autonomous driving and similar, the aspect of understanding why such methods work at all could offer interesting knowledge regarding their *corner-case* behaviour. Even though perturbation-based studies are commonly used to understand individual decisions better, it often remains unclear what representations were learned and whether they generalize out-of-distribution. Recently, tasks such as ARC (Chollet, 2019) have unveiled that deep learning-based approaches are not sufficient for solving few-shot reasoning-based tasks – here, a promising endeavour to explore the link between the learned representations and *program synthesis* – the paradigm of inducing simple *programs* which produce a given output.

Apart from exploring the scalability aspects of neuro-symbolic learning, a promising research venue that is of increasing interest to the community is *few shot learning*. Here, a given algorithm's capability to learn well from as few data as possible (*but not fewer*) is the main focus. As one of the main discrepancies between symbolic and sub-symbolic learning is precisely the amount of data needed to learn well, such a research endeavour can unveil the limiting properties of existing models and the techniques to improve them. By being able to learn from smaller amounts of data, contemporary breakthroughs in machine learning become relevant for, e.g., less-resourced languages or domains where obtaining new instances is prohibitively expensive (e.g., biomedicine). Currently, the approach of

model pre-training has seen mainstream adoption. Deep learning models are here first *pre-trained* on large collections of data (e.g., a crawl of the internet), and only *fine-tuned* for solving a particular task. The success of model pre-training raises the question of whether this idea generalizes to *all* representation types in a similar manner. Even though we have partially explored this issue by demonstrating that incorporating domain knowledge in the form of knowledge graphs can be a sensible way of including priors about the observed facts into an existing process, many similar techniques are possible and potentially even more robust. For example, by maintaining a collection of *grounded* subject-predicate-object triplets that correspond to physically realistic relations, a model could, instead of learning the representation of an object for each task separately, learn to *combine* pre-learned representations of the objects. This process could be faster compared to model pre-training, even though task-dependent – the impact of a given representation on the final, e.g., document representation, would be context-dependent in a manner similar to representations obtained by using contextual language models.

The discussed autoML approach offered the initial results regarding *meta-transfer* learning for text-based tasks. We believe that building on this idea to understand better, but especially transfer symbolic knowledge from task to task, is a promising endeavour, mostly bound by the available computing resources (which is becoming less of a problem). Ideally, a *database* of algorithm configurations comprising of different representation type-weight associations should be available at the beginning of each new run, serving as a database of *meta priors*. This way, each autoML run would first determine the task type it is dealing with and jump-start the evolution accordingly. It remains an interesting question whether the neuro-symbolic paradigm can transcend the capabilities of the sub-symbolic learning-only paradigm. For example, in autoBOT, as the representation space was represented with sparse matrices, the downstream learning algorithms had to exploit sparse data structures to operate in space efficiently. We solved this issue by adopting simple linear classifiers which are very efficient when working with sparse data. However, such learners are *myopic*, and, as demonstrated, compared to larger neural language models, potentially miss interesting feature interactions (apart from not being pre-trained). We believe a logical next step at this point, which operates in a complementary manner to the current developments in the field of neural language models, would be to adopt a different *classification* paradigm – one similar to the *PropStar* presented in this work. Here, *embeddings* of features would be learned alongside the embeddings of *labels*, producing a joint latent space within which the classification takes place. The classification is based solely on measuring distances between the learning objects' representations, making the object placement a hard but solvable problem with contemporary gradient-based optimization methods. Interestingly, such space would contain both symbolic and sub-symbolic points (features), which has, to our knowledge, not been studied before in this setting.

One of the key lessons learned was the fact that if one is given an abundance of computing resources, one should be able to adopt a given approach to *leverage* that additional computing power. The unpublished part of this thesis related to scaling the autoBOT explored one of the possibilities (Section 4.2.4), indicating that more robust solutions which adopt dockerization (and are not prone to some of the jobs failing) need to be considered. In general, with more computing resources, especially the *search* part of a given approach can be many times seamlessly scaled. Examples include exploration of the space of hyperparameters, but also the weights themselves. We believe that with the increasing computing resources, the symbolic paradigm might offer the only solution to *better understand* the nature of the conducted search, helping the researchers/practitioners with profiling the (non-convex) objective space. A promising symbolic paradigm, which is yet to be fully exploited for this purpose, is *rule learning* – by being able to identify simple

logical patterns which *cover* parts of the observed space, the system's user can immediately identify the relations between the variables of interest (e.g., hyperparameters).

The current mainstream research direction focuses primarily on sub-symbolic (neural) algorithms. Given the success of this branch of algorithms, the necessity of maintaining parts of systems purely symbolic can be questioned. Tasks such as the mentioned ARC and similar are currently canonical examples where neural-only approaches do not dominate. Further, similar results were recently also observed when considering the best solutions to the NetHack Challenge (Küttler et al., 2020)[1], a reinforcement learning benchmark where the goal was to build agents which consider the NetHack game as the environment. Here, symbolic-only agents were substantially better performing when compared to neural-only solutions. According to the organizers of this shared task, efficient background knowledge incorporation is highly relevant for constructing successful agents; Given very sparse rewards, such knowledge can be the decisive factor between succeeding or failing at specific points in the game. Even though neural-only approaches can learn constraints relevant to a given system, incorporation of e.g., physical laws can speed up learning, and remains a lively research area (Jagtap et al., 2020; Shin et al., 2020).

With the advent of computing power suitable for machine learning purposes in recent years, a promising research direction includes re-implementation of methods capable of *inductive computational reasoning* on specialized hardware. Examples include the early approaches from the field of *inductive logic programming* including Aleph (Srinivasan, 2001) and similar computational reasoning engines. The recent developments in the field of neuro-symbolic learning such as DeepProbLog are already exploring this direction with, e.g., *the neural predicate*. However, a promising direction would include speeding up the rule discovery/induction procedures, making them subject to multithreading and other out-of-the-box functionalities of modern programming languages. Furthermore, one of the main caveats of the contemporary ILP-based approaches remains their usefulness to the researchers outside of the community. Simple, tutorial-like demonstrations utilizing high-level wrappers which abstract away the unnecessary complexity appear to have emerged as one of the key solutions to this problem. Similar developments boosted the adoption of neural methods in recent years. By emphasizing robust, simple-to-use implementations, the learning curve for the newcomers to a given field has decreased, increasing wider adoption. Thus, the neuro-symbolic paradigm could play a pivotal role in bringing decades-old methods which work very well into the spotlight (again), potentially systematically improving the machine learning toolkit used for practical applications.

As this thesis focused mainly on the development of novel methods, we believe the next step should include mostly *applications*. The reason for this would be two-fold. First, using the developed methods on new data sets, the implementations' validity is assessed, with imminent adaptations mostly needed when dealing with realistic (noisy) data sets. Furthermore, the added value of being partially explainable could offer significantly better understanding when considering tasks that are also hard for humans. Examples include ambiguity and fake news detection. The applicability of the neuro-symbolic paradigm remains to be evaluated in the following years; the current resurgence of novel methods/ideas on this topic indicates that the field carries the potential to offer methods that learn better and are more *inspectable*. To further this claim, the neuro-symbolic paradigm potentially transcends the notion of having both paradigms within the same 'method', as presented in this work. Even learner/explainer combinations can be understood as a whole as neuro-symbolic systems, implying that the widespread adoption of neuro-symbolic methods (at this level) is already present. Applications of immediate relevance for the developed set of methods include analysis of high-dimensional biological data sets, text classification

[1] https://nethackchallenge.com/

(when the data is scarce), node classification, link prediction, and model explanation. A promising direction would also explore how deep neural network-based representations, representing contextually different corpora could be jointly used for learning by adopting the representation ensemble idea considered in autoBOT paper.

More ambitious further work would explore beyond the current paradigm of *classical computing*. Albeit in its infancy, quantum computing could offer drastic speedups to a particular subset of machine learning-relevant problems, including kernel computation and search. Currently, it remains unclear whether this paradigm scales/will scale and when will it be suitable for mainstream adoption. We believe, however, that if a given neuro-symbolic approach is built mostly with linear-algebraic operations from widely supported libraries, it is possible that there will be drop-in replacements for parts of the codebase which could already prove beneficial. However, the development of quantum machine learning algorithms from scratch is possibly a much more complex challenge that will be tackled in the years.

References

Amizadeh, S., Palangi, H., Polozov, A., Huang, Y., & Koishida, K. (2020). Neuro-symbolic visual reasoning: Disentangling visual from reasoning. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th International Conference on Machine Learning* (pp. 279–290). PMLR.

Arik, Sercan Ö. and Pfister, Tomas. (2021). TabNet: Attentive Interpretable Tabular Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(8), 6679–6687. https://ojs.aaai.org/index.php/AAAI/article/view/16826

Benavoli, A., Corani, G., Demšar, J., & Zaffalon, M. (2017). Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis. *Journal of Machine Learning Research*, *18*(77), 1–36. http://jmlr.org/papers/v18/16-305.html

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., … Liang, P. (2021). On the Opportunities and Risks of Foundation Models. https://arxiv.org/abs/2108.07258

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating Embeddings for Modeling Multi-relational Data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 1877–1901). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Chollet, F. (2019). On the Measure of Intelligence. *arXiv preprint 1911.01547*.

Cohen, W., Yang, F., & Mazaitis, K. R. (2020). TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *Journal of Artificial Intelligence Research*, *67*, 285–325. https://doi.org/10.1613/jair.1.11944

Corus, D., & Oliveto, P. S. (2017). Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *22*(5), 720–732.

Cropper, A., Dumančić, S., & Muggleton, S. H. (2020). Turning 30: New ideas in inductive logic programming. *arXiv preprint arXiv:2002.11002*.

Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, *27*(4), 1071–1092.

Davis, L. (1991). Handbook of genetic algorithms. *Van Nostrand Reinhold; 1st edition (January 1, 1991)*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/n19-1423

Doerr, B., Le, H. P., Makhmara, R., & Nguyen, T. D. (2017). Fast genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference*, 777–784.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., & Zhou, D. (2019). Neural Logic Machines. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=B1xY-hRctX

Dong, Y., Chawla, N. V., & Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. *Proceedings of the 23rd ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 135–144.

Elsken, T., Staffler, B., Metzen, J. H., & Hutter, F. (2020). Meta-Learning of Neural Architectures for Few-Shot Learning. *arXiv preprint arxiv.org/abs/1911.11090*.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). Auto-sklearn: efficient and robust automated machine learning. *Automated Machine Learning* (pp. 113–134). Springer, Cham.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. *International Conference on Inductive Logic Programming*, 92–103.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, *90*(3), 317–346. https://doi.org/10.1007/s10994-012-5320-9

Garcez, A. d., & Lamb, L. C. (2020). Neurosymbolic AI: The 3rd Wave. *arXiv preprint arXiv:2012.05876*.

Gijsbers, P., & Vanschoren, J. (2019). GAMA: Genetic Automated Machine learning Assistant. *Journal of Open Source Software*, *4*(33), 1132. https://doi.org/10.21105/joss.01132

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Grčar, M., Trdin, N., & Lavrač, N. (2013). A Methodology for Mining Document-Enriched Heterogeneous Information Networks. *The Computer Journal*, *56*(3), 321–335. https://doi.org/10.1093/comjnl/bxs058

Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Hatolkar, Y., Agarwal, P., & Patil, S. (2018). A survey on road traffic sign recognition system using convolution neural network. *International Journal of Current Engineering and Technology*, *8*(1), 104–108.

He, X., Zhao, K., & Chu, X. (2021). AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, *212*, 106622.

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, *11*(1), 63–90. https://doi.org/10.1023/A:1022631118932

Ibrahim, M. M., & Kramann, R. (2019). GenesorteR: feature ranking in clustered single cell data. *bioRxiv*, 676379.

Jagtap, A. D., Kawaguchi, K., & Karniadakis, G. E. (2020). Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, *404*, 109136.

Jin, H., Song, Q., & Hu, X. (2019). Auto-Keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1946–1956.

Jomaa, H. S., Schmidt-Thieme, L., & Grabocka, J. (2021). Dataset2vec: Learning dataset meta-features. *Data Mining and Knowledge Discovery*, 1–22.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., ... Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*. https://doi.org/10.1038/s41586-021-03819-2

Karunaratne, T., & Boström, H. (2009). Graph Propositionalization for Random Forests. *2009 International Conference on Machine Learning and Applications*, 196–201. https://doi.org/10.1109/ICMLA.2009.113

Kaur, N., Kunapuli, G., Khot, T., Kersting, K., Cohen, W., & Natarajan, S. (2018). Relational Restricted Boltzmann Machines: A Probabilistic Logic Learning Approach. In N. Lachiche & C. Vrain (Eds.), *Proceedings of the inductive logic programming conference* (pp. 94–111). Springer International Publishing.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, *30*(10), 947–954.

Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*.

Kira, K., & Rendell, L. A. (1992). The Feature Selection Problem: Traditional Methods and a New Algorithm. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 129–134.

Kolata, G. (1982). How can computers get common sense? *Science*, *217*(4566), 1237–1238.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2019). Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. *Automated Machine Learning* (pp. 81–95). Springer, Cham.

Kralj, J., Robnik-Sikonja, M., & Lavrac, N. (2019). NetSDM: Semantic Data Mining with Network Analysis. *Journal of Machine Learning Research*, *20*(32), 1–50. http://jmlr.org/papers/v20/17-066.html

Kralj, J., Robnik-Šikonja, M., & Lavrač, N. (2018). HINMINE: heterogeneous information network mining with information retrieval heuristics. *Journal of Intelligent Information Systems*, *50*(1), 29–61. https://doi.org/10.1007/s10844-017-0444-9

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization Approaches to Relational Data Mining. In S. Džeroski & N. Lavrač (Eds.), *Relational data mining* (pp. 262–291). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-04599-2_11

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, *12*(5), 1–20. https://doi.org/10.1371/journal.pone.0177459

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., & Rocktäschel, T. (2020). The nethack learning environment. *arXiv preprint arXiv:2006.13760*.

Lam, Siu Kwan and Pitrou, Antoine and Seibert, Stanley. (2015). Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*.

Lamb, L., Garcez, A., Gori, M., Prates, M., Avelar, P., & Vardi, M. (2020). Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. Ellis Horwood.

Lavrač, N., & Flach, P. (2001). An Extended Transformation Approach to Inductive Logic Programming. *ACM Transactions on Computational Logic*, *2*(4), 458–494.

Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, *109*(7), 1465–1507. https://doi.org/10.1007/s10994-020-05890-8

Le, Q., Miralles-Pechuán, L., Kulkarni, S., Su, J., & Boydell, O. (2020). An overview of deep learning in industry. *Data Analytics and AI*, 65–98.

Le, Q., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *Proceedings of the 31st International Conference on International Conference on Machine Learning Volume 32*, 1188–1196.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lemke, C., Budka, M., & Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial intelligence review*, *44*(1), 117–130.

Leonori, S., Paschero, M., Mascioli, F. M. F., & Rizzi, A. (2020). Optimization strategies for Microgrid energy management systems by Genetic Algorithms. *Applied Soft Computing*, *86*, 105903.

Li, D., Deng, L., & Cai, Z. (2020). Intelligent vehicle network system and smart city management based on genetic algorithms and image perception. *Mechanical Systems and Signal Processing*, *141*, 106623.

Li, Q., Huang, S., Hong, Y., Chen, Y., Wu, Y. N., & Zhu, S.-C. (2020). Closed Loop Neural-Symbolic Learning via Integrating Neural Perception, Grammar Parsing, and Symbolic Reasoning. *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, *119*, 5884–5894. http://proceedings.mlr.press/v119/li20f.html

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, *16*(2), 146–160. https://doi.org/10.1007/BF01931367

Lodhi, H. (2013). Deep Relational Machines. *Proceedings, Part II, of the 20th International Conference on Neural Information Processing - Volume 8227*, 212–219. https://doi.org/10.1007/978-3-642-42042-9_27

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=rJgMlhRctm

Marra, G., Giannini, F., Faggi, L., Diligenti, M., Gori, M., & Maggini, M. (2020). Inference in relational neural machines. *Proceedings of the First International Workshop on New Foundations for Human-Centered AI (NeHuAI) co-located with 24th European Conference on Artificial Intelligence (ECAI 2020), Santiago de Compostella, Spain, September 4, 2020*, *2659*, 71–74.

Martinc, M., Škrlj, B., & Pollak, S. (2021). TNT-KID: transformer-based neural tagger for keyword identification. *Natural Language Engineering*, 1–40. https://doi.org/10.1017/S1351324921000127

Matsumoto, A., Kubota, C., & Ohwada, H. (2017). Extracting rules for successful conditions for artificial insemination in dairy cattle using inductive logic programming. *Proceedings of the 9th International Conference on Machine Learning and Computing*, 6–10.

McInnes, L., Healy, J., Saul, N., & Großberger, L. (2018). UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, *3*(29), 861. https://doi.org/10.21105/joss.00861

Mežnar, S., Lavrač, N., & Škrlj, B. (2020). SNoRe: scalable Unsupervised Learning of Symbolic Node Representations. *IEEE Access*, *8*, 212568–212588. https://doi.org/10.1109/ACCESS.2020.3039541

Michalski, R. S., Mozetič, I., Hong, J., & Lavrač, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, 1041–1045.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and Their Compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems Volume 2*, 3111–3119.

Mohr, F., Wever, M., & Hüllermeier, E. (2018). ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, *107*(8), 1495–1515. https://doi.org/10.1007/s10994-018-5735-z

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, *8*(4), 295–318. https://doi.org/10.1007/BF03037089

Muggleton, S. (1992). *Inductive logic programming*. Morgan Kaufmann.

Nazarczuk, M., & Mikolajczyk, K. (2020). Shop-vrb: A visual reasoning benchmark for object perception. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 6898–6904.

Olson, R. S., & Moore, J. H. (2016). TPOT: A tree-based pipeline optimization tool for automating machine learning. *Workshop on automatic machine learning*, 66–74.

Osojnik, A., Panov, P., & Džeroski, S. (2017). Multi-label classification via multi-target regression on data streams. *Machine Learning*, *106*(6), 745–770. https://doi.org/10.1007/s10994-016-5613-5

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The PageRank citation ranking: Bringing order to the web.* (tech. rep.). Stanford InfoLab.

Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T. B., & Leiserson, C. E. (2020). EvolveGCN: Evolving graph convolutional networks for dynamic graphs. *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perovšek, M., Vavpetič, A., Cestnik, B., & Lavrač, N. (2013). A Wordification Approach to Relational Data Mining. In J. Fürnkranz, E. Hüllermeier, & T. Higuchi (Eds.), *Discovery science* (pp. 141–154). Springer Berlin Heidelberg.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online Learning of Social Representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Petković, M., Kocev, D., & Džeroski, S. (2020). Feature ranking for multi-target regression. *Machine Learning*, *109*(6), 1179–1204. https://doi.org/10.1007/s10994-019-05829-8

Petković, M., Škrlj, B., Kocev, D., & Simidjievski, N. (2021). Fuzzy jaccard index: A robust comparison of ordered lists. *Applied Soft Computing*, *113*, 107849. https://doi.org/https://doi.org/10.1016/j.asoc.2021.107849

Pinkas, G., & Cohen, S. (2019). High-order Networks that Learn to Satisfy Logic Constraints. *FLAP*, *6*(4), 653–694. https://collegepublications.co.uk/ifcolog/?00033

Pisano, G., Ciatto, G., Calegari, R., & Omicini, A. (2020). Neuro-symbolic Computation for XAI: Towards a Unified Model. *WOA*, *1613*, 101.

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C., & Iyengar, S. S. (2018). A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, *51*(5), 1–36.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., & Tang, J. (2018). Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. *Proceedings of the eleventh ACM international conference on web search and data mining*, 459–467.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/BF00116251

Raedt, L. d., Dumančić, S., Manhaeve, R., & Marra, G. (2020). From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In C. Bessiere (Ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20* (pp. 4943–4950). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2020/688

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. https://arxiv.org/abs/1908.10084

Reimers, N., & Gurevych, I. (2020). Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. https://arxiv.org/abs/2004.09813

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton (Project Para)*. Cornell Aeronautical Laboratory.

Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S., & Beling, P. (2018). Deep learning detecting fraud in credit card transactions. *2018 Systems and Information Engineering Design Symposium (SIEDS)*, 129–134.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (tech. rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Russell, S., & Norvig, P. (2002). Artificial intelligence: A modern approach.

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1249.

Sarker, M. K., Zhou, L., Eberhart, A., & Hitzler, P. (2021). Neuro-Symbolic Artificial Intelligence Current Trends. *arXiv preprint arXiv:2105.05330*.

Shapiro, E. Y. (1981). *Inductive inference of theories from facts*. Yale University, Department of Computer Science.

Shin, Y., Darbon, J., & Karniadakis, G. E. (2020). On the convergence and generalization of physics informed neural networks. *arXiv e-prints*, arXiv–2004.

Sivaraman, A., Zhang, T., Van den Broeck, G., & Kim, M. (2019). Active inductive logic programming for code search. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 292–303.

Škrlj, B., Kralj, J., Konc, J., Robnik-Šikonja, M., & Lavrač, N. (2021). Deep node ranking for neuro-symbolic structural node embedding and classification. *International Journal of Intelligent Systems*, 1–30. https://doi.org/https://doi.org/10.1002/int.22651

Škrlj, B., Džeroski, S., Lavrač, N., & Petković, M. (2021). ReliefE: feature ranking in high-dimensional spaces via manifold embeddings. *Machine Learning*, 1–45. https://doi.org/10.1007/s10994-021-05998-5

Škrlj, B., Džeroski, S., Lavrač, N., & Petkovič, M. (2020). Feature Importance Estimation with Self-Attention Networks. In G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, & J. Lang (Eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)* (pp. 1491–1498). IOS Press. https://doi.org/10.3233/FAIA200256

Škrlj, B., Eržen, N., Lavrač, N., Kunej, T., & Konc, J. (2020). CaNDis: a web server for investigation of causal relationships between diseases, drugs and drug targets. *Bioinformatics*, *37*(6), 885–887. https://doi.org/10.1093/bioinformatics/btaa762

Škrlj, B., Kralj, J., & Lavrač, N. (2020). Embedding-based silhouette community detection. *Machine Learning*, *109*(11), 2161–2193. https://doi.org/10.1007/s10994-020-05882-8

Škrlj, B., Martinc, M., Kralj, J., Lavrač, N., & Pollak, S. (2020). tax2vec: constructing Interpretable Features from Taxonomies for Short Text Classification. *Computer Speech & Language*, 101104. https://doi.org/https://doi.org/10.1016/j.csl.2020.101104

Škrlj, B., Martinc, M., Lavrač, N., & Pollak, S. (2021). autoBOT: evolving neuro-symbolic representations for explainable low resource text classification. *Machine Learning*, 989–1028. https://doi.org/10.1007/s10994-021-05968-x

Škrlj, B., Sheehan, S., Eržen, N., Robnik-Šikonja, M., Luz, S., & Pollak, S. (2021). Exploring neural language models via analysis of local and global self-attention spaces. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 76–83. https://aclanthology.org/2021.hackashop-1.11

Slavkov, I., Karcheska, J., Kocev, D., & Džeroski, S. (2018). HMC-ReliefF: Feature ranking for hierarchical multi-label classification. *Computer Science and Information Systems*, *15*(1), 187–209.

Srinivasan, A. (2001). The aleph manual.

Srinivasan, A., Vig, L., & Bain, M. (2019). Logical Explanations for Deep Relational Machines Using Relevance Information. *Journal of Machine Learning Research*, *20*(130), 1–47.

Sun, Z., Deng, Z.-H., Nie, J.-Y., & Tang, J. (2019). RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *International Conference on Learning Representations*. https://openreview.net/forum?id=HkgEQnRqYQ

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, 6105–6114.

Theis, T. N., & Wong, H.-S. P. (2017). The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering*, *19*(2), 41–50.

Thomas, J., Coors, S., & Bischl, B. (2018). Automatic Gradient Boosting. *International Workshop on Automatic Machine Learning at ICML*.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855.

Trouillon, T., Dance, C. R., Gaussier, É., Welbl, J., Riedel, S., & Bouchard, G. (2017). Knowledge Graph Completion via Complex Tensor Factorization. *Journal of Machine Learning Research*, *18*(130), 1–38. http://jmlr.org/papers/v18/16-563.html

Turing, A. M. (1950). I.—computing machinery and intelligence. *Mind*, *59*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Vanschoren, J. (2019). Meta-Learning. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning: Methods, Systems, Challenges* (pp. 35–61). Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5_2

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998–6008.

Vavpetič, A., Novak, P. K., Grčar, M., Mozetič, I., & Lavrač, N. (2013). Semantic data mining of financial news articles. *International Conference on Discovery Science*, 294–307.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... 1.0 Contributors, S. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, *2018*.

Wang, C., Wu, Q., Weimer, M., & Zhu, E. ((2021). FLAML: A Fast and Lightweight AutoML Library. *Fourth Conference on Machine Learning and Systems (MLSys 2021)*. https://www.microsoft.com/en-us/research/publication/flaml-a-fast-and-lightweight-automl-library/

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, *1*(1), 67–82.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? *International Conference on Learning Representations*. https://openreview.net/forum?id=ryGs6iA5Km

Yang, C., Akimoto, Y., Kim, D. W., & Udell, M. (2019). OBOE: Collaborative filtering for AutoML model selection. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1173–1183.

Železný, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*(1-2), 33–63.

Zhang, C., Song, D., Huang, C., Swami, A., & Chawla, N. V. (2019). Heterogeneous graph neural network. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 793–803.

Zhang, S., Tay, Y., Yao, L., & Liu, Q. (2019). Quaternion Knowledge Graph Embeddings. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/d961e9f236177d65d21100592edb0769-Paper.pdf

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. *arXiv preprint arxiv.org/abs/1707.07012*.

Bibliography

Publications Related to the Thesis

Journal Articles

Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, *109*(7), 1465–1507. https://doi.org/10.1007/s10994-020-05890-8

Martinc, M., Škrlj, B., & Pollak, S. (2021). TNT-KID: transformer-based neural tagger for keyword identification. *Natural Language Engineering*, 1–40. https://doi.org/10.1017/S1351324921000127

Mežnar, S., Lavrač, N., & Škrlj, B. (2020). SNoRe: scalable Unsupervised Learning of Symbolic Node Representations. *IEEE Access*, *8*, 212568–212588. https://doi.org/10.1109/ACCESS.2020.3039541

Škrlj, B., Kralj, J., Konc, J., Robnik-Šikonja, M., & Lavrač, N. (2021). Deep node ranking for neuro-symbolic structural node embedding and classification. *International Journal of Intelligent Systems*, 1–30. https://doi.org/https://doi.org/10.1002/int.22651

Škrlj, B., Džeroski, S., Lavrač, N., & Petković, M. (2021). ReliefE: feature ranking in high-dimensional spaces via manifold embeddings. *Machine Learning*, 1–45. https://doi.org/10.1007/s10994-021-05998-5

Škrlj, B., Eržen, N., Lavrač, N., Kunej, T., & Konc, J. (2020). CaNDis: a web server for investigation of causal relationships between diseases, drugs and drug targets. *Bioinformatics*, *37*(6), 885–887. https://doi.org/10.1093/bioinformatics/btaa762

Škrlj, B., Kralj, J., & Lavrač, N. (2020). Embedding-based silhouette community detection. *Machine Learning*, *109*(11), 2161–2193. https://doi.org/10.1007/s10994-020-05882-8

Škrlj, B., Martinc, M., Kralj, J., Lavrač, N., & Pollak, S. (2020). tax2vec: constructing Interpretable Features from Taxonomies for Short Text Classification. *Computer Speech & Language*, 101104. https://doi.org/https://doi.org/10.1016/j.csl.2020.101104

Škrlj, B., Martinc, M., Lavrač, N., & Pollak, S. (2021). autoBOT: evolving neuro-symbolic representations for explainable low resource text classification. *Machine Learning*, 989–1028. https://doi.org/10.1007/s10994-021-05968-x

Conference Papers

Škrlj, B., Džeroski, S., Lavrač, N., & Petkovič, M. (2020). Feature Importance Estimation with Self-Attention Networks. In G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, & J. Lang (Eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious*

Applications of Artificial Intelligence (PAIS 2020) (pp. 1491–1498). IOS Press. https://doi.org/10.3233/FAIA200256

Škrlj, B., Sheehan, S., Eržen, N., Robnik-Šikonja, M., Luz, S., & Pollak, S. (2021). Exploring neural language models via analysis of local and global self-attention spaces. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 76–83. https://aclanthology.org/2021.hackashop-1.11

Other Publications

Dermastia, M., Škrlj, B., Strah, R., Anžič, B., Tomaž, Š., Križnik, M., Schönhuber, C., Riedle-Bauer, M., Ramšak, Ž., Petek, M., Kladnik, A., Lavrač, N., Gruden, K., Roitsch, T., Brader, G., & Pompe-Novak, M. (2021). Differential Response of Grapevine to Infection with 'Candidatus Phytoplasma solani' in Early and Late Growing Season through Complex Regulation of mRNA and Small RNA Transcriptomes. *International Journal of Molecular Sciences*, *22*(7). https://doi.org/10.3390/ijms22073531

Janež, N., Škrlj, B., Sterniša, M., Klančnik, A., & Sabotič, J. (2021). The role of the listeria monocytogenes surfactome in biofilm formation. *Microbial Biotechnology*, *14*(4), 1269–1281. https://doi.org/https://doi.org/10.1111/1751-7915.13847

Jukič, M., Škrlj, B., Tomšič, G., Pleško, S., Podlipnik, Č., & Bren, U. (2021). Prioritisation of compounds for 3clpro inhibitor development on sars-cov-2 variants. *Molecules*, *26*(10). https://doi.org/10.3390/molecules26103003

Kokalj, E., Škrlj, B., Lavrač, N., Pollak, S., & Robnik-Šikonja, M. (2021). BERT meets shapley: Extending SHAP explanations to transformer-based classifiers. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 16–21. https://aclanthology.org/2021.hackashop-1.3

Koloski, B., Stepišnik-Perdih, T., Pollak, S., & Škrlj, B. (2021). Identification of covid-19 related fake news via neural stacking. In T. Chakraborty, K. Shu, H. R. Bernard, H. Liu, & M. S. Akhtar (Eds.), *Combating online hostile posts in regional languages during emergency situation* (pp. 177–188). Springer International Publishing.

Koloski, B., Zosa, E., Stepišnik-Perdih, T., Škrlj, B., Paju, T., & Pollak, S. (2021). Interesting cross-border news discovery using cross-lingual article linking and document similarity. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 116–120. https://aclanthology.org/2021.hackashop-1.16

Martinc, M., Škrlj, B., Pirkmajer, S., Lavrač, N., Cestnik, B., Marzidovšek, M., & Pollak, S. (2020). Covid-19 therapy target discovery with context-aware literature mining. In A. Appice, G. Tsoumakas, Y. Manolopoulos, & S. Matwin (Eds.), *Discovery science* (pp. 109–123). Springer International Publishing.

Mežnar, S., Lavrač, N., & Škrlj, B. (2021). Prediction of the effects of epidemic spreading with graph neural networks. In R. M. Benito, C. Cherifi, H. Cherifi, E. Moro, L. M. Rocha, & M. Sales-Pardo (Eds.), *Complex networks & their applications ix* (pp. 420–431). Springer International Publishing.

Miok, K., Škrlj, B., Zaharie, D., & Robnik-Šikonja, M. (2021). To BAN or not to BAN: Bayesian attention networks for reliable hate speech detection. https://doi.org/10.1007/s12559-021-09826-9

Pelicon, A., Pranjić, M., Miljković, D., Škrlj, B., & Pollak, S. (2020). Zero-shot learning for cross-lingual news sentiment classification. *Applied Sciences*, *10*(17). https://doi.org/10.3390/app10175993

Pelicon, A., Shekhar, R., Škrlj, B., Purver, M., & Pollak, S. (2021). Investigating cross-lingual training for offensive language detection. *PeerJ Computer Science*, *7*, 559. https://doi.org/10.7717/peerj-cs.559

Pollak, S., Robnik-Šikonja, M., Purver, M., Boggia, M., Shekhar, R., Pranjić, M., Salmela, S., Krustok, I., Paju, T., Linden, C.-G., Leppänen, L., Zosa, E., Ulčar, M., Freienthal, L., Traat, S., Cabrera-Diego, L. A., Martinc, M., Lavrač, N., Škrlj, B., ... Toivonen, H. (2021). EMBEDDIA tools, datasets and challenges: Resources and hackathon contributions. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, 99–109. https://aclanthology.org/2021.hackashop-1.14

Škrlj, B., Kralj, J., & Lavrač, N. (2019). Py3plex toolkit for visualization and analysis of multilayer networks. *4*(1). https://doi.org/10.1007/s41109-019-0203-7

Škrlj, B., Novak, M. P., Brader, G., Anžič, B., Ramšak, Ž., Gruden, K., Kralj, J., Kladnik, A., Lavrač, N., Roitsch, T., & Dermastia, M. (2021). New Cross-Talks between Pathways Involved in Grapevine Infection with 'Candidatus Phytoplasma solani' Revealed by Temporal Network Modelling. *Plants*, *10*(4). https://doi.org/10.3390/plants10040646

Škrlj, B., & Renoust, B. (2020). Layer entanglement in multiplex, temporal multiplex, and coupled multilayer networks. *5*(1). https://doi.org/10.1007/s41109-020-00331-w

Stepišnik Perdih, T., Pollak, S., & Škrlj, B. (2021). Jsi at the finsim-2 task: Ontology-augmented financial concept classification. *Companion Proceedings of the Web Conference 2021*, 298–301. https://doi.org/10.1145/3442442.3451383

Biography

The author of this thesis, after finishing his Bachelor's degree in molecular biotechnology, pursued computer science (Information and Communication Technologies) MSc and PhD studies at the Jožef Stefan International Postgraduate School. During his studies, he was supervised by Prof. Nada Lavrač, with whom he had the opportunity to explore the neuro-symbolic paradigm, its relation to the established field of inductive logic programming, and address the selected open problems in this field. During his PhD studies, the author also collaborated with researchers from the National Institute of Biology and the National Institute of Chemistry, with whom he worked on applying the machine learning techniques developed as part of his studies to real-life problems. During his second year of PhD studies, he received the ASEF scholarship which enabled him to visit the Stanford InfoLab where he worked under the supervision of Prof. Jure Leskovec. During his final year he also visited the group of Prof. Christian B. Mendl at Technische Universität München (TUM), where he had the opportunity to study quantum walks and their applicability to walk-based representation learning. During the course of his PhD, he worked on numerous national and European research projects and co-supervised multiple students.