WEB WORKFLOWS FOR DATA MINING IN THE CLOUD

Janez Kranjc

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Nada Lavrač, Jožef Stefan Institute, Ljubljana, Slovenia, and Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
Co-Supervisor: Assoc. Prof. Dr. Marko Robnik Šikonja, Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Evaluation Board:
Assist. Prof. Dr. Martin Žnidaršič, Chair, Jožef Stefan Institute, Ljubljana, Slovenia, and Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
Assoc. Prof. Dr. Igor Mozetič, Member, Jožef Stefan Institute, Ljubljana, Slovenia, and Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
Prof. Dr. Hendrik Blockeel, Member, Kaotolieke Universiteit Leuven, Leuven, Belgium

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Janez Kranjc

WEB WORKFLOWS FOR DATA MINING IN THE CLOUD

Doctoral Dissertation

SPLETNI DELOTOKI ZA RUDARJENJE PODATKOV V OBLAKU

Doktorska disertacija

Supervisor: Prof. Dr. Nada Lavrač

Co-Supervisor: Assoc. Prof. Dr. Marko Robnik Šikonja

Ljubljana, Slovenia, March 2017

Acknowledgments

This thesis would have never been finished without the help of a number of people. Their support and help throughout my studies are greatly appreciated.

First of all I would like to thank my supervisor Nada Lavrač for her infinite amount of patience, enthusiasm, support and her always helpful and insightful research ideas and comments. I am also very grateful to my co-supervisor Marko Robnik Šikonja for the copious amount of time spent reviewing my work and his greatly appreciated comments and ideas. I would also like to thank the members of my evaluation board Martin Žnidaršič, Igor Mozetič and Hendrik Blockeel for reading my thesis and providing comments for improvement.

I also wish to thank the funding bodies that financially supported my research: the Department of Knowledge Technologies at the Jožef Stefan Institute, the Jožef Stefan International Postgraduate School, and the European Commission for funding the research projects e-LICO, BISON, FIRST, ConCreTe, MUSE, and HBP in which I was involved.

Special thanks goes to Vid Podpečan for his work that helped spark the ideas that eventually became this dissertation and his contributions to my work and all his helpful insights and ideas. I would also like to thank everyone I had the pleasure of cooperating and co-authoring papers with, especially: Matic Perovšek for the work we did together on TextFlows; Jasmina Smailović for our work on sentiment analysis and active learning; Roman Orač for implementing the DiscoMLL library that contributed to big data processing in ClowdFlows; and Anže Vavpetič for his numerous contributions and creating the Relational Data Mining package.

I am thankful to all the work colleagues at the Department of Knowledge Technologies for creating a great working environment. I would like to thank Darko Aleksovski for his numerous contributions to the open-source projects developed during the time of my studies. Thanks also to Jan Kralj, Borut Sluban and Miha Grčar for their valuable time for productive and insightful discussions that greatly motivated my research work.

Last but not least, I must thank my family and friends for their encouragement and support. Finally, I owe sincere and earnest thankfulness to my wife Nina for her love, understanding, support and patience throughout my study.

Abstract

The thesis addresses the development of an innovative data mining platform ClowdFlows and novel knowledge discovery scenarios implemented therein as executable data mining workflows. The ClowdFlows platform is implemented as a cloud-based web application with a graphical user interface which supports the construction, execution and sharing of data mining workflows. Big data analytics is supported by several algorithms, including novel ensemble techniques implemented using the MapReduce programming model, and a special stream mining module for real-time analysis using continuous parallel workflow execution.

The adaptability of ClowdFlows is demonstrated through descriptions of two platforms that expand the usage of the web-based workflow environment to fields other than data mining. The ConCreTeFlows platform is a novel platform for computational creativity, while TextFlows is focused mainly on text mining and natural language processing. The ConCreTeFlows platform is demonstrated with a conceptual blending use case, while features of TextFlows are demonstrated on three use cases: comparison of document classifiers and different part-of-speech taggers on a text categorization problem, and outlier detection in document corpora.

We present novel use cases in Inductive Logic Programming (ILP) and Relational Data Mining (RDM). The main novelty is a propositionalization technique called wordification which can be seen as a transformation of a relational database into a corpus of text documents. The wordification methodology and the evaluation procedure are implemented as executable workflows in ClowdFlows. The implemented workflows include several other ILP and RDM algorithms, as well as the utility components that enable access to these techniques to a wider research audience.

The real-time analysis and data stream mining capabilites are demonstrated on a novel active learning scenario for dynamic adaptive sentiment analysis, which is able to handle changes in data streams and adapt its behavior over time. Established stream mining techniques are transferred to the visual programming paradigm and demonstrated through the sentiment analysis use case, using active learning of microblogging data streams with a linear Support Vector Machine.

The thesis contributes to open-source scientific software. The ClowdFlows platform, its adaptations ConCreTeFlows and TextFlows, and all the described use cases are publicly available. This enables experiment reproducibility, as well as workflow adaptations and enhancements.

Povzetek

Disertacija obravnava razvoj inovativne platforme za podatkovno rudarjenje, imenovane ClowdFlows, ter nove načine odkrivanja znanja, implementirane v platformi v obliki izvršljivih delotokov. Platforma ClowdFlows je izdelana kot spletna aplikacija v oblaku, njen grafični uporabniški vmesnik pa omogoča izdelavo, izvedbo in objavo delotokov za podatkovno rudarjenje. Analiza velikih podatkov je mogoča s pomočjo več algoritmov, med drugim z inovativnimi ansambelskimi tehnikami, ki uporabljajo paradigmo MapReduce, in z modulom rudarjenja podatkovnih tokov v realnem času, ki uporablja neprekinjeno vzporedno izvajanje delotokov.

Prilagodljivost platforme ClowdFlows je prikazana z opisom dveh izpeljanih platform, ki širita uporabnost delotokov spletne platforme na druga področja. Platforma ConCreTeFlows je namenjena računalniški ustvarjalnosti, medtem ko se platforma TextFlows osredotoča na tekstovno rudarjenje in procesiranje naravnega jezika. Platforma ConCreTeFlowa je prikazana s primerom uporabe konceptualnega povezovanja, medtem ko so posebnosti platforme TextFlows prikazane na treh primerih uporabe: s primerjavo klasifikatorjev dokumentov in različnih označevalcev besednih vrst ter z odkrivanjem osamelcev v zbirkah besedil.

V disertaciji so predstavljeni novi primeri uporabe na področjih induktivnega logičnega programiranja in relacijskega podatkovnega rudarjenja. Novost je metoda transformacije relacijskih podatkovnih baz v zbirke besedil, imenovana *wordification* (besedizacija). Evalvacija razvite metode je vključena v platformo ClowdFlows kot izvršljiv delotok. Poleg opisane evalvacije so kot delotoki dostopni tudi drugi algoritmi za relacijsko podatkovno rudarjenje in induktivno logično programiranje, prav tako pa tudi pomožne komponente, ki omogočajo dostopnost teh tehnologij širši javnosti.

Možnosti analiz v realnem času in rudarjenje podatkovnih tokov je prikazano na primeru uporabe aktivnega učenja za dinamično prilagajanje analize sentimenta v podatkovnih tokovih. Uveljavljene tehnike za rudarjenje podatkovnih tokov smo uporabili po načelu vizualnega programiranja in prikazali na primeru uporabe analize sentimenta na podatkovnih tokovih kratkih spletnih sporočil.

Disertacija prispeva tudi k odprtokodni programski opremi. Vsi delotoki in izvorna koda opisanih primerov uporabe, platform ClowdFlows, ConCreTeFlows in TextFlows so javno dostopni. S tem omogočimo ponovljivost eksperimentov in prilagoditve ter izboljšave delotokov.

Contents

Abbreviations

ADC	… Annotated Document Corpus
AMQP	… Advanced Message Queueing Protocol
API	… Application Programming Interface
BOW	… Bag Of Words
DM	… Data Mining
GPU	… Graphics Processing Unit
GUI	… Graphical User Interface
IE	… Information Extraction
ILP	… Inductive Logic Programming
IR	… Information Retrieval
RDBM	… Relational Data Base Management
RDM	… Relation Data Mining
SCM	… Source Code Management
SOAP	… Simple Object Access Protocol
TF-IDF	… Term Frequency - Inverse Document Frequency
URL	… Universal Resource Locator
WSDL	… Web Services Description Language

Chapter 1

Introduction

We first describe the problem addressed in the thesis, explain the line of reasoning for choosing the given research topic and list the contributions to science. After defining the problem, we explain the purpose, hypotheses, goals and scientific contributions of the thesis. We describe the dissemination of software developed as part of this thesis. The chapter concludes with a structural overview of the thesis.

1.1 Problem Definition

Computer-assisted data analysis has gone a long way since its humble beginnings on first digital computers with stored programs. The 1962 paper entitled “The future of data analysis” [1] stated that the availability of electronic computers for some tasks is surprisingly ‘important but not vital’ and ‘vital’ for others. Without doubt, the situation has changed profoundly and any serious attempt to mine knowledge from real-world data must take advantage of modern computing methods and modern computer organization. However, while most scientists claim that the today’s main challenge in data mining is the size of data and their rate of production, it is often forgotten that the developments in the field of computer data analysis have also produced an almost unimaginable amount of methods, algorithms, software and architectures. They are available to anyone; however, their complexity and various requirements prevent the general public and especially domain specialists to use them effectively on their data without knowing the internal details.

This problem has been recognized years ago, and the development of modern programming languages, programming paradigms and operating systems initiated the research on computer platforms for data mining. Such platforms offer a high level of abstraction, which enables the users to focus their efforts on the analysis of results rather than on ways of obtaining them. In the beginning, single algorithms were implemented as complete solutions to specific data mining problems, e.g., the C4.5 algorithm [2] for the induction of decision trees. Later developments took advantage of the operating system independent languages such as Java to produce complete solutions which also include data preprocessing methods and visual representation of results. Today’s data mining platforms like Weka [3], RapidMiner [4], KNIME [5] and Orange [6] provide a large collection of generic algorithm implementations, usually coupled with an easy-to-use graphical user interface, following the visual programming paradigm [7].

Visual programming is an approach to programming where a procedure or a program is constructed by arranging program elements graphically instead of writing the program as text. This paradigm has also become widely recognized as an important element of an intuitive interface to complex data mining procedures. The popularity of existing knowledge discovery tools can be attributed to this paradigm as it eliminated the need for understanding the internal design and APIs, and has enabled expression of knowledge discovery processes as a sequence of processing steps in a visually appealing manner. Visual representations of sequences of processing steps are referred to as workflows in this thesis.

All advanced modern knowledge discovery systems offer some form of workflow construction and execution, as this is of crucial importance for conducting complex scientific experiments, which need to be reproducible and verifiable at an abstract level and through experimental evaluation. However, while the ability to execute data mining workflows and operating system independence was the most distinguished feature of data mining platforms a decade ago, today's data mining software is confronted with the challenge of implementing a service-oriented architecture [8], how to make use of newly developed paradigms for big data processing [9], and how to effectively employ modern technologies, such as a distributed cloud-based architecture [10].

Big data is widely regarded as data that is so large and complex that traditional tools and methods for data processing become insufficient [11]. Due to social media, sensors, internet of things, and user generated content, the rate at which data is produced is growing steadily, creating larger and larger streams of continuously evolving data. Steady growth of data leads to growing needs for specialized methodologies and open-source software for mining big data as a combination of two approaches: mining evolving data streams (low-latency processing) [12] and batch processing of static data of large proportions [13]. Low-latency processing requires processing of examples one at a time (and only once), using a limited amount of memory, working for a limited amount of time, and being ready to predict at any time [9]. On the other hand, a key design pattern in batch processing is the MapReduce programming model [14] for processing large data sets with parallel distributed algorithms on clusters by implementing solutions in two steps—map and reduce. Currently, there are no environments or frameworks that utilize both real-time low-latency processing and batch-processing on a single cluster that are available through a visual programming paradigm.

To summarize, knowledge discovery of big data, including evolving data streams, pose a problem for traditional data mining and knowledge discovery environments and platforms as does a thorough implementation of a service-oriented architecture. A novel data mining platform should not only enable a full utilization of cloud-based and service-oriented architectures that expose functional workflows as services, but also enhance the user experience with tools previously unavailable in big data processing systems. The platform should implement the visual programming paradigm and a way to share data, experiments and workflows through the web and social media.

1.2 Purpose of the Dissertation

The main purpose of the dissertation is to develop a framework and an implementation of a cloud-based knowledge discovery platform for creating, executing and sharing data mining workflows. With the developed framework we wish to address big data mining and real-time data analytics, and demonstrate its abilities and unique features which distinguish it from the existing solutions.

The developed platform described in this thesis, named ClowdFlows, is a new, open-source, web-based data mining platform that supports the design and composition of scientific procedures implemented as executable, reusable workflows. The platform is implemented as a cloud-based web application, meaning that it can be deployed on clusters of machines of varying sizes. The platform is simple to use (i.e. implements visual programming, is web-based and requires no installation), and enables workflow sharing. The platform enables combining workflow components (called 'widgets') from different software libraries to create previously unavailable unified workflows. The platform is capable of processing big data using MapReduce algorithms and processing infinite data streams by utilizing its always-online cloud-based architecture. Moreover, the platform is extensible in two ways: either the tools provided by the service-oriented architecture of the platform are used to create workflow components from web service endpoints during run-time, or new components can be developed from provided templates.

The dissertation also provides several novel knowledge discovery scenarios developed using the proposed methodology and the developed reference implementation of the platform, and their evaluation on real-world data mining tasks.

1.3 Goals of the Dissertation

The main goal of the dissertation is to develop a framework and a reference implementation of a cloud-based knowledge discovery and data mining platform for big data to fulfill the requirements of ever-growing heterogeneous streams of continuously evolving data. This reference platform integrates:

- a visual programming paradigm with a graphical user interface accessible from a web browser on a computer or mobile device,
- a framework for sharing data and workflows and making them publicly available to a broad audience,
- a framework for service-oriented computing enabling both consumption and generation of SOAP 1.1. compatible web services,
- a cloud-based architecture that allows distributed computing on a cluster of machines,
- an integration of a distributed file system that allows storing and processing of arbitrary large amounts of data with the MapReduce programming model applied to scientific workflows,
- a real-time processing system for the consumption and processing of heterogeneous data streams from the web.

The dissertation also aims to provide novel knowledge discovery scenarios that demonstrate the usefulness of the cloud-based knowledge discovery framework and the practical abilities of the implemented reference platform. The goal is to develop a framework that combines two modes of big data processing: batch processing and real-time analysis in real-world scenarios combining data stream mining and processing vast amounts of data using a unified graphical interface and hosted on the same cluster. These scenarios include:

- a knowledge discovery scenario featuring implementations of methods from several knowledge discovery platforms in a unified workflow environment,
- a relational data mining and an inductive logic programming scenario to exploit public, semantically annotated data sources,
- a streaming text mining scenario featuring various text mining components (language detection, stemming, lemmatization, document visualization, sentiment analysis from texts) and active learning on data streams.

1.4 Hypotheses

The main hypothesis of the thesis is that big data can be efficiently processed with a system that implements the visual programming paradigm and utilizes cloud computing by means of distributed hardware and software resources to improve scalability and adaptation to data of large proportions. We hypothesize that larger data sets can efficiently be processed and the through-put of the real-time data stream analysis can be increased to match the rate of incoming data by adding more computing power to the cluster of computing nodes.

The hypothesis of this thesis is also that the developed platform can be effectively utilized in real-life application scenarios and that the platform will prove to be useful also by 1) developing separate adapted versions of the platform, dedicated to different data processing scenarios, and 2) providing necessary tools and documentation for modification of the platform without the intervention of the authors. These modifications will simplify the knowledge discovery process to non-experts and scientists from different domains of research, e.g., biologists and natural language processing experts.

We hypothesize that by unifying functionalities of different knowledge discovery platforms and adding features to facilitate big data mining conjoined with real-time data analytics, we will be able to address novel scenarios which were not possible within the same platform until now, such as the introduction of the MapReduce programming model to scientific workflows and an active learning sentiment analysis.

1.5 Scientific Contributions

The main scientific contributions of the thesis are the following:

Contribution 1 A framework and a reference implementation of a web-based knowledge discovery platform ClowdFlows that encompasses:

- a browser-based workflow editor and facility for workflow sharing on the web,
- numerous data mining algorithms available as workflow components,
- a service-oriented architecture enabling an incorporation of remote web services into workflows and vice-versa, exposing workflows as stand-alone web services,
- cloud deployment of the platform and data analysis with distributed data,
- big data analytics in batch or real-time, including on-line processing of big data streams.

The work related to the development of the methodology and reference implementation of the web-based knowledge discovery platform is covered in Chapter 2 and was published in the following publications:

Journal Article

J. Kranjc, R. Orač, V. Podpečan, N. Lavrač, and M. Robnik-Šikonja, "ClowdFlows: Online workflows for distributed big data mining," *Future Generation Computer Systems*, vol. 68, pp. 38–58, 2016.

Conference Papers

J. Kranjc, V. Podpečan, and N. Lavrač, "ClowdFlows: A cloud based scientific workflow platform," in *Planning to Learn and Service-Oriented Knowledge Discovery, PlanSoKD'11*, Springer, 2012, pp. 816–819.

J. Kranjc, V. Podpečan, and N. Lavrač, “Real-time data analysis in ClowdFlows,” in *Proceedings of the 2013 IEEE International Conference on Big Data*, IEEE, 2013, pp. 15–22.

J. Kranjc, V. Podpečan, and N. Lavrač, “Knowledge discovery using a service oriented web application,” in *Proceedings of the 4th International Conference on Information, Process, and Knowledge Management*, IARIA, 2012, pp. 82–87.

J. Kranjc, V. Podpečan, and N. Lavrač, “A browser-based platform for service-oriented knowledge discovery,” in *Proceedings of the 23rd European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2011, pp. 30–35.

Contribution 2 A demonstrated extensibility of the platform to other fields of expertise. The adapted platforms include:

- a specialized platform for text mining and natural language processing,
- a dedicated platform for computational creativity.

The work related to the development of the adaptations of ClowdFlows for use in other fields of expertise covered in Chapter 3 was published in the following publications:

Journal Article

M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, and N. Lavrač, “TextFlows: A visual programming platform for text mining and natural language processing,” *Science of Computer Programming*, vol. 121, pp. 128–152, 2016.

Conference Papers

M. Perovšek, V. Podpečan, J. Kranjc, T. Erjavec, S. Pollak, N. Q. Do Thi, X. Liu, C. Smith, M. Cavazza, and N. Lavrač, “Text mining platform for NLP workflow design, replication and reuse,” in *Proceedings of the IJCAI Workshop on Replicability and Reusability in Natural Language Processing: From Data to Software Sharing*, AAAI Press, 2015, pp. 12–20.

M. Žnidaršič, A. Cardoso, P. Gervás, P. Martins, R. Hervás, A. O. Alves, H. G. Oliveira, P. Xiao, S. Linkola, H. Toivonen, J. Kranjc, and N. Lavrač, “Computational creativity infrastructure for online software composition: A conceptual blending use case,” in *Proceedings of the 7th International Conference on Computational Creativity*, ICCC, 2016, pp. 371–378.

Contribution 3 Workflow-based solutions for novel knowledge discovery scenarios, exploiting the features of the developed platform:

- Relational Data Mining (RDM) and Inductive Logic Programming (ILP) on the web, enabling the reuse of existing ILP components (covered in Chapter 4),
- on-line sentiment analysis from various big data sources supported by the active learning module (covered in Chapter 5).

The work related to the workflow-based solutions for novel knowledge discovery scenarios, exploiting the novel features of the developed platform was published in the following publications:

Journal Articles

J. Kranjc, J. Smailović, V. Podpečan, M. Grčar, M. Žnidaršič, and N. Lavrač, "Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform," *Information Processing & Management*, vol. 51, no. 2, pp. 187–203, 2015.

M. Perovšek, A. Vavpetič, J. Kranjc, B. Cestnik, and N. Lavrač, "Wordification: Propositionalization by unfolding relational data into bags of words," *Expert Systems with Applications*, vol. 42, no. 17, pp. 6442–6456, 2015.

Conference Papers

J. Smailović, J. Kranjc, M. Grčar, M. Žnidaršič, and I. Mozetič, "Monitoring the Twitter sentiment during the Bulgarian elections," in *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2015, pp. 1–10.

S. Pollak, A. Vavpetič, J. Kranjc, N. Lavrač, and Š. Vintar, "NLP workflow for on-line definition extraction from English and Slovene text corpora.," in *Proceedings of the 11th Conference on Natural Language Processing*, Österreichischen Gesellschaft für Artificial Intelligende, 2012, pp. 53–60.

1.6 Dissemination of the Developed Software

The reference implementation of the ClowdFlows platform as well as the related adapted platforms were released as open-source software and are available to the general public. The source code is released under the permissive MIT license.

A public installation of the ClowdFlows platform is hosted and maintained by the authors. At the time of writing the live installation of the platform hosts more than eight thousand workflows by more than a thousand registered users. The live installation is available at `http://www.clowdflows.org/`.

The source code is hosted on the popular Git repository hosting service GitHub at `https://github.com/xflows/clowdflows/`. It is available under the permissive MIT license. The platform has been forked[1] several times and is actively maintained by independent software developers and open-source enthusiasts. Instructions for installation of the platform are provided with the sources. Information about the workflow components and all necessary documentation required for adaptation of the platform is available at `http://clowdflows.readthedocs.io/`.

The sources of the Disco Machine Learning Library for machine learning with MapReduce paradigm are available under the permissive Apache 2.0 license at `https://github.com/romanorac/discomll/`.

The ConCreTeFlows platform is available at `http://concreteflows.ijs.si/`. The public roster of computational creativity workflows is available at `http://concreteflows.ijs.si/existing-workflows/`.

Likewise, the presented TextFlows platform is released under the MIT license, available at `https://github.com/xflows/textflows/`. Its documentation is available at `http:`

[1]According to the GitHub terminology, a fork is a copy of a repository. Forking a repository allows developers to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to the parent project or to use a project as a starting point for new software.

`//docs.textflows.org/`. A public installation of the platform is available for online use at `http://textflows.org/`.

The Wordification source code is accessible as part of the Python Relational Data Mining package implemented as ClowdFlows workflow components and is available at `https://github.com/xflows/rdm/` under the MIT license.

The workflows developed within the platform are publicly accessible in the public installation and their URLs may be published in scientific papers or shared via conventional methods.

1.7 Organization of the Thesis

In Chapter 1 we placed the thesis in the context of data mining and big data platforms and provided the problem definition as well as the hypotheses, goals and scientific contributions. The rest of the thesis is structured as follows.

Chapter 2 presents the ClowdFlows platform, which is a novel web-based data mining platform utilizing visual programming paradigms to allow creation and sharing of complex data mining workflows. The chapter begins with a description of related platforms and tools, and identifies their important features. The main part of this chapter is a journal publication which presents the related work, the platform itself, two big data mining aspects of the platform, and provides three use cases to demonstrate the developed features.

Chapter 3 describes the derivative works of ClowdFlows. It mentions three ClowdFlows adaptations and describes their purpose and usage. This chapter features a conference paper which describes ConCreTeFlows, a ClowdFlows adaptation for computational creativity, and a journal publication which describes the development of TextFlows, which is an adaptation of ClowdFlows focused on text mining.

Chapter 4 demonstrates how ClowdFlows can be used as a test bed for validating new methodologies by comparing them to existing algorithms in a unified workflow for validation. The journal publication in this chapter describes the wordification methodology which is a propositionalization technique used in relational data mining. In the publication, the methodology is validated using ClowdFlows.

Chapter 5 presents an active learning use case for sentiment analysis on data streams. This use case demonstrates the ability of ClowdFlows to run experiments indefinitely and continuous improvement of models via active learning.

Chapter 6 concludes the thesis and summarises the presented work while pointing out possible directions for further research.

Chapter 2

The ClowdFlows Platform

This chapter presents the ClowdFlows platform. First we describe related platforms and tools, and identify their important features and deficiencies. This is followed by a brief introduction to big data analytics. Both, the batch mode and the streaming mode of big data mining are presented. This serves as a motivation to design a new platform that tries to integrate the useful features and overcome some of the drawbacks of the existing platforms and implement functionalities for big data analytics. An overview of the platform design which highlights its most important features is presented. This is followed by a related journal publication that constitutes the core of the chapter. The enclosed paper describes the development of the platform in detail along with several use cases. The results and scientific contributions of the paper are highlighted and presented before its inclusion.

2.1 Data Mining Platforms

At the time when data mining algorithms, such as C4.5 [2] and CN2 [15], were invented and implemented, each implementation of a data mining algorithm was a standalone program and used incompatible file formats. This spawned a need for unified data mining platforms that allow practitioners to assemble different data mining algorithms into complex procedures known as data mining workflows.

In order to make use of a vast variety of existing data analysis methods, it is essential that such data mining platforms are easy and intuitive to use, allow quick and interactive changes to the analytic process and enable users to visually explore results. The platforms shall therefore follow a visual programming paradigm where the composition of a workflow is done visually by dragging and dropping components onto a canvas and adding connections which are represented by curves or lines on the canvas.

The early major data mining platforms, such as WEKA [3], RapidMiner [4], KNIME [5] and Orange [6], support workflow construction via graphical user interfaces. Their components typically include database connectivity, data loading from files, data and pattern mining algorithms, performance evaluation and visualizations. The platforms are installable applications and the requirement of the platforms to be installed on a particular hardware and software is in some cases considered a drawback.

Existing web-based workflow construction environments are mostly too general and not coupled to any data mining library. For example, Oryx Editor [16] can be used for modeling business processes and workflows, while the genome analysis tool Galaxy [17] (implemented as a web application) is limited to workflow components provided by the project itself. An exception is the ARGO project [18], where the aim was to develop an online workbench for analysis of textual data. It is based on a standardized architecture, supporting interactive scientific workflow construction and user collaboration through workflow sharing. ARGO provides a selection of data readers, consumers and some components for text analytics (mostly tagging, annotation and feature extraction). Finally, the OnlineHPC web application, which is based on the Taverna server as the execution engine, offers an online workflow editor, which is mostly a user-friendly interface to Taverna [19].

New tools were created to benefit from service-oriented architecture concepts, which utilize web services and access large public databases. Examples of such platforms are Weka4WS [20], Orange4WS [21], Web Extension for RapidMiner and Taverna [19], which allow integration of web services as workflow components. With the exception of Orange4WS and Web Extension for RapidMiner, most of these platforms are focused on specific scientific domains and do not offer general purpose machine learning and data mining algorithm implementations.

Remote workflow execution (on machines different from the one used for workflow construction) is employed by KNIME Cluster execution and RapidMiner using the RapidAnalytics servers. This allows data sharing with other users, with the requirement that the client software is installed on the user's machine. The client software is used for designing workflows, which are executed on remote machines, while only the results can be viewed using a web interface.

Grid workflow systems such as Pegasus [22], DAGMan [23] and ASKALON [24] were developed with the aim of simplifying intensive scientific processing of large amounts of data where the emphasis is on distribution of independent command line applications (grid jobs or tasks) and summarization of results. As the interactive analysis and graphical interfaces are not their most important features, some of them do not implement graphical interfaces to workflows but provide flexible programming interfaces instead.

Sharing data and experiments has been implemented in the Experiment Database [25], which is a database of machine learning experimentation results. Instead of a workflow engine it features a visual query engine for querying the database, and an API for submitting experiments and data.

Recent additions to the family of machine learning software are Google's TensorFlow [26] and the Machine Learning Service from Microsoft Azure [27]. TensorFlow is an implementation of machine learning algorithms on thousands of computational devices such as GPU processors. The system can be used for problems in speech recognition, computer vision, robotics, and natural language processing. The system lacks a conventional graphical user interface and is invoked as a software library. The Machine Learning Service from Microsoft Azure provides an easy-to-use and intuitive graphical user interface to construct data mining workflows on a canvas, but it is proprietary and requires users to subscribe to Microsoft's cloud services.

By examining the related work we have determined that the useful common features include: an implementation of the visual programming paradigm (i.e. a graphical user interface that allows construction of visual workflows), the ability to import web services as workflow components, the ability to share workflows and results, and executing data on different machines than the workflow is designed on. We also observe that there is a lack of interoperability between the platforms and implementations of algorithms in said platforms.

2.2 Big Data Mining

Big Data is a term used to identify data sets that, due to their large size, we can not manage with our current methodologies or data mining software tools [28]. The extraction of useful information from these large datasets or streams of data is called *Big Data Mining*.

The origin of the term ‘big data’ stems from the year 1998 when it was mentioned in a book about data mining [29]. The term originated due to the fact that we are creating large amounts of data every day.

The problem of Big Data management can best be described with 5 V’s [30]:

Volume There are more data than ever before, their size continues to increase, but not the percent of data that our tools can process.

Variety There are many different types of data, such as text, sensor data, audio, video, graph, and more.

Velocity Data is arriving continuously as streams of data, and we are interested in obtaining useful information from it in real time.

Variability There are differences in the structure of the data and how users want to interpret that data.

Value Business value that gives organizations a compelling advantage, due to the ability of making decisions based on answering questions that were previously considered beyond reach.

As a response to the ever increasing amounts of data several new distributed software platforms have emerged. In general, such platforms can be categorized into two groups: batch data processing and data stream processing.

2.2.1 Batch data processing

Batch data processing deals with analysis of data that is too big to use conventional methods. To perform analyses of such data new tools and software libraries have emerged. In this section we list selected relevant tools.

Apache Hadoop [13] is a tool for data-intensive distributed applications implemented in Java, based on the *MapReduce* programming model and a distributed file system called Hadoop Distributed Filesystem (HDFS). Hadoop allows writing applications that rapidly process large amounts of data in parallel on large clusters of machines. A MapReduce job divides the input dataset into independent subsets that are processed by *map* tasks in parallel. This step of mappings is followed by a step of *reducing* tasks. The reduce tasks use the output of the maps to obtain the final result of the job. Hadoop is used in many environments and several modifications and extensions exist, also for online (stream) processing [31], e.g., parallelization of several learning algorithms using an adaptation of MapReduce [32].

The Disco Project [33] is an alternative to Apache Hadoop. It is a lightweight open-source framework for distributed computing based on the MapReduce paradigm and written in Erlang and Python.

Apache Mahout [34] is a scalable machine learning and data mining open-source software based on Hadoop. It implements a wide range of machine learning and data mining algorithms: clustering, recommendation mining, classification, collaborative filtering and frequent pattern mining.

Apache Spark [35] is a recent alternative to Hadoop. Spark was developed to overcome Hadoop’s shortcoming that it is not optimized for iterative algorithms and interactive data analysis, which performs multiple operations on the same set of data.

Radoop [36] is a commercial big data analytics solution. It is based on RapidMiner and Mahout, and uses RapidMiner’s data flow interface.

2.2.2 Data stream mining

Data stream mining is the process of extracting knowledge structures from continuous, rapid data records [37]. A data stream is an ordered sequence of instances that can be read only once or a small number of times using limited computing and storage capabilities.

To solve the challenge of data stream mining several tools have emerged. In this section we briefly describe selected relevant tools.

Apache S4 [38] (Simple Scalable Streaming System) is a general-purpose, distributed, scalable, fault-tolerant, pluggable platform that allows programmers to develop applications for processing continuous, unbounded streams of data. The stream operators are defined by user code and the configuration jobs described with XML.

Apache Storm [12] is an open-source distributed real-time computation system similar to S4. The user constructs workflows in different programming languages such as Python, Java, or Clojure. Storm makes it easy to reliably process unbounded streams of data. Storm markets itself as doing for real-time processing what Hadoop does for batch processing.

MOA [9] is a stream data mining open-source software platform for performing data mining in real time. It has implementations of classification, regression, clustering and frequent item set mining and frequent graph mining. It started as a project of the Machine Learning group of University of Waikato, New Zealand (the authors of WEKA). MOA does not support visual programming of workflows but the ADAMS project [39] provides a workflow engine for MOA, which uses a tree-like structure instead of an interactive canvas.

SAMOA [10] is an example of a new generation platform that targets processing of big data streams. In contrast to distributed data mining tools for batch processing using MapReduce (e.g., Apache Mahout), SAMOA features a pluggable architecture on top of S4 and Storm for performing common tasks, such as classification and clustering. The platform does not support visual programming with workflows.

2.3 Development of the ClowdFlows Platform

The ClowdFlows platform was first concieved as a successor to the Orange4WS platform [21]. Orange4WS is a service-oriented data mining platform that is based on Orange [6]. The purpose of these platforms is to enable an easy composition of data mining workflows and to ease the sharing of experiments and results. These platforms achieve this purpose, but with several drawbacks.

To design and develop a suitable data mining platform we have reviewed the existing related platforms, libraries, tools and environments for knowledge discovery and identified all their distinguishing useful features. We have determined that there is a need for a data mining platform that offers the following features:

- remote execution of workflows,
- utilization of remote web services as workflow components,
- sharing workflows and reproduction of results of published experiments,
- mining continuous data streams, and
- processing big data that cannot be processed using conventional methods.

Since its inception the ClowdFlows platform distinguished itself from other platforms in the fact that it can be accessed from a web browser and does not need to be installed on users' computers [40]. This was achieved by designing ClowdFlows as a web application. The data model of ClowdFlows consists of *workflows*, workflow components (entitled *widgets*), *connections* and *users*. Each user can create a multitude of workflows that consist of multiple widgets and connections. All workflow, widget, and user data are stored in an arbitrary RDBM system. The platform features a graphical user interface that is accessible from a web browser and facilitates visual programming operations.

The platform can import any Web Service Description Language (WSDL) described web service as a workflow component [41]. A special interface integrated in the platform allows users to enter the URL of a web service during run time. Upon doing so, the WSDL is parsed by the platform and widgets are generated for corresponding web service operations. These widgets can be reused multiple times in the user's workflows.

The platform also features a roster of public workflows which can be contributed to by registered users. This allows for easy sharing of data mining workflows [42]. Workflows that are made public can be accessed by other users. Whenever a public workflow is accessed, a copy is made so that it can be modified by anyone without harming the original workflow. This allows for reproduction of experiments and results, and is a step towards solving the executable paper challenge [43].

To provide real-time analytics, the ability to mine infinite streams of data was added to the platform by introducing a different type of workflow that is executed simultaneously many times over to handle incoming data with a minimum latency [44]. This is achieved by a special mechanism that starts and stops workflow execution depending on the data. With this mechanism it is possible to create widgets that implement many classical stream mining approaches such as sliding windows, data aggregation, data selection and joining data instances in batches.

Finally, to provide a complete data mining platform with capabilities for mining big data, MapReduce data mining algorithms were added to the ClowdFlows platform [45]. This was achieved by implementing a new data mining library—DiscoMLL. The library uses the Disco framwork for big data mining and features re-implementations of data mining algorithms in the MapReduce paradigm. The library features bindings that can be exposed as ClowdFlows workflow components—widgets. By using these widgets in a workflow it is possible to construct big data mining workflows. It is important to note however that even though there are several widgets that do support big data mining using MapReduce, the majority of widgets in ClowdFlows still utilize data analysis in the classical sense.

2.4 Related Publication

The implementation details of the platform along with a review of related work, description of big data mining in batch mode and in streaming mode are described in the following publication:

J. Kranjc, R. Orač, V. Podpečan, N. Lavrač, and M. Robnik-Šikonja, "ClowdFlows: Online workflows for distributed big data mining," *Future Generation Computer Systems*, vol. 68, pp. 38–58, 2016.

In this publication we achieve the following:

- We review and present the existing platforms and environments for data mining with a visual programming paradigm and present libraries and tools for mining big data both in batch mode and in a streaming setting.

- We present a cloud-based platform for big data and stream processing with workflows entitled ClowdFlows. The design decisions taken to implement the platform are argumented and presented. The implementation is presented in detail.
- We show empirically that the ClowdFlows platform enables processing of multiple concurrent data streams.
- We present a novel use case that constructs a semantic graph from a stream of news articles in real time.
- Several machine learning algorithms were implemented in the MapReduce paradigm and made available for use in the ClowdFlows platform.
- We show that using all data in distributed mode is better than using a subset in non-distributed mode.
- We show empirically that the ClowdFlows platform handles big data sets with nearly perfect linear speedup.
- In each major section of the publication we present a novel use case that is implemented as a re-usable workflow. The use cases demonstrate the following: the use of data mining algorithms from different platforms in a unified way, mining continuous data sources and visualizing them in novel ways, applying a MapReduce algorithm on big data sets in a workflow-based environment.

The authors' contributions are as follows. The ClowdFlows platform was designed and implemented by Janez Kranjc with helpful insights from Vid Podpečan (the author of the Orange4WS platform). Janez Kranjc implemented the big data mining aspects of ClowdFlows (both in batch mode and streaming mode). The MapReduce implementations of various algorithms were implemented by Roman Orač and integrated in the DiscoMLL library. Nada Lavrač contributed the idea of developing a web-based platform for big data mining and supervised the work, while Marko Robnik Šikonja supervised the implementation of data mining algorithms in MapReduce mode and provided ideas for distributed ensemble methods.

Future Generation Computer Systems 68 (2017) 38–58

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

ClowdFlows: Online workflows for distributed big data mining

Janez Kranjc[a,b,*], Roman Orač[c], Vid Podpečan[a], Nada Lavrač[a,b,d], Marko Robnik-Šikonja[c]

[a] *Jožef Stefan Institute, Ljubljana, Slovenia*
[b] *Jožef Stefan International Postgraduate School, Ljubljana, Slovenia*
[c] *Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia*
[d] *University of Nova Gorica, Nova Gorica, Slovenia*

HIGHLIGHTS

- We present a cloud based platform for big data and stream processing with workflows.
- The ClowdFlows platform enables processing of multiple concurrent data streams.
- Several machine learning algorithms were implemented in the map-reduce paradigm.
- Using all data in distributed mode is better than using a subset in non-distributed.
- The ClowdFlows platform handles big data sets with nearly perfect linear speedup.

ARTICLE INFO

Article history:
Received 1 February 2016
Received in revised form
14 June 2016
Accepted 4 July 2016
Available online 8 September 2016

Keywords:
Data mining platform
Cloud computing
Scientific workflows
Batch processing
Map-reduce
Big data

ABSTRACT

The paper presents a platform for distributed computing, developed using the latest software technologies and computing paradigms to enable big data mining. The platform, called ClowdFlows, is implemented as a cloud-based web application with a graphical user interface which supports the construction and execution of data mining workflows, including web services used as workflow components. As a web application, the ClowdFlows platform poses no software requirements and can be used from any modern browser, including mobile devices. The constructed workflows can be declared either as private or public, which enables sharing the developed solutions, data and results on the web and in scientific publications. The server-side software of ClowdFlows can be multiplied and distributed to any number of computing nodes. From a developer's perspective the platform is easy to extend and supports distributed development with packages. The paper focuses on big data processing in the batch and real-time processing mode. Big data analytics is provided through several algorithms, including novel ensemble techniques, implemented using the map-reduce paradigm and a special stream mining module for continuous parallel workflow execution. The batch mode and real-time processing mode are demonstrated with practical use cases. Performance analysis shows the benefit of using all available data for learning in distributed mode compared to using only subsets of data in non-distributed mode. The ability of ClowdFlows to handle big data sets and its nearly perfect linear speedup is demonstrated.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Computer-assisted data analysis has come a long way since its humble beginnings on first digital computers with stored programs. In his 1962 paper entitled "The future of data analysis" [1] J. W. Tukey stated that the availability of electronic computers for some tasks is surprisingly "important but not vital" and "is vital" for others. Without doubt, the situation has changed profoundly and any serious attempt to mine knowledge from real world data must take advantage of modern computing methods and modern computer organization. However, while most scientists claim that the size of data and their rate of production is one of today's main challenges in data mining [2], it is often forgotten that developments in the field of data analysis have also produced an almost unimaginable amount of methods, algorithms, software and architectures. They are available to anyone, but their complexity and specific requirements prevent the general public and also research specialists to use them effectively without knowing the internal details. This problem has

* Corresponding author at: Jožef Stefan Institute, Ljubljana, Slovenia.
E-mail address: janez.kranjc@ijs.si (J. Kranjc).

http://dx.doi.org/10.1016/j.future.2016.07.018
0167-739X/© 2016 Elsevier B.V. All rights reserved.

been recognized years ago [3]; in his seminal work John Chambers noted that "there should not be a sharp distinction between users and programmers" and that language, objects, and interfaces are key concepts that make computing with data effective [4].

The development of modern programming languages, programming paradigms and operating systems initiated research on computer platforms for data analysis and development of modern integrated data analysis software. Such platforms offer a high level of abstraction, enabling the user to focus on the analysis of results rather than on the ways to obtaining them. In the beginning, single algorithms were implemented as complete solutions to specific data mining problems, e.g., the C4.5 algorithm [5] for the induction of decision trees. Second generation systems like SPSS Clementine, SGI Mineset, IBM Intelligent Miner, and SAS Enterprise Miner were developed by large vendors, offering solutions to typical data preprocessing, transformation and discovery tasks, and also providing graphical user interfaces [6]. Many of the later developments took advantage of the operating system independent languages such as the Java platform to produce complete solutions, which also include methods for data preprocessing and visual representation of results [7].

Visual programming [8], an approach to programming, where a procedure or a program is constructed by arranging program elements graphically instead of writing the program as a text, has become widely recognized as an important element of an intuitive interface to complex data mining procedures. All modern advanced knowledge discovery systems offer some form of workflow construction and execution, as this is of crucial importance for conducting complex scientific experiments, which need to be repeatable, and easy to verify at an abstract level and through experimental evaluation. The standard data mining platforms like Weka [7], RapidMiner [9], KNIME [10] and Orange [11] provide a large collection of generic algorithm implementations, usually coupled with an easy-to-use graphical user interface. While the operating system independence and the ability to execute data mining workflows was the most distinguished feature of standard data mining platforms a decade ago, today's data mining software is confronted with the challenge how to make use of newly developed paradigms for big data processing and how to effectively employ modern programming technologies.

This paper presents a data mining platform called ClowdFlows [12] and focuses on its capabilities of big data processing. ClowdFlows was developed with the aim to become a new generation platform for data mining, using the latest technologies in the implementation. It implements the following advanced features.

Most notably, the ClowdFlows platform runs as a web application and poses no software requirements to the users—e.g., ClowdFlows can also be accessed from modern mobile devices. Its server side software is easily multiplied and distributed to any number of processing nodes. As a modern data mining platform, ClowdFlows supports interactive data mining workflows which are composed, inspected and executed by using the ClowdFlows web interface. ClowdFlows workflows can be private or public, the later offer a unique way of sharing implemented solutions in scientific publications thus solving the Executable paper challenge [13]. Web services are supported and can be used as workflow components in an intuitive way. Processing of big data in batch mode is enabled by integrating the Disco framework, a lightweight, open-source framework for distributed computing using the map reduce paradigm [14]. In order to process big data in batch mode, we have developed a new machine learning library for the Disco MapReduce framework and included it in ClowdFlows. The library includes several standard algorithms as well as new ensemble techniques, which we evaluate and show the benefit of exploiting all available data in distributed setting compared to using only subsamples in a single node. Finally, ClowdFlows is developer-friendly as its server-side is written in Python, easily extensible, and supports distributed development using packages.

The ClowdFlows platform is released under an open source license and is publicly available on the web [12]. Users can choose either to use the publicly deployed version available at http://clowdflows.org, or clone the sources and deploy the system on their own machine or cluster of machines [15].

The rest of the paper is structured as follows. Section 2 presents the related work on data mining platforms and modern data processing paradigms. In Section 3 we present a motivational use case followed by the presentation of the design and implementation of the ClowdFlows platform. Section 4 presents the real-time analysis features of the ClowdFlows platform and demonstrates the ClowdFlows stream mining capabilities with a use case of dynamic semantic analysis of news feeds. The processing of big data in batch mode and the development of a machine learning library for the Map Reduce paradigm is presented in Section 5 which also includes a practical use case. Section 6 presents newly developed distributed random forest based ensemble methods. The batch processing mode is evaluated and validated in Section 7. In Section 8 the ClowdFlows platform is compared to related platforms and options for their integration are presented. Section 9 summarizes the work and concludes the paper by suggesting directions for further work. In the Appendix, summation form algorithms in DiscoMLL are described.

2. Related work

Visual construction and execution of scientific workflows is one of the key features of the majority of current data mining software platforms. It enables the users to construct complex data analysis scenarios without programming and allows easy comparison of different options. All early major data mining platforms, such as Weka [7], RapidMiner [9], KNIME [10] and Orange [11] support workflow construction. The most important common feature is the implementation of a *workflow canvas* where complex workflows can be constructed using drag, drop and connect operations with available components. The range of available components typically includes database connectivity, data loading from files and pre-processing, data and pattern mining algorithms, algorithm performance evaluation, and interactive and non-interactive visualizations.

Even though such data mining software solutions are user-friendly and offer a wide range of components, some of their deficiencies severely limit their utility. Firstly, all available workflow components are specific and can be used in only one platform. Secondly, the described platforms are implemented as standalone applications and have specific hardware and software dependences. Thirdly, in order to extend the range of available workflow components in any of these platforms, knowledge of a specific programming language is required. This also means that they are not capable of using existing software components, implemented as web services, freely available on the web.

In order to benefit from service-oriented architecture concepts, software tools have emerged, which are able to use web services and access large public databases. Environments such as Weka4WS [16], Orange4WS [17], Web Extension for RapidMiner and Taverna [18] allow the integration of web services as workflow components. However, with the exception of Orange4WS and Web Extension for RapidMiner, these environments are mostly focused on specific scientific domains such as systems biology, chemistry, medical imaging, ecology and geology and do not offer general purpose machine learning and data mining algorithm implementations.

Remote workflow execution (on machines different from the one used for workflow construction) is employed by KNIME Cluster execution and RapidMiner using the RapidAnalytics server. This allows the execution of local workflows on more powerful machines and data sharing with other users, with the requirement that the client software is installed on the user's machine. The client software is used for designing workflows, which are executed on remote machines, while only the results can be viewed using a web interface.

All the above mentioned platforms are based on technologies that are becoming legacy and do not benefit from modern web technologies, which enable truly independent software solutions. On the other hand, web-based workflow construction environments exist, but they are mostly too general and not coupled to any data mining library. For example, Oryx Editor [19] can be used for modeling business processes and workflows, while the genome analysis tool Galaxy [20] (implemented as a web application) is limited to workflow components provided by the project itself. An exception is the ARGO project [21], where the aim was to develop an online workbench for analyzing textual data based on a standardized architecture (UIMA), supporting interactive scientific workflow construction and user collaboration through workflow sharing, providing a selection of data readers, consumers and some components for text analytics (mostly tagging, annotation and feature extraction). Finally, the OnlineHPC web application, which is based on the Taverna server as the execution engine, offers an online workflow editor, which is mostly a user friendly interface to Taverna.

Grid workflow systems such as Pegasus [22], DAGMan [23] and ASKALON [24] were developed with the aim of simplifying intensive scientific processing of large amounts of data where the emphasis is on distribution of independent command line applications (grid jobs or tasks) and summarization of results. As the interactive analysis and graphical interfaces are not their most important features, some of them do not implement graphical interface to workflows but provide flexible programming interfaces instead. These platforms contain one or more grid middleware layers, which enable the execution on computer grids such as HTCondor and Globus.

Some platforms support more than one model of computation (see the analysis of workflow interoperability by Elmroth et al. [25]) and enable the use of web services, grid services as well as other execution environments (e.g., custom modules via foreign language interfaces). Kepler [26], Triana [27] and Taverna [18] are the most well known examples of such platforms.

As a response to the ever increasing amount of data several new distributed software platforms have emerged. In general, such platforms can be categorized into two groups: batch data processing and data stream processing.

A well known example of a distributed batch processing framework is Apache Hadoop [28], an open-source implementation of the MapReduce programming model [14] and a distributed file system called Hadoop Distributed Filesystem (HDFS). It is used in many environments and several modifications and extensions exist, also for online (stream) processing [29] (e.g., parallelization of several learning algorithms using an adaptation of MapReduce is discussed by Chu et al. [30]). Apache Hadoop is also the base framework of Apache Mahout [31], a machine learning library for large data sets, which currently supports recommendation mining, clustering, classification and frequent itemset mining. A more recent alternative to Hadoop is Apache Spark. Spark was developed to overcome Hadoop's shortcoming that it is not optimized for iterative algorithms and interactive data analysis, which performs multiple operations on the same set of data [32]. Radoop [33], a commercial big data analytics solution, is based on RapidMiner and Mahout, and uses RapidMiner's data flow interface. The Disco project [34] that we use is an alternative to Apache Hadoop. It is a lightweight open source framework for distributed computing based on the MapReduce paradigm and written in Erlang and Python.

For data stream processing, two best known platforms are S4 [35] and Storm [36]. The S4 platform is a fully distributed real-time stream processing framework. The stream operators are defined by the user code and the configuration jobs described with XML. Storm is a stream processing framework that focuses on guaranteed message processing. The user constructs workflows in different programming languages such as Python, Java, or Clojure. Neither of these two platforms features an easy to use graphical user interface.

SAMOA [37] is an example of a new generation platform that targets processing of big data streams. In contrast to distributed data mining tools for batch processing using MapReduce (e.g., Apache Mahout), SAMOA features a pluggable architecture on top of S4 and Storm for performing common tasks, such as classification and clustering. The platform does not support visual programming with workflows. MOA (Massive On-line Analysis) is a non-distributed framework for mining data streams [38]. It is related to the Weka project and a bi-directional interaction of the two is possible. MOA does not support visual programming of workflows but the ADAMS project [39] provides a workflow engine for MOA, which uses a tree-like structure instead of an interactive canvas.

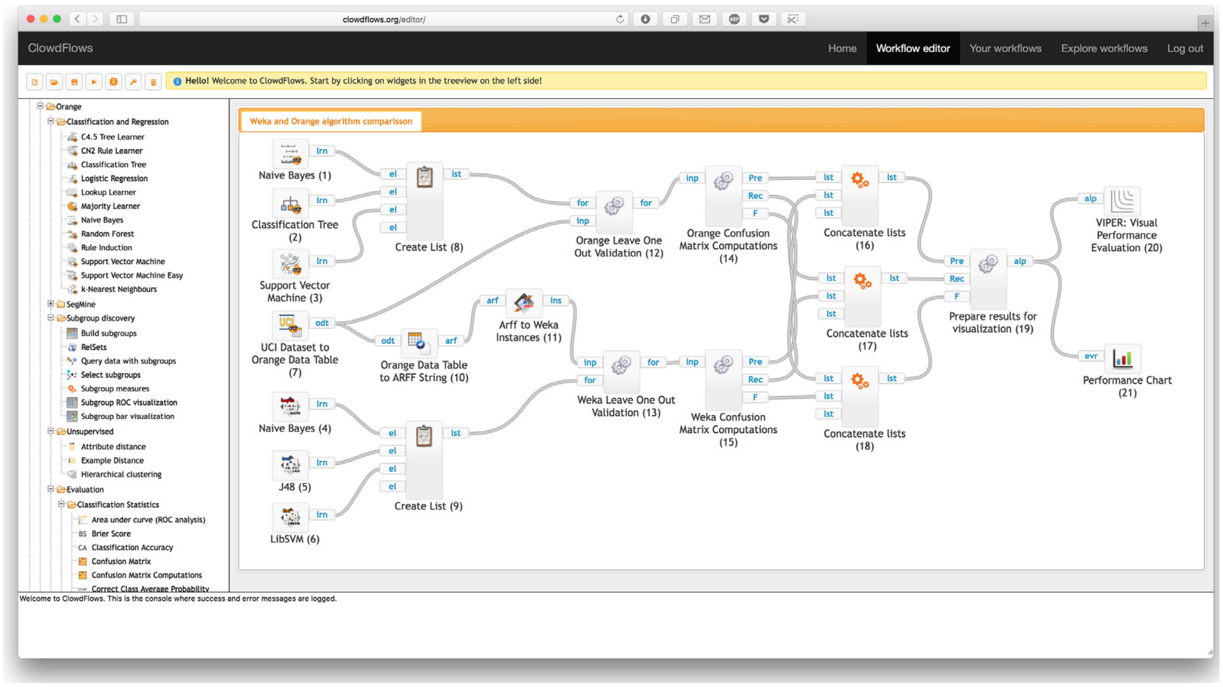
Sharing data and experiments has been implemented in the Experiment Database [40], which is a database of standardized machine learning experimentation results. Instead of a workflow engine it features a visual query engine for querying the database, and an API for submitting experiments and data.

Substantial efforts have also been invested in developing systems for streamlining experimentation and data analysis in multi agent based systems, particularly for game-playing, in distributed systems [41]. In such systems distributed-computing applications can be customized by a human monitoring expert who controls the execution of an experiment through a web-based graphical user interface.

More recent additions to the family of machine learning software are Google's TensorFlow [42] and the Machine Learning Service from Microsoft Azure [43]. TensorFlow is an implementation for executing machine learning algorithms on thousands of computational devices such as GPU cards. The system can be used to express a wide variety of algorithms for problems in speech recognition, computer vision, robotics, and natural language processing. However, the system lacks a conventional graphical user interface and is invoked as a software library. The Machine Learning Service from Microsoft Azure provides an easy to use and intuitive graphical user interface to construct data mining workflows on a canvas, but it is proprietary and requires the user to subscribe to Microsoft's cloud services.

3. ClowdFlows platform

Software technologies that were used to implement the ClowdFlows platform allow an easy integration of very diverse programming libraries and algorithm implementations. Various wrappers allow the Python programming language environment to connect to software written in Java, C, C++, C#, Fortran, etc. Several libraries for web service interoperability are also available. The ClowdFlows platform currently integrates three major machine learning libraries: Weka [7], Orange [11] and scikit-learn [44]. Integration of Orange and scikit-learn is native as both are written in Python/C++. Weka algorithms are implemented as web services using the JPype [45] wrapper library.



Hello! Welcome to ClowdFlows. Start by clicking on widgets in the treeview on the left side!

Welcome to ClowdFlows. This is the console where success and error messages are logged.

Fig. 1. A ClowdFlows workflow for comparison of algorithms from two different machine learning libraries (Weka and Orange). Algorithms are evaluated on a UCI data set using the leave one out cross validation and their performance is visualized using VIPER charts. This workflow is publicly available at http://clowdflows.org/workflow/6038/.

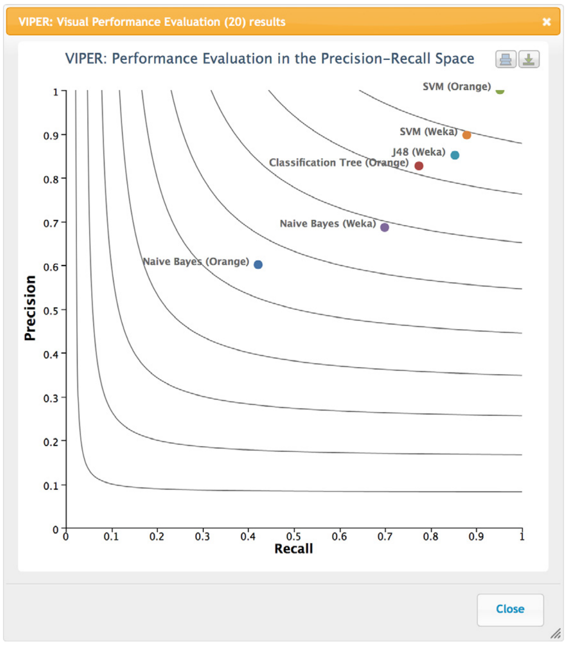


Fig. 2. Visual performance evaluation of several machine learning algorithms implemented in ClowdFlows.

3.1. Illustrative example

We first demonstrate the use of the platform with an illustrative use case followed by the design and architecture of ClowdFlows.

The goal of this simple use case is to present a few basic features of ClowdFlows. To this end, we have developed a workflow for evaluating and comparing several machine learning algorithm implementations. Decision tree, Naive Bayes and Support Vector Machines algorithms from Weka and Orange are evaluated with the leave-one-out cross-validation method on several publicly available data sets from the UCI repository [46] and the results of the evaluation are presented using the VIPER (Visual Performance Evaluation) [47] interactive performance evaluation charts. The interactive workflow demonstrates the use of web services as workflow components, as the employed Weka algorithms have been made available as web services.

The sample workflow for the evaluation and comparison of several machine learning algorithm implementations is shown in Fig. 1. This workflow is publicly available at http://clowdflows.org/workflow/6038/.

The workflow performs as follows. First, instances of the selected machine learning algorithm implementations are created from the libraries (Weka and Orange) and concatenated into a list. Second, a UCI data set is selected, loaded, and transformed into two different data structures, one for each library. Validation is performed using the leave-one-out method on three pairs of algorithm implementations from different libraries. Confusion matrices are computed, and the results are prepared for visualization. In the final step, the VIPER chart (Visual performance evaluation) is shown. The chart offers interactive visualization of algorithm results in the precision–recall space thus allowing a visual comparison of several algorithms and export of publication quality figures. This performance visualization is shown in Fig. 2.

The public ClowdFlows installation features many other example workflows including workflows that demonstrate regression[1] and clustering.[2]

3.2. Platform architecture and technologies

ClowdFlows is a cloud-based web application that can be accessed and controlled from anywhere using a web browser, while the processing is performed in a cloud of computing nodes.

The architecture of the ClowdFlows platform is shown in Fig. 3. The platform consists of the following components: a graphical user interface, a core processing server, a database, a

1 http://clowdflows.org/workflow/7539/.
2 http://clowdflows.org/workflow/7492/.

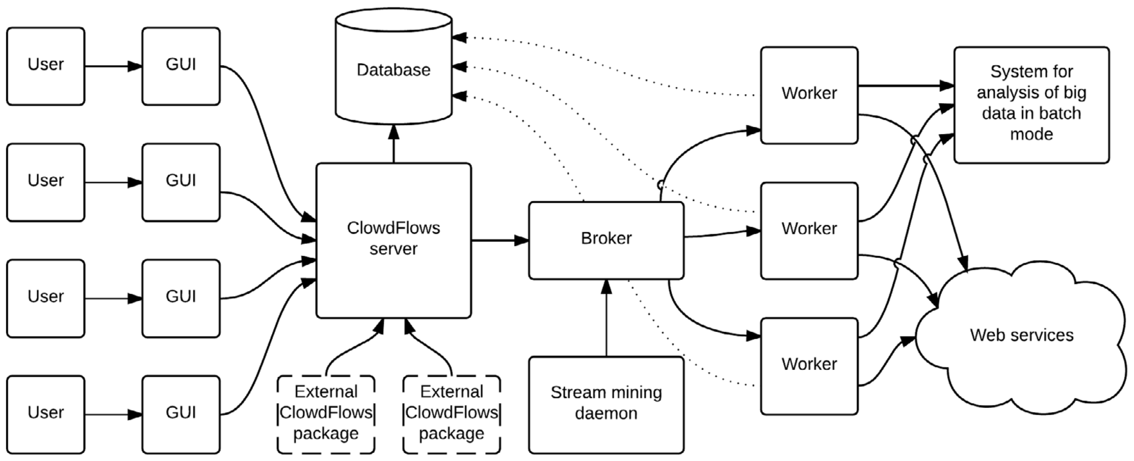


Fig. 3. An overview of the ClowdFlows platform architecture. A similar figure (without the big data analytics components) has appeared in our previous publication [48].

stream mining daemon, a broker that delegates execution tasks, distributed workers, web services, and a module for big data analysis in batch mode.

3.2.1. Graphical user interface

Users interact with the platform primarily through the graphical user interface in a web browser. We implemented the graphical user interface in HTML and JavaScript, with an extensive use of the jQuery library [49]. The jQuery library was designed to simplify client-side scripting, and is the most popular JavaScript library in use [50]. The user interface is served from the primary ClowdFlows server.

As illustrated in Fig. 1, the graphical user interface provides a workflow canvas where workflow components (widgets) can be added, deleted, re-positioned and connected with one another to form a coherent workflow. The graphical user interface is synchronized with the ClowdFlows server using asynchronous HTTP requests, which notify the server of any and all user actions.

The job of the graphical user interface is to also render results in a meaningful representation. Each widget that can produce visualized results does so by sending the represented data in HTML and JavaScript to the graphical user interface, which in turn shows it to the user in a non-obtrusive pop-up dialog.

Aside from the workflow construction and results visualization capabilities, this layer of the application also displays a list of public workflows that can be copied by the currently logged in user.

All the graphical user interface code resides on the server but is executed in the users' browsers.

3.2.2. ClowdFlows server

The ClowdFlows server software is written in Python and uses the Django web framework [51]. The Django framework is a high level Python web framework that encourages rapid development and provides an object-relational mapper and a powerful template system.

The ClowdFlows server consists of a web application and the widget repository.

The web application defines the models, views, and templates of ClowdFlows. The integral part of the ClowdFlows platform is the data model, which consists of an abstract representation of workflows and widgets. Workflows are executable graphical representations of complex procedures. A workflow in ClowdFlows is a set of widgets and their connections. A widget is a single workflow processing unit with inputs, outputs and parameters. Each widget performs a task considering its inputs and parameters, and stores the results of the task to its outputs. Connections are used to transfer data between two widgets and may exist only between an output of a widget and an input to another widget. Data is transferred through connections, so inputs can only receive data from connected outputs. Parameters are similar to inputs, but need to be entered by the user. Inputs can be transformed into parameters and vice-versa, depending on the user's needs.

Data from the data model are stored in the database. It should be noted that there are two representations of widgets in the data model and database. The *abstract widget* is a description of a specific widget in the repository and holds no other information than the inputs, outputs, parameters and function of the widget. The non-abstract widget is a separate entity that represents a particular instance of an abstract widget and contains information about the data inputs and outputs, and its spatial position in a specific workflow. To summarize, when the user constructs a workflow, she chooses from a set of abstract widgets to create instances of non-abstract widgets that can process data. This design decision was made to allow user customization of widgets and to ensure the functionality of workflows, even when the abstract widgets change over time.

The ClowdFlows application also implements a workflow execution engine which executes widgets in the workflow in the correct order. The engine issues tasks to the workers to execute widgets. Initially only the widgets that have no predecessors are executed (a predecessor is a widget that is connected on the input of a widget). Once these widgets have successfully executed the engine searches for widgets whose predecessors have been successfully executed and issues tasks to the workers to execute them. When there are no more widgets to execute, a workflow is considered successfully executed.

The ClowdFlows widget repository consists of four groups of widgets: regular widgets, visualization widgets, interactive widgets and workflow control widgets.

Regular widgets are widgets that take data on the input and return data on the output. Visualization widgets do the same with the addition that they provide an HTML/JavaScript view which can be rendered by the graphical user interface to display results. Interactive widgets are widgets that serve a view to the user during execution time. Interactive widgets can show data on the input to the user and can process the user interaction in its function which affects the data on the output.

Workflow control widgets are special widgets that allow creation of subworkflows (workflows encapsulated in widgets that contain a workflow), creation of *for loops* and special types of for loops that are used for cross validation. With the workflow control widgets, the workflows can also be exposed as REST API services. These REST API services provide HTTP endpoints that can be called from anywhere to invoke the execution of a workflow and fetch the results.

The functions of regular widgets, visualization widgets, and interactive widgets are stored in the widgets libraries. The widget libraries are packages of functions that are called when a widget is executed. These functions define the functionality of each particular widget.

By default, ClowdFlows comes with a set of widgets that can be expanded. The initial set of widgets encompasses solutions to many data mining, machine learning and other tasks such as: classification, clustering, regression, association rule learning, noise detection, decision support, text analysis, natural language processing, inductive logic programming, graph mining, visual performance evaluation, and others.

3.2.3. Database

The database is the part of the system that stores all the information about the workflows and all the user uploaded data.

The object-relational mapper implemented in Django provides an API that links objects to a database, which means that the ClowdFlows platform is database agnostic. PostgreSQL, MySQL, SQLite and Oracle databases are supported. MySQL is used in the public installation of ClowdFlows. The database installation can be deployed on a cluster to ensure scalability of the system.

3.2.4. Worker nodes

Worker instances are instances of the ClowdFlows server that do not serve the graphical user interface and are only accessed by a broker that delegates execution tasks. They execute workflows and workflow components. The workers report success or error messages to the broker and feature timeouts that ensure fault tolerance if a worker goes offline during the run-time. The number of workers is arbitrary and they can be connected or disconnected during run-time to ensure scalability and robustness. Workers subscribe to the message broker system, which can be deployed on multiple machines. The ClowdFlows system offers support for several message broker systems. RabbitMQ [52] is used in the ClowdFlows public installation.

3.2.5. Web services

In order to allow consumption of web services and import them as workflow components, the PySimpleSoap library [53] is used. PySimpleSoap is a light-weight library written in Python and provides an interface for client and server web service communication, which allows importing WSDL (Web Service Definition Language) web services as workflow components, and exposing entire workflows as WSDL web services.

3.2.6. Scaling and process distribution over cloud resources

The ClowdFlows platform can scale horizontally in a very straight forward way. The four components that need to be scaled are the ClowdFlows server, the database, the broker, and the worker nodes.

Scaling the ClowdFlows server is done simply by installing it on multiple machines and running it behind a web server with load balancing capabilities such as Nginx. Horizontally scaling the ClowdFlows server is required when there are many simultaneous users accessing the platform at once. The requests are then routed round robin to different ClowdFlows server instances to reduce the load.

As the ClowdFlows platform is database agnostic it is entirely dependent on the scaling ability of the selected database software. Popular database solutions such as MySQL and PostgreSQL can be transformed into distributed scaled-out systems, however as the ClowdFlows platform does not normally perform demanding database operations (apart from simple insertions and selections) this is not likely to require scaling.

The scalability of the broker also depends on the choice of its implementation. Both RabbitMQ and Redis, which are popular broker solutions, can be easily deployed into clusters where nodes are added and removed. Scaling the broker is required if the data that passes from one workflow component to another is large.

Similarly to the ClowdFlows server, the worker nodes (which are just headless instances of the ClowdFlows server) can easily be executed in parallel on multiple machines as long as they all connect to a single broker (or broker cluster). The broker ensures that tasks are distributed evenly across the workers. Increasing the number of workers is by far the most frequently required scaling operation. Each worker can execute a set number of widgets in parallel at any given time. If there are more workflows executed than available workers some workload will be delayed until workers finish tasks of active workflows.

The scaling of the stream mining deamon and the batch processing module is handled separately and is explained in Sections 4 and 5, respectively.

3.3. Public workflows

Since workflows in the ClowdFlows platform are processed and stored on remote servers they can be accessed from anywhere over the internet. By default, each workflow can only be accessed by its author. We have implemented an option that allows users to create public versions of their workflows.

The ClowdFlows platform generates a URL for each workflow that is defined as public. Users can share their workflows by publicizing this URL. Whenever a public workflow is accessed by the user, a copy of the workflow is created on the fly and added to the user's private workflow repository. The workflow is copied with all the data to ensure the reproducibility of experiments. Each such copied public workflow can be edited, augmented or used as a template to create a new workflow, which can be made public as well.

3.4. Extensibility and widget development

There are two ways to add widgets to the ClowdFlows platform. A widget can be either implemented as a Python function and included in a ClowdFlows package or be manually imported via the graphical user interface as a WSDL Web service.

In the ClowdFlows platform the widgets are grouped into packages. Each package consists of a set of widgets with common functionalities. A package can be bundled with the code of the platform or released as a separate Python package, which can be installed on demand. An example of such a package is the Relational Data Mining (RDM) package for ClowdFlows.[3] This package exists as a stand alone Python package but can also be included in ClowdFlows. Upon doing so the widgets are automatically discovered by the ClowdFlows platform and added to the repository.

Creating a custom package for ClowdFlows requires the developer to have ClowdFlows installed locally. The local installation of ClowdFlows provides several command line utilities for dealing with packages. These utilities create bare-bones packages that include skeleton code for widgets. A ClowdFlows widget is a Python function that receives a dictionary of widget inputs on the input and returns a dictionary of outputs as its output. The body of the function needs to be filled in by the developer of the widget to transform the inputs into the outputs as expected. The widget's inputs and outputs need to be described in the JSON format in order to be shared with other installations of ClowdFlows (including

3 https://github.com/xflows/rdm.

the public installation). This JSON file can be written manually or by using the ClowdFlows administration interface, which provides simple forms where widget details can be entered. The JSON files are then generated from the database using command line utilities bundled with the local installation of ClowdFlows.

In a multiple worker setting each worker node needs to have all the available external packages installed. Fig. 4 shows a worker node with external packages installed and its JSON description of widgets imported using the administration tools.

Another way of adding widgets is by using the graphical user interface and entering the URL of a WSDL described Web service. This can be done without altering the code of the platform which makes it easy to use. The Web service description file is consumed by ClowdFlows and parsed to determine the inputs and outputs of the functions of the Web service. Each function of a Web service is represented as a single widget in ClowdFlows that can be used and reused in any number of workflows by the user that imported the service.

3.5. Data exchange with external systems

The ClowdFlows platform provides several ways of exchanging data with external systems. We distinguish between two types of functionalities: the inward and outward interoperability.

Inward interoperability is achieved either by consuming web services and presenting them to the users as widgets of the ClowdFlows platform, or by creating widgets that call code from other systems. By default the ClowdFlows platform consumes web services that provide functionalities of the Weka platform, and provides widgets that access code and data from the Orange and scikit-learn packages.

Outward interoperability allows any workflow to be exposed as a REST API endpoint. In order to benefit from this feature each workflow can have several API Input and API Output widgets on the canvas. These widgets are linked with the inputs that the REST API endpoint receives and the JSON output of results that it should return. In this way it is possible to construct a workflow in the ClowdFlows platform and execute it without using the graphical user interface, which makes it suitable for use in external applications.

4. Real-time data stream mining

Processing of real-time data streams is enabled in ClowdFlows: a specialized stream mining deamon was implemented that continuously executes workflows in parallel with a modified workflow execution engine that implements a halting mechanism. The stream mining capabilities of the ClowdFlows platform are described below.

4.1. Stream mining workflows and stream mining deamon

Stream mining workflows are workflows that are connected to a potentially infinite source of incoming data and need to be executed whenever there is new data on the input. Due to the nature of online data sources, it is often necessary to poll a data source for new data instead of having the data source push the data to external services such as ClowdFlows. To control execution of stream mining workflows we implemented a special deamon that executes workflows at a fixed time interval and provided a functionality to halt the execution of a workflow to stream mining widgets.

The stream mining deamon is a process that runs alongside the ClowdFlows server, loops through deployed stream mining workflows and executes them. The execution is similar to the regular workflow execution with the difference that widgets may

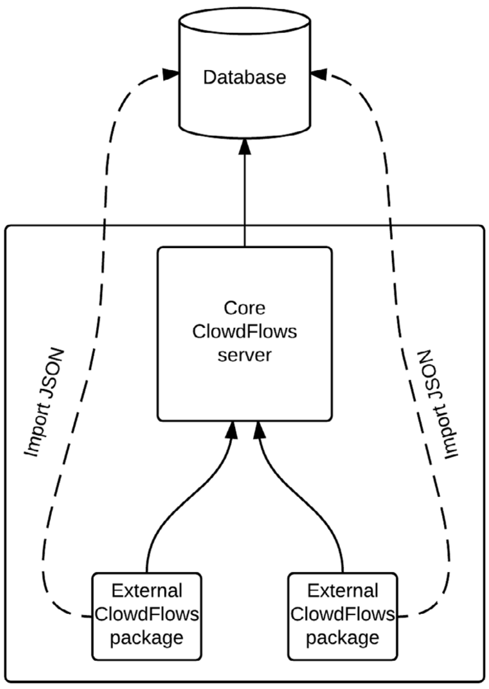


Fig. 4. A worker node of the ClowdFlows architecture with two external packages installed.

halt the execution of workflows. In practice, the workflow is executed as frequently as the data appears on the data source and produces outputs with a fixed latency depending on the workflow complexity. Stream mining workflows are, in contrast to regular workflows, executed a potentially infinite number of times until the execution is stopped by the user.

4.2. Stream mining widgets

In contrast to widgets in regular workflows, widgets in stream mining workflows have the internal memory and the ability to halt the execution of the current workflow. The internal memory is used to store information about the data stream, such as the timestamp of the last processed data instance, or an instance of the data itself. These two mechanisms were used to develop several specialized stream mining widgets.

In order to process data streams, *streaming data inputs* were implemented. Each type of stream requires its own widget to consume the stream. In principle, a streaming input widget connects to an external data stream source, collects instances of the data that it has not yet seen, and uses its internal memory to remember the current data instances. This can be done by saving small hashes of the data to preserve space or only the timestamp of the latest instance if timestamps are available in the stream. If the input widget encounters no new data instances at the stream source it halts the execution. No other widgets that are directly connected to it via its outputs will be executed until the workflow is executed again.

Several popular stream mining approaches [54] were implemented as workflow components. The *aggregation* widget was implemented to collect a fixed number of data instances before passing the data to the next widget. The internal memory of the widget is used to save the data instances until the threshold is reached. While the number of instances is below the threshold, the widget halts the execution. The internal memory is emptied and the data instances are passed to the next widget once the threshold is reached.

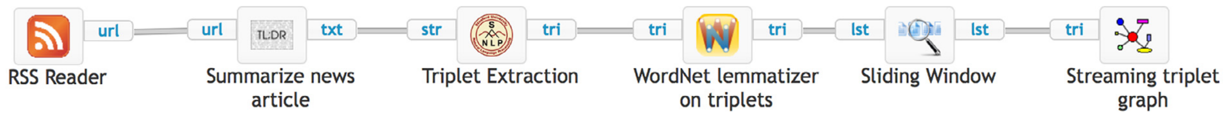


Fig. 5. The semantic triplet graph from an RSS feed workflow constructed in the ClowdFlows platform. The workflow is publicly available at http://clowdflows.org/workflow/1729/.

The *sliding window* widget is similar to the aggregation widget, except that it does not empty its entire internal memory upon reaching the threshold. Only the oldest few instances are *forgotten* and the instances inside the sliding window are released to other widgets in the workflow for processing. By using the sliding window, each data instance can be processed more than once.

Sampling widgets either pass the instance to the next widget or halt the execution, based on a condition. This condition can be dependent on the data or not (e.g., drop every second instance). The internal memory can store counters, which are used to decide which data is part of the sample.

Special *stream visualization* widgets were developed for the purpose of examining results of real-time analyses. Each instance of a stream visualization widget creates a web page with a unique URL that displays the results in various formats. This is useful because the results can be shared without having to share the actual workflows.

4.3. Illustrative use case

The aim of this use case is to construct a semantic graph from a stream of news articles in real time.

A semantic triplet graph [55] is a graph constructed from *subject–verb–object* triplets extracted from sentences. Our goal is to develop a reusable workflow that allows extraction of triplets from an arbitrary source of news articles, displays them in a two-dimensional force directed graph with words as nodes and updates them in real-time. By doing this, we transform an incoming stream of news articles into a live semantic graph that is continuously updated.

The use case is presented as a step-by-step report on how the workflow was constructed. Following this description the user can construct a fully functional workflow for an arbitrary stream of data and examine the results. In this particular workflow the Middle East section of the CNN news website is used as the incoming stream on which the semantic graph is constructed.

4.3.1. Identification and development of necessary workflow components

In order to produce a workflow that transforms an incoming stream of news articles into a semantic triplet graph we require three components: a component that connects to the RSS feed and fetches new articles as they appear, a component that extracts triplets from the articles, and a component that the triplets in the graph form. In order to create a visually appealing and useful semantic graph we decided to create three supporting widgets: a widget that fetches the article text and summarizes it by selecting five most important sentences from the article, a widget that normalizes the triplets by performing lemmatization on the extracted words, and a sliding window to force the graph to “forget” older news.

We implemented a widget that connects to an arbitrary RSS feed. This widget accepts a single parameter: the URL of the RSS feed. The internal memory of widgets was utilized to store hash codes of article URLs that have already been processed. With this we ensure that each article is only processed once. If all URLs in the feed have already been processed the widget halts the execution. The widget’s single output is the URL of the article that should be processed. This widget was implemented as a Python function as explained in Section 3.4. The function has access to the argument (the URL of the RSS feed) and to the internal memory of the widget which is persistent for a particular execution of a stream. The function uses a high level Python library *requests* for fetching the data and parsing the feed.

We implemented a widget that fetches the article text from the URL, extracts the article’s title and body, and summarizes it by selecting five most important sentences in the article. We developed a widget that wraps the PyTeaser library for text summarization [56]. The most important sentences are selected based on their relevance to the title and keywords, as well as the position and length of the sentences. The widget outputs the summary as a string of characters.

The triplet extraction widget implements the algorithm proposed in [57] using the Stanford Parser [58]. The widget first tokenizes the sentences and generates a parse tree for each sentence. The algorithm searches the parse tree for the subject, predicate, and object triplet. If the triplet is found, it is appended to the list of triplets that the widget returns as its output. The triplet extraction widget was implemented as a WSDL Web service which was imported into the ClowdFlows.

Normalization of words helps in building a more cohesive graph by joining similar nodes into a single node. We employ the WordNet Lemmatizer implemented in the Python NLTK library [59,60]. The lemmatization uses the WordNet’s built-in morphy function. The input words are returned unchanged if they cannot be found in WordNet. This technique helps to eliminate repetitions of similar or same entities in the graph. As NLTK is a Python library, we implemented this widget as a Python function as explained in Section 3.4.

To dynamically visualize the semantic triplet graph we use a sliding window widget to keep only the 100 recent triplets. By doing this the graph only shows current news and “forgets” older news.

The visualization widget utilizes the D^3 Data-Driven Documents JavaScript library [61] to display the semantic triplet graph. The graph is constructed by creating a node for each unique word in the current sliding window. Edges are constructed from the subject–verb and verb–object connections. The graph is rendered using a force-directed algorithm [62]. The visualization is updated in real-time, new nodes and edges are created in real-time and old ones are removed from the graph. Visualization widgets also render an HTML view, which allowed us to use a JavaScript library to implement the visualization. Anything that can be displayed on a web page can be displayed as a result of a visualization widget.

All the developed widgets were added to the repository and are part of the stream mining ClowdFlows package. For the sake of simplicity the Triplet Extraction Web service call was wrapped into a Python function otherwise new installations of ClowdFlows would require users to manually import it as a Web service.

4.3.2. Constructing the workflow

We constructed the workflow using the ClowdFlows graphical user interface. Widgets were selected from the widget repository, added to the canvas and connected as shown in Fig. 5.

In the RSS reader widget we entered the URL of the CNN news section feed—http://rss.cnn.com/rss/edition_meast.rss. We

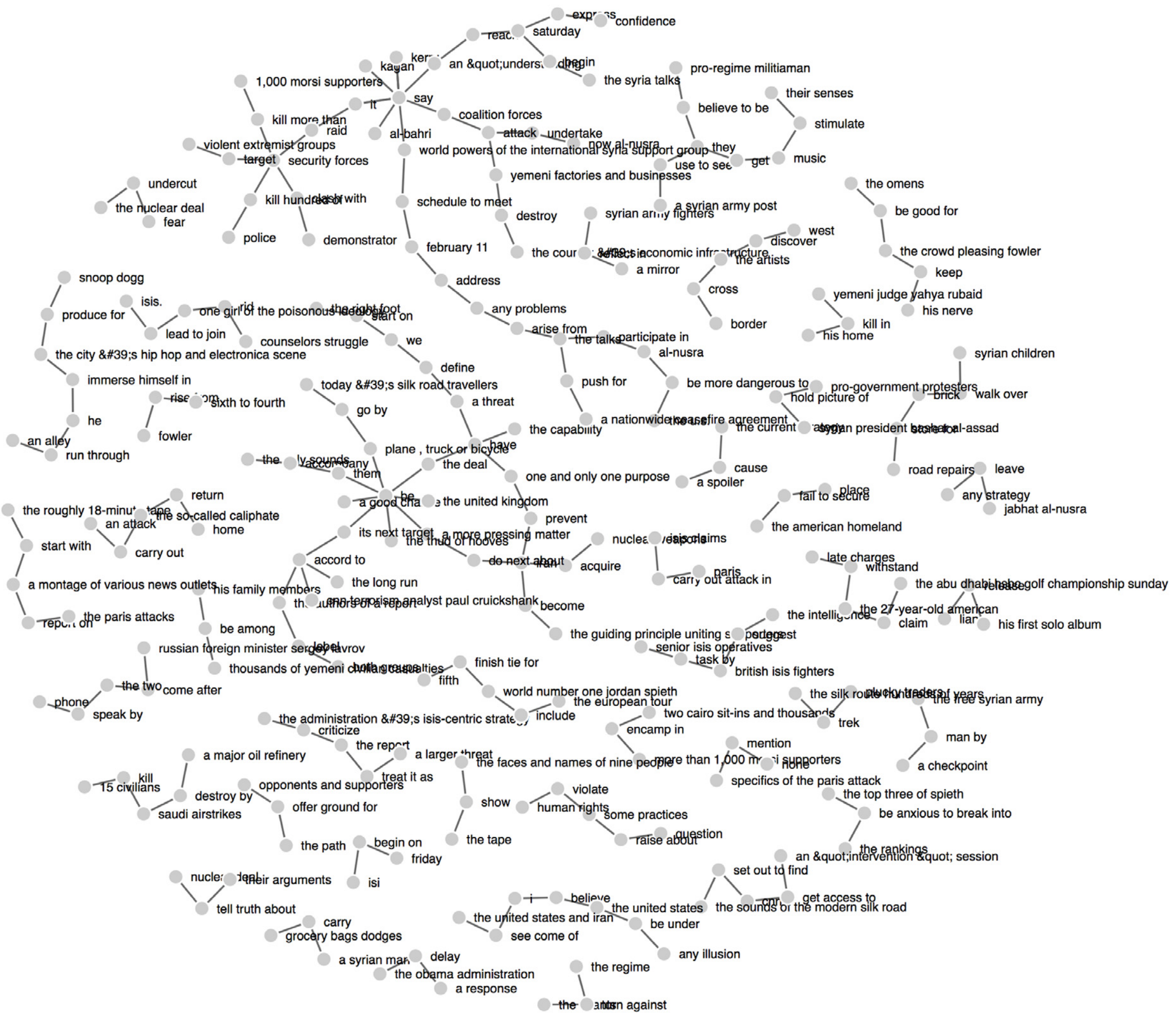


Fig. 6. The results of monitoring the Middle East edition of the CNN RSS feed. This visualization is publicly available at http://clowdflows.org/streams/data/110/63907/.

have set the size of the sliding window to 100, thereby showing the latest 100 news triplets in the graph.

We marked the workflow as public so that it can be viewed and copied. The URL of the workflow is http://clowdflows.org/workflow/1729/. By clicking the button "Start stream mining" on the workflows view (http://clowdflows.org/your-workflows/) we instructed the platform to start executing the workflow with the stream mining daemon. A web page is created with detailed information about the stream mining process. This page contains the link to the visualization page with the generated semantic graph.

4.3.3. *Monitoring the results*

By using a stream visualization widget in the workflow we can observe the results of the execution in real time. The ClowdFlows platform generates a web page for each stream visualization widget in the workflow.

Our workflow has only one stream visualization widget (see Fig. 5), therefore the ClowdFlows platform generates one web page with the results. The visualization of the CNN data stream can be found at http://clowdflows.org/streams/data/110/63907/. This visualization shows how the words in the most important sentences of multiple articles are linked to each other via the extracted *subject–verb–object* triplets. A screenshot of the semantic triplet graph is shown in Fig. 6.

The workflow presented is general and reusable. The RSS feed chosen for processing is arbitrary and can be trivially changed. The results of the workflow can be further exploited by performing additional data analysis on the graph, such as constructing a summary of several news articles or discovering links between them. Such a graph could be used to recommend related news to the reader.

4.3.4. *Evaluating the limitations of stream mining in ClowdFlows*

Stream mining workflows are executed by the Stream mining deamon, which is a separate process that exists with the sole purpose of executing stream mining workflows on a set time interval. For streams with a high rate of incoming data this time interval has to be set in such way that the rate of processing the data is higher or equal to the production rate of the data on the inputs, while producing the results with a fixed latency (the amount of time it takes to execute a workflow).

Streaming workflows usually run for a fixed amount of time. Having the execution interval (frequency) shorter than the workflow execution time (latency) means that the workflow for the same stream will be executed many times in parallel. Since ClowdFlows is a collaborative platform with many concurrent executions of stream mining workflows it is important to know the limitations of such executions so that it is possible to determine when to add new resources to ensure all data is processed. We have identified two possible scenarios when executing stream mining workflows: workflows that process data faster than the data is produced, and workflows that process data slower than it is produced.

Table 1
Average data analysis time for streaming data based on different incoming rates of data instances and different setups of worker nodes. The cells display the average processing time for each data instance in seconds (plus the standard deviation) and the relative amount of data instances processed during a minute of stream execution time.

Data spawn rate/Worker nodes	1×8	2×8	3×8
0.33 instances per second	3.597 ± 0.005 (100%)	3.603 ± 0.007 (100%)	3.602 ± 0.008 (100%)
1 instance per second	3.596 ± 0.006 (100%)	3.597 ± 0.009 (100%)	3.600 ± 0.014 (100%)
3 instances per second	7.702 ± 2.096 (85%)	3.599 ± 0.010 (100%)	3.600 ± 0.009 (100%)
6 instances per second	20.049 ± 8.220 (44%)	7.673 ± 2.061 (76%)	3.618 ± 0.010 (100%)

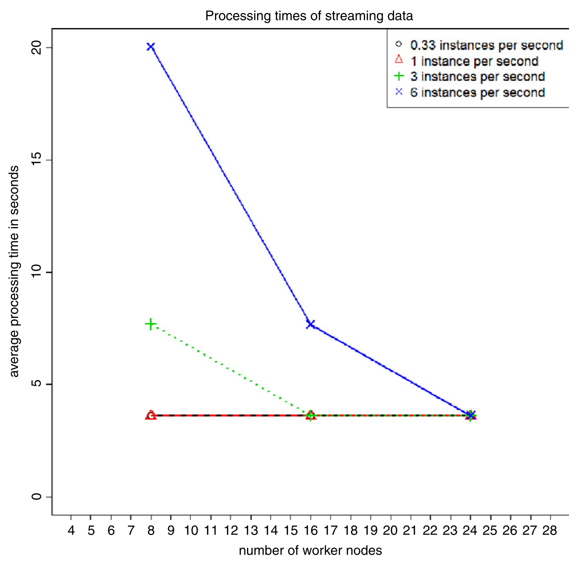


Fig. 7. Average data analysis times for streaming data based on different incoming rates of data instances over different setups of worker nodes.

For cases where the latency is shorter than the amount of time it takes for new data to appear it is clear that these workflows will never be executed in parallel. Each workflow will process the data before there is new data on the input. Even with a very small time interval the subsequent parallel executions would be immediately halted due to no data being present on the input, and thus this would not affect the resources.

We conducted an experiment where we created artificial streams with several production rates and processed them in a workflow with a latency of 3 s per data instance. We tested the stream mining capabilities of ClowdFlows against different rates of data production on the inputs: 0.33 instance per second, 1 instance per second and 3 instances per second for different setups of worker nodes. For each setting we adjusted the frequency to ensure processing of data.

We tested the platform with one, two, and three worker nodes. Worker nodes were installed on equivalent computers with 8 CPU cores. The workers were setup to work on 8 concurrent threads. We measured the processing time for processing each instance of data and calculated the relative amount of data processed in a minute in percentages. A hundred percent means that all the data on the input stream was successfully processed. The results are presented in Table 1 and displayed on the chart in Fig. 7.

The results show that using a single worker it is possible to process data at a rate three times slower than the rate of production on the input and preserve the same throughput as on the input stream. Using two workers it is possible to process data ten times slower, while a configuration with three workers allows for a twenty times slower processing rate and still cope with the demand. This allows users to have complex workflows perform analyses on the data and still see results in real time while being confident that all data was processed.

It is important to note that inter-node communication does not increase with the addition of new worker nodes. Worker nodes communicate exclusively with the broker. Tasks are issued to the broker by the stream mining daemon and results are returned to the broker by the worker nodes. Even if there is a single worker node processing the stream, the communication is always going to and from the broker. The communication delay is therefore constant.

The test also shows that the leniency for slow processing can be improved by adjusting the number of worker nodes. The worker nodes can be added and removed during runtime, which means that processing high volume streams can be resolved simply by adding more computing power to the ClowdFlows worker cluster.

5. Batch data processing with DiscoMLL library

We present the ClowdFlows system for analysis of big data in batch mode. We have chosen the Disco MapReduce framework to perform MapReduce tasks as Disco is written in Python and allows for a tighter and easier integration with the ClowdFlows platform. The downside of this choice is an apparent lack of a specialized machine learning library or toolkit within the framework, which motivated us to develop our own library with a limited but useful set of machine learning algorithms.

In this section we first introduce the MapReduce paradigm, followed by a description of the Disco Framework. We describe the implementation details of the Disco Machine Learning Library. Finally we present the integration details of batch big data processing in ClowdFlows and conclude with an illustrative use case.

5.1. *MapReduce paradigm*

MapReduce is a programming model for processing large data sets, which are typically stored in a distributed filesystem. Algorithms based on the MapReduce paradigm are automatically parallelized and distributed across the cluster. The user of MapReduce paradigm defines a map function and a reduce function. The map function takes an input key/value pair and generates a set of intermediate key/value pairs. Intermediate keys are grouped and passed to the reduce function. An iterator is used to access the intermediate keys and values. The reduce function merges these values and usually forms a smaller set of values. The output of a MapReduce job can be used as the input for the next job or as the result.

5.2. *Disco framework*

Disco is designed for storage and large scale processing of data sets on clusters of commodity server machines. It provides fault-tolerant scheduling, execution layer, and a distributed replicated storage layer. Core aspects of cluster monitoring, job management, task scheduling and distributed file system are implemented in Erlang, while the standard Disco library is implemented in Python. Activities of Disco cluster are coordinated by a central master host, which handles computational resource monitoring and allocation, job and task scheduling, log handling, and client interaction. A distributed MapReduce job executes multiple tasks, where each task runs on a single host. The job scheduler assigns resources to jobs, minimizes the data transfer over network, takes care of load balancing and handles changes in cluster topology.

Disco Distributed Filesystem (DDFS) provides a distributed storage layer for Disco. It is a tag based filesystem designed for storage and processing of massive amounts of immutable data. DDFS provides data distribution, replication, persistence, addressing and access.

5.3. *Disco Machine Learning Library*

To the best of our knowledge there is currently no Python package with machine learning algorithms based on the MapReduce paradigm for Disco, which motivated the development of the Disco Machine Learning Library (DiscoMLL) [63]. DiscoMLL is an open source library, build on NumPy [64] and the Disco framework. DiscoMLL is a part of the ClowdFlows platform and is responsible for the analysis of big batch data. This enables ClowdFlows users to process big batch data using visual programming. DiscoMLL provides many options for data set processing: multiple input data sources, feature selection, handling missing data, etc. It supports several data formats: plain text data, chunked data on DDFS and gzipped data formats. Data can be accessed locally or via file servers.

To take advantages of the MapReduce paradigm, it is necessary that algorithms have certain properties. As shown in [65], machine learning algorithms that fit the statistical query model [66] can be expressed in so called *summation form* and distributed on a multi-core system. An example is an algorithm that requires statistics that sum over the data. The summation can be done independently on each core by dividing the data, assigning the computation to multiple cores and at the end aggregating the results.

All implemented algorithms have a fit phase and predict phase. The fit phase consists of map and combine tasks, which are parallelized across the cluster. Usually algorithms have one reduce task that aggregates outputs of map tasks and returns URL of the fit model, which is stored on DDFS. The learned model differs for each algorithm, since it contains the actual model and the parameters needed for the predict phase. The predict phase consists only of map tasks which are also parallelized across the cluster. The first step of the predict phase is to read the model from DDFS and pass it as parameter to map tasks. Then the data is read and processed with the model. The predict phase stores predictions on DDFS and outputs the URL.

We have implemented the following algorithms which are in summation form: Naive Bayes, Logistic Regression, K-means clustering, Linear regression, Locally weighted linear regression, and Support Vector Machines. Implementations of these algorithms can be found in the Appendix.

To assure enough algorithms with state-of-the-art performance [67] we also implemented several existing and new variants of ensemble methods adapted to distributed computing paradigm. The proposed ensembles are based on decision trees, which are not in summation form. These methods are presented in Section 6.

5.4. *Integration in the ClowdFlows platform*

The batch mode big data analysis in ClowdFlows is a separate module that is accessed by the worker nodes via an HTTP interface of the Disco cluster. This configuration is presented in Fig. 8.

To configure a Disco cluster, it is only necessary to set slave server hostnames or IP addresses, and the number of their CPU cores. All nodes of the cluster are connected to the Internet to access the input data on file servers. An input data is transferred to workers using HTTP GET requests and passed directly to map tasks. The ClowdFlows platform submits MapReduce jobs using HTTP

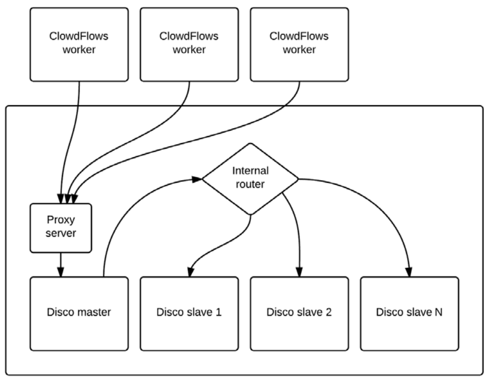


Fig. 8. An overview of the ClowdFlows system for batch mode big data analysis.

interface via a proxy server to the Disco master host. In contrast to input data, results of MapReduce jobs are stored on DDFS and their locations are passed back to the ClowdFlows platform.

Widgets that submit MapReduce jobs were developed which allow construction of workflows for big data. Each widget calls the Disco master using an HTTP interface. The workflows designed with these widget do not differ in presentation from workflows that deal with regular data. The MapReduce paradigm and implementation methods are not obvious from the workflows themselves.

5.5. *Use case: Naive Bayes classifier for big data*

In order to demonstrate the batch big data processing mode of ClowdFlows we have implemented a simple workflow that is capable of processing data sets that do not fit into memory of conventional machines.

The aim of the use case is to build a classifier from a large test set and to use it to classify data. We first describe the implementation details of the Naive Bayes classifier in the Disco Machine Learning Library, then we follow it with a description of a ClowdFlows workflow that utilizes the DiscoMLL.

5.5.1. *Naive Bayes in Disco Machine Learning Library*

The basic form of the Naive Bayes (NB) classifier uses discrete features. It estimates conditional probabilities $P(x_j = k|y = c)$ and prior probabilities $P(y)$ from the training data, where k denotes the value of discrete feature x_j and c denotes a training label. The map function (Algorithm 1) takes the training vector, breaks it into individual features and generates output key/value pairs. Each output pair contains the training label i.e., value of y, feature index j and feature value of x_j as the key and 1 as the value. This output pair marks the occurrence of a feature value given training label. The map function also outputs the training label as the key and 1 as the value, to mark the occurrence of a training label. The map function is invoked for every training instance. The reduce function (Algorithm 2) takes the iterator over key/value pairs. Values with the same key are grouped together in the intermediate phase. If the key consist of one element, it represents an instance's label and values are summed and stored for further calculation of the prior probability. The values of other pairs are summed and output. These pairs represent the occurrences of $x_j = k \wedge y = c$ and enable calculation of conditional probabilities $P(x|y)$ in the predict phase. After all pairs are processed, prior probabilities are calculated and output. The output of the reduce function presents a model that is used in the predict phase. The outputs of the predict phase were compared with the Orange implementation of the Naive Bayes Classifier [11] and return identical results.

As an example, consider the NB classifier with the input data set in Table 2, where the target label is *Sex*. At the beginning of the

Table 2
Example data set of the human physical appearance.

Sex	Hair length	Height
M	Short	Tall
M	Short	Tall
F	Long	Medium

MapReduce job, each training instance is read and passed to the map function as its input argument (*sample*). For the first training instance, the *sample* assigns $x = [Short, Tall]$ and $y = M$. The for loop iterates through x and outputs pairs $((M, 0, Short), 1)$ and $((M, 1, Tall), 1)$. The value 1 is added to mark one occurrence of the specific feature value and training label. The map function also outputs the pair $(M, 1)$ to mark the occurrence of label M. The procedure is repeated for all training instances. Note that the first two training instances in Table 2 are the same and produce the same output pairs. Prior to invocation of the reduce function, the output pairs are grouped by the key. We get the following pairs: $((M, 0, Short), [1, 1])$, $((F, 0, Long), [1])$, $((M, 1, Tall), [1, 1])$, $((F, 1, Medium), [1])$, $(M, [1, 1])$, $(F, [1])$. Notice that values from the first and second training instance are grouped by the key and their counts are merged in a list [1, 1]. The *iterator* over pairs is passed to the reduce function. For each key, the values are summed. The keys that mark training label occurrences are used to calculate prior probabilities, others are output in the form of $((M, 0, Short), 2)$. These values constitute the model that is used in the predict phase.

Algorithm 1: The map function of the fit phase in the NB

```
function map(sample, params)
    x, y = sample
    for j = 0 to length(x)
        #key: label, attr index and value
        #value: 1 occurrence
        output((y, j, x_j), 1)
    #mark label occurrence
    output(y, 1)
```

Algorithm 2: The reduce function of the fit phase in the NB.

```
function reduce(iterator, params)
    y_dist = hashmap()
    for key, values in iterator
        if key has 1 element
            #label frequencies
            y_dist.put(key, sum(values))
        else if key has 3 elements
            #occurrences x_j = k and y = c
            output(key, sum(values))
    prior = calculate_prior_probs(y_dist)
    output("prior", prior) #P(y)
```

For numeric attributes, a NB classifier uses a different approach to probability estimation. We provide a description of this method in the Appendix.

5.5.2. *Constructing the workflow*

We constructed a workflow that analyzes a big data set using the Naive Bayes machine learning algorithm based on the MapReduce paradigm.

For the purpose of this use case, we generated 6 GB of semi-artificial data [68] from the UCI image segmentation data set. The training and testing data set each contain 3 GB of data. Both data sets were divided into chunks and loaded on a file server. The components are connected as shown in Fig. 9.

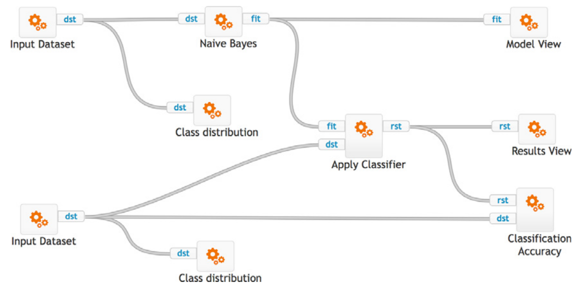


Fig. 9. The workflow for learning the NB classifier, predicting unseen instances and displaying the evaluation results. The workflow can be accessed at http://clowdflows.org/workflow/2788/.

The ClowdFlows workflow consists of training a model on the image segmentation training data set with NB, using the model to predict testing instances and visualization of the results.

First, we set the input parameters specifying the image segmentation train data object. We entered multiple URLs to process the data in parallel. The feature index parameter was set to 2–21, to include all the features available in the data set. The identifier attribute was set to 0 and the target label index was set to 1 as it represents the target label. In the widget that represents the prediction data object, we entered the URLs of the corresponding chunks. The Naive Bayes widget learns the model. The Apply Classifier widget takes the prediction data and the model's location to predict the data. The Results View shows the results, the Class Distribution widget shows the distribution of labels in the data set, the Model View widget enables us to review the statistics of the model and the Classification Accuracy widget is used to calculate the accuracy of the classifier. By pressing the button "start", the workflow executes a series of MapReduce jobs on the cluster. After the workflow is finished the results are provided as a hosted file on the distributed file system.

6. Distributed ensemble methods for batch processing

Ensemble methods are known for their robust, state-of-the-art predictive performance [67]. As we want to assure high usability of ClowdFlows we developed several tree-based ensembles adapted for distributed computation with MapReduce and implemented them in DiscoMLL. We first describe the ideas behind the most successful ensemble method, random forest [69], then we review existing distributed ensembles, followed by our methods.

Random forest is one of the most robust and successful data mining algorithms [70]. It is an ensemble of randomized decision trees used as the basic classifiers. Two randomization mechanisms are used: a bootstrap sampling with replacements on the training set, separately for each tree, and random selection of a subset of attributes in each interior node of the tree. A notable consequence of using bootstrap sampling with replacement for selection of training sets is that on average $1/e \approx 37\%$ of training instances are not selected in each tree (so called out-of-bag set or OOB). This set can be used for unbiased evaluation of the model's performance and its visualization.

Tree-based ensembles can exploit distributed computing in two ways: either computing basic models independently on local subsets of instances stored in worker nodes or computing individual trees with several nodes. The first approach is used in the MReC4.5 [71] and COMET [72] systems, while the second is used in the PLANET [73]. These systems are not publicly available, so direct comparison with them is not possible.

The MReC4.5 system [71] implements a variant of bagging where the master node bootstrap samples the training instances and distributes them to local nodes, where C4.5 like decision trees [74] are constructed in the map step and returned to the master node in the reduce step. In the prediction phase all trees return their votes. The weakness of this approach is memory consumption as worker nodes operate on the data set the size of the whole data set and keep it in its internal memory.

The COMET system [72] uses non-overlapping data samples in worker nodes. During the map phase many trees are created in each node using small training sets obtained with importance-sampled voting [75]. Importance-sampled voting uses OOB set to steer the training set sampling for consecutive trees in the same worker node. A large collection of trees are returned to the master node (the reduce phase). During prediction only a subset of trees is used, depending on the agreement of already returned votes.

The PLANET system [73] constructs each tree from all data in a distributed fashion. To keep the number of map steps low and to evaluate attributes in several nodes at once it creates trees level by level instead of in a depth-first manner as usual.

We developed three tree-based ensemble methods using the MapReduce approach. To allow processing of big data we split data sets to chunks that fit into local memory of worker nodes. All three methods construct decision trees on local nodes in the map phase and gather collected trees into a forest in the reduce phase. We implemented a binary decision tree learning algorithm, which runs on a single worker and expands decision tree nodes using a priority queue. The algorithm allows different types of attribute sampling in interior tree nodes and offers several types of attribute evaluation functions, e.g. information gain [74] and MDL [76]. As the trees are constructed locally in workers we need a single map step.

In the reminder of the section we present the three methods. Their evaluation is included in Section 7.2.

6.1. *Forest of Distributed Decision Trees*

The first variant, called Forest of Distributed Decision Trees (FDDT), performs a distributed variant of bagging [77]. Instead of bootstrap sampling of the whole data set it builds decision trees on chunks of training data. In each interior decision tree node all attributes are evaluated and the best one is selected as the splitting criterion. In the prediction phase all trees are used and their majority vote is returned as a prediction.

6.2. *Distributed Random Forest*

The second variant, called Distributed Random Forest (DRF), is a distributed variant of the random forest algorithm [69]. It uses bootstrap sampling with replacement on local data chunks to construct the training sets. In each interior node a random subsample of attributes is evaluated and the best one is selected as the splitting criterion. In the prediction phase a subset of trees is randomly selected and used for prediction. If the difference between the most probable prediction and the second most probable prediction is larger than a pre-specified parameter the most probable prediction is returned, otherwise more trees are selected and the process is repeated. The process ends when the difference is large enough or all the trees have been used for prediction. This process speeds up the prediction phase by using only a small subset of trees for prediction of less difficult instances.

6.3. *Distributed Weighted Forest*

The third variant, called Distributed Weighted Forest (DWF), is based on the idea that not all trees perform equally well for each instance, so it weights the trees for each prediction instance separately, extending the idea of [78] to a distributed environment. The construction phase of randomized trees is the same as in DRF, but after each tree is constructed, it is used to predict class values of its OOB instances. If two instances are classified into the same leaf node, their similarity score is increased. In this way we get a similarity score for all instances stored in a worker node, which we can divide by the number of trees in a worker t to get a $[0, 1]$ normalized distance [69]. The distances are passed to the k-medoid clustering algorithm, which returns medoids (instances, whose average distance to all instances in the same cluster are minimal). The default value for the number of clusters k is set to $\lceil\sqrt{a}+1\rceil$, where a is the number of attributes. Larger values of k improve the prediction accuracy, but also increase the prediction time. Medoids are used to determine prediction reliability of trees. A reliability score used is the prediction margin, defined as a difference between the predicted probability of the correct class and the most probable incorrect class. This margin is used to weight trees during the prediction phase. The DWF algorithm returns a local (small) forest for each worker node, the medoids, and reliability scores for each tree in the local forest.

During the prediction phase we first compute the similarity between a test instance and all the medoids in all the forests using the Gower coefficient [79], which is defined for both nominal and numeric attributes. The medoids with the highest similarity to the test case are used as tree quality probe. Only trees with positive and large enough reliability score (larger than median) for each medoid are used and their prediction scores are weighted with the reliability scores. The class with the highest sum of weighted predictions is returned. The described prediction process aims to improve the prediction by using only trees that perform well on instances similar to the given new instance.

To reduce memory consumption of k-medoid algorithm and the space required to store instance similarities, we sample the instances used in the tree reliability estimation process. The size of the sample is a parameter of the method.

7. Evaluation of big data processing in ClowdFlows

To validate our implementation of batch processing of big data we evaluated the big data processing in ClowdFlows by first empirically proving that our map-reduce implementations of algorithms are equivalent in performance to their standard counterparts implemented in widely used libraries. Next we use large data sets and assume that data sets is too big to fit into the main memory of a single worker we test their performance in distributed environment and on assumption that the data set is to big to fit into the main memory of a single worker

We also validated our newly developed distributed ensemble methods by comparing them to bagging and random forests implemented in the scikit-learn toolkit [44]. Additionally, we test their performance in distributed environment and on assumption that the data set is to big to fit into the main memory of a single worker.

7.1. *Evaluation of DiscoMLL summation form algorithms*

In order to verify the quality of implemented algorithms we compared them with standard single processor based implementations from the scikit-learn toolkit [44]. We used 10 relatively small data sets from UCI repository [80] and 3 big data sets. The characteristics of data sets are presented in Table 3.

We first tested DiscoMLL algorithms based on statistical queries, which, although implemented in a distributed fashion,

Table 3
The characteristics of small UCI data sets (above the line) and big data sets (below the line). The labels in column header have the following meaning: D = number of discrete attributes, R = number of numeric attributes, C = number of class values, N = number of instances, NC = number of chunks used in distributed processing, TC = number of trees per chunk for ensemble methods, Cl = the majority class used in binary classification.

Data set	D	R	C	N	NC	TC	Cl
abalone	1	7	29	2 087	18	30	9
adult	8	6	2	24 420	18	30	≤50K
car	6	0	4	864	18	30	unacc
isolet	0	617	26	3 899	18	30	17
segmentation	0	19	7	1 155	18	30	sky
semeion	256	0	10	796	18	30	1
spambase	0	57	2	2 300	18	30	–
wilt	0	5	2	2 418	18	30	–
wine-white	0	11	10	2 448	18	30	6
yeast	1	8	10	740	18	30	cyt
covertype	54	0	7	290 506	8	10	2
epsilon	0	2000	2	250 000	65	3	–
mnist8m	0	784	10	4 050 000	119	3	1

follow the same principles as non-distributed implementations and shall achieve the same accuracy.[4]

For MapReduce methods we use a distributed computational environment with 10 nodes (a master node and 9 worker nodes). Each computational node is a 2 CPU AMD Opteron 8431 2.4 GHz with 1 GB RAM using Ubuntu 12.04, so in total we have 18 concurrent processes and each is assigned approximately 1/18 of training instances.

On small data sets 5 × 2 cross-validation was used to test the performance of algorithms. Each training set was randomly split into 18 chunks matching 18 processors in our testing scenario. The results are collected in Table 4. For each algorithm we present two scores: in the left-hand columns the scores of scikit-learn using the whole training set are given and in the right-hand columns the results of DiscoMLL are presented, where each of 18 workers received 1/18 of the training data. We observe that distributed algorithms achieve similar accuracies as non-distributed algorithms. The comparison across algorithms is not possible as for binary classifiers, logistic regression and linear SVM, we binarized all non-binary data sets by setting the label for instances with the most frequent class value to 0 (as indicated in Table 3, column Cl) and labels of all the other instances to 1. We also compare the clusterings produced by the k-means algorithm and the clusterings defined by the target labels. The number of clusters for k-means was set to the number of target labels in each data set. In Table 4 we show the values of the adjusted rand index. One can notice that the values for scikit-learn and DMLL are very similar, which indicates that similar clusterings are produced.

In Table 5 we present the results of summation form algorithms on big data sets, for which we assume that they do not fit into the memory. scikit-learn models are therefore trained on subsets with N/NC samples (see Table 3 for these values), while DiscoMLL models are trained on the entire data set distributed over worker nodes (each node contains N/NC samples). All models were tested on the entire test set. The performance of DiscoMLL models is equal or significantly better than the performance of scikit-learn models, except for Naive Bayes and Linear SVM on the mnist8m data set.

7.2. Performance of distributed ensembles

We evaluate the performance of developed ensemble methods (FDDT, DRF and DWF), described in Section 6, and compare their classification accuracy with bagging (*scikit BG*) and random forests (*scikit RF*) implemented in the scikit-learn toolkit. We compare distributed and non-distributed algorithms in two ways:

- giving each worker the whole data set (only possible for small data sets) we expect comparable predictive performance;
- training the distributed ensembles on subsets, we can expect decreased accuracy in comparison with non-distributed methods with all instances at their disposal. Altogether the distributed algorithms can process far more data than single machines so we expect improved performance in comparison to single machines with only chunks of data. For these methods we therefore analyze the decrease of accuracy due to distributed learning. We start with small data sets and observe if the findings generalize to big data sets.

We use a similar testing scenario as for summation form algorithms in Section 7.1, i.e. we use 5 × 2 cross-validation and each training set is randomly split into 18 chunks to match the number of processors in our distributed environment. To measure the decreased performance due to distributed implementations, we simulate the data distribution process using scikit-learn, as follows. We train a model on a subset of 1/18 training instances, predict the entire testing set and measure the classification accuracy. This process is repeated for each of 18 subsets. The same testing method is used with DiscoMLL ensemble algorithms (DiscoMLL subset). We expect similar performance: *scikit subset* $\simeq$ *DiscoMLL subset*. To measure the upper bound of classification accuracy for the algorithms, we train the classification models using all the instances with scikit (scikit ideal). In practice this is not always feasible due to the size of data sets, therefore we also measure the performance in the distributed scenario using the distributed ensembles, which produce models in parallel using subsets and then combine local models in the prediction phase. Due to distributed learning, we expect the following relation between the performance scores: *scikit subset* $\leq$ *DiscoMLL dist* $\leq$ *scikit ideal*.

We present average classification accuracy of distributed and non-distributed ensembles on small data sets in Table 6. To statistically quantify the differences between different methods we test the null hypothesis that distributed ensembles (training a model on the entire data set split into chunks) return models with the same prediction accuracy compared to single processor based implementations, which train models on subsets of data. We applied a paired t-test to measure the significance of differences between matching *scikit subset* and *DiscoMLL dist* scores. In cases when the null hypothesis is rejected we can assume that distributed ensembles are beneficial. The *scikit BG subset* score is compared with *FDDT dist* score and *scikit RF subset* score is compared with *DRF dist* and *DWF dist* scores. The + signs in *FDDT dist*, *DRF dist*, and *DWF dist* columns denote significant improvements of classification accuracy and the - signs denote significant decrease in accuracy. We observe that FDDT achieves significant improvement over scikit BG on every data set except yeast. Similarly DRF achieves a significant improvement over scikit RF on all data sets except on car and wilt data sets. The DWF algorithm achieves several significant improvements over scikit RF, but also two significant decreases of classification accuracy (on wilt and yeast data sets). In general DWF is mostly inferior to DRF. We believe that the reason for this is unreliable assessment of sample similarity based on k-medoid clustering, which is sensitive to noise in the form of less important features. This causes certain samples labeled differently to appear similar. Improvement in the efficient instance similarity assessment is a topic of future work.

Based on the above performance we can confirm our expectations and report a significant increase of accuracy for distributed methods, which use the entire data set split into chunks, compared to single processor methods using only subsets of data. On

[4] In contrast to that, the implemented distributed ensemble methods are not equivalent to the non-distributed implementations as we train them in a distributed fashion using subsets of training data. We compare them to scikit-learn ensemble methods in Section 7.2.

Table 4
Results of summation form algorithms in distributed and non-distributed fashion on small data sets. Classification accuracy is presented for classification algorithms and the adjusted rand index is given for *k*-means clustering. For logistic regression and Linear SVM the data sets are binarized. The + and − signs denote a significant increase or decrease of classification accuracy, respectively.

Data set	Naive Bayes		Logistic regression		Linear SVM		*k*-means	
	scikit	DMLL	scikit	DMLL	scikit	DMLL	scikit	DMLL
abalone	0.22	0.23	0.83	0.83−	0.83	0.83	0.04	0.06+
adult	0.81	0.83+	0.80	0.82+	0.80	0.81+	−0.01	−0.01
car	0.71	0.84+	0.86	0.86	0.87	0.86	0.02	0.01
isolet	0.81	0.81	1.00	0.99−	1.00	0.99−	0.46	0.44
segmentation	0.77	0.77	1.00	1.00+	1.00	1.00	0.37	0.36
semeion	0.82	0.84+	0.97	0.91−	0.97	0.95−	0.37	0.38
spambase	0.82	0.81	0.92	0.92	0.92	0.89−	0.04	0.04
wilt	0.88	0.88	0.95	0.97+	0.95	0.94−	−0.01	−0.01
wine-white	0.44	0.44	0.56	0.56+	0.56	0.56	0.01	0.01
yeast	0.23	0.23	0.70	0.69	0.70	0.70	0.03	0.03

Table 5
Results of summation form algorithms on big data sets. scikit-learn trains models on data subsets while DiscoMLL trains models on the entire data set, but in distributed fashion. Classification accuracy is presented for classification algorithms and adjusted rand index is given for clustering.

Data set	Naive Bayes		Logistic regression		Linear SVM		*k*-means	
	scikit	DMLL	scikit	DMLL	scikit	DMLL	scikit	DMLL
covertype	0.26	0.68	0.76	0.76	0.76	0.76	0.00	0.00
epsilon	0.60	0.67	0.80	0.90	0.80	0.90	0.00	0.00
mnist8m	0.49	0.46	0.98	0.98	0.98	0.95	0.27	0.29

Table 6
Classification accuracy of ensemble methods in distributed and uniprocessor mode on small data sets. The scikit BG subset performance is compared with FDDT dist and scikit RF subset performance is compared with DRF dist and DWF dist. The + and − signs denote a significant improvement and reduction of classification accuracy, respectively.

Data set	scikit BG		scikit RF		FDDT		DRF		DWF	
	subset	ideal	subset	ideal	subset	dist	subset	dist	subset	dist
abalone	0.22	0.23	0.22	0.24	0.22	0.26+	0.22	0.26+	0.22	0.26+
adult	0.84	0.86	0.84	0.85	0.84	0.86+	0.85	0.86+	0.84	0.86+
car	0.79	0.96	0.77	0.95	0.79	0.84+	0.78	0.80	0.77	0.79
isolet	0.75	0.90	0.76	0.94	0.72	0.88+	0.74	0.88+	0.72	0.85+
segmentation	0.87	0.97	0.87	0.97	0.88	0.92+	0.88	0.92+	0.80	0.89+
semeion	0.54	0.84	0.58	0.92	0.48	0.77+	0.53	0.82+	0.50	0.80+
spambase	0.89	0.95	0.91	0.95	0.90	0.92+	0.91	0.92+	0.91	0.93+
wilt	0.96	0.98	0.96	0.98	0.96	0.97+	0.96	0.95	0.96	0.95−
wine-white	0.49	0.64	0.50	0.65	0.51	0.53+	0.51	0.55+	0.49	0.54+
yeast	0.46	0.61	0.43	0.61	0.46	0.51	0.48	0.50+	0.44	0.40−

the other hand, the performance of distributed methods is well beyond the upper bound achieved by using the entire data set in non-distributed mode which leaves much opportunity for further research.

Table 7 presents the results of ensemble methods on big data sets, which are too big to fit into memory of a single machine (ideal scores cannot be computed). For these data sets, scikit-learn algorithms use subsets with N/NC samples while DiscoMLL distributed ensembles use the entire data sets split into chunks to train a model. To make parameters comparable, scikit-learn ensembles construct all the trees on a single subset, while DiscoMLL ensembles construct less trees per subset (as indicated in Table 3, column TC) and combine them into ensembles of equal size. We observe that distributed ensembles mostly achieve higher accuracy. This confirms the benefits of a distributed approach: for data sets too large to fit into memory, the distributed ensemble methods can use more data split into chunks and thereby outperform the non-distributed methods.

Fig. 10 shows learning time of DiscoMLL ensemble methods and their speedup with different number of CPUs on artificially increased [68] segmentation data set (for other data sets the behavior is similar). All methods achieve almost ideal linear speedup i.e. if the time using one node (2 CPUs) is t, the time using x nodes ($2x$ CPUs) is only slightly larger than t/x.

The actual times used by the methods are parameter dependent, but in general DWF is slower than DRF and FDDT as it uses clustering. DRF is faster than FDDT if it uses the same number of trees as it estimates only $\sqrt{a}$ features in each tree node, where a is the number of attributes. FDDT is faster than scikit RF using the same number of processors.

Table 7
Results of ensemble methods on big data sets, which do not fit into memory of single machines. Scikit algorithms use subsets of data, while DiscoMLL algorithms use distributed computation (dist) to learn on the entire data sets split into chunks.

Data set	scikit BG subset	scikit RF subset	FDDT dist	DRF dist	DWF dist
covertype	0.82	0.79	0.82	0.82	0.82
epsilon	0.71	0.70	0.73	0.74	0.73
mnist8m	0.89	0.92	0.90	0.92	0.92

8. Comparison and integration of ClowdFlows with related platforms

ClowdFlows is an open source data mining platform that can successfully process big data and handle potentially infinite streams of data.

The graphical user interface of ClowdFlows is implemented as a web application that is executed on a remote server. This distinguishes the platform from other platforms such as RapidMiner, KNIME, Weka, and Orange which require an installation which poses distinct software and hardware requirements. The side effect of

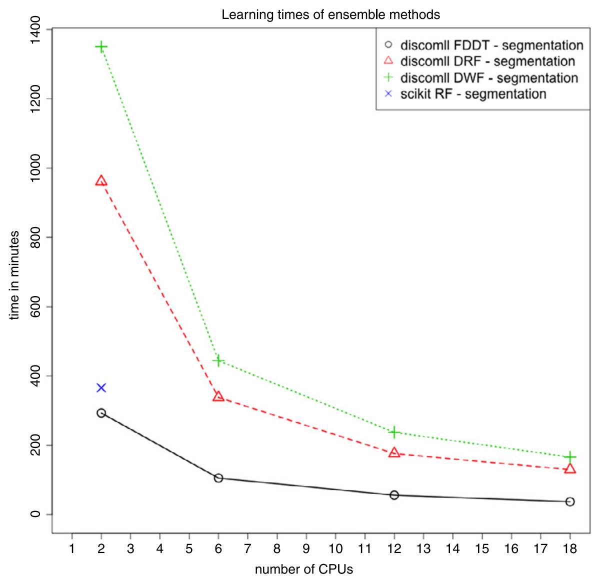


Fig. 10. Learning times of MapReduce ensemble methods with different number of CPUs.

this feature is that users can share their work with anybody as only a web browser is required to view, modify or execute a workflow in ClowdFlows.

Even though ClowdFlows can be deployed on a single machine it is designed with a modular architecture and scalability in mind. Different parts of the system are decoupled so that they may be duplicated for higher performance.

The non-local nature of ClowdFlows makes it an ideal platform for running long term workflows for processing data streams as users do not need to worry about leaving their machines turned on during the process of mining the stream.

ClowdFlows is to the best of our knowledge the only platform that provides a graphical user interface to the Disco Framework and publicly available implementations of data mining algorithms for big data mining in this framework.

The ClowdFlows platform can also be regarded as an open source alternative to the Microsoft Azure Machine Learning platform which requires a subscription with the Azure cloud. ClowdFlows is released under a permissive open source license and can be deployed on public or private clouds.

ClowdFlows in its current state cannot handle processing finished workflows of other platforms but can integrate procedures and algorithms implemented in other platforms and use them internally as part of its own workflows. ClowdFlows features algorithm implementations from Weka, Orange, and scikit-learn. In this way ClowdFlows can be used as a graphical user interface for other platforms, including platforms that do not have graphical user interfaces for constructing workflows, such as TensorFlow. While the TensorFlow interface is not yet implemented in ClowdFlows, it is a subject for further work. The platform is compatible with ClowdFlows as they are both interfaced and expressed in Python.

The ClowdFlows platform can be used within other data mining platforms either by calling ClowdFlows Python functions or by importing REST API services deployed by any live installation of ClowdFlows. The second method exposes an HTTP URL endpoint which executes the workflow when input data is posted to it.

The ClowdFlows platform has been evaluated in independent surveys where it was valued as one of the leading platforms with regards to the number of features [81].

9. Conclusions and further work

We presented the ClowdFlows, a data mining platform that supports the construction and execution of scientific workflows. The platform implements a visual programming paradigm, which allows users to present complex procedures as a sequence of simple steps. This makes the platform usable for non-experts. The ClowdFlows platform allows importing web services as workflow components. With this feature the processing abilities of the ClowdFlows platform are not limited to the initial roster of processing components, but can be expanded with web services. The interface for constructing and monitoring of workflow execution is implemented as a web application and can be accessed from any contemporary web browser. The data and the workflows are stored on the server or a cluster of servers (i.e., a cloud), so that the users are not limited to a single device to access their work. Similarly, workflows are not limited to a single user, but can be made public, which allows other users to use existing workflows either to reproduce the experiments, or as templates to expand and create new workflows.

We presented two modules for big data processing: a real-time analysis module and a batch processing module. Both modules are accessible via an intuitive graphical user interface that are easy to use for data mining practitioners, students, and non-experts.

The stream mining mode uses the stream mining daemon that executes workflows in parallel. The workflow components offer several novel features, so that workflows can connect to potentially infinite number of data streams and process their data. We demonstrated their use by extracting semantical triplets from a live RSS feed.

For analyzing big data in batch mode we developed DiscoMLL, a machine learning library for the Disco MapReduce framework and several ClowdFlows widgets that can interact with a Disco cluster and issue MapReduce tasks. We implemented several summation form algorithms and developed three new ensemble methods based on random forests. Performance analysis shows the benefit of using all available data for learning in the distributed mode compared to using only subsets of data in the non-distributed mode. We demonstrate the ability of our implementation to handle big data sets and its nearly perfect linear speedup.

Both the batch-mode and real-time processing modules were demonstrated with practical use cases that can be reproduced and executed either on the public installation of the ClowdFlows platform, or on a private cluster.

There are several directions for future work. First, the ClowdFlows platform currently implements its own stream processing engine which is built-in and easy to use. On the other hand, the Storm project is widely accepted as a de facto solution for massive stream processing and we plan to provide a loose integration of Storm into the ClowdFlows platform. Likewise, the Apache Hadoop and Spark will be integrated to complement the Disco framework.

Second, the existing ClowdFlows workflow engine will be extended to support different underlying processing platforms. The workflow engine should be able to delegate and monitor tasks transparently providing an easy-to-use programming interface.

Third, the ClowdFlows platform currently is not able to import or export workflows from other visual programming tools. For example, workflows constructed in Taverna or RapidMiner using web services and standard input/output components could easily be imported.

Fourth, we will simplify the installation procedures of ClowdFlows clusters by providing one-click deployment and automatization of scaling.

Finally, the available widget repository will be extended with high quality open-source data processing libraries to cover several new data analysis scenarios, e.g., high-throughput bioinformatics, large scale text processing, graph mining, etc.

To conclude, we believe that ClowdFlows has the potential to become a leading platform for data mining and sharing experiments and results due to its open source nature and its non-opinionated design regarding its collections of workflow components. It is our strategic vision for developers to create their own workflow components, expand the ClowdFlows workflow repository and deploy their own versions of ClowdFlows as a part of a large ClowdFlows network. Since ClowdFlows was released as open source software it has been forked many times and deployed on public servers with custom opinionated sets of workflow components (e.g. ClowdFlows Unistra and TextFlows [82]). With the addition of big data mining and stream mining capabilities presented in this paper we expect the number of users and ClowdFlows installations to increase.

Acknowledgments

This work was supported by the Slovenian Research Agency (Grant Agreement No. P2-0103) and the project ConCreTe, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET Grant No. 611733, and the Human Brain Project (HBP, Grant Agreement No. 604102). We wish to thank Borut Sluban for implementing VIPER in the ClowdFlows platform.

Appendix. Summation form algorithms in DiscoMLL

Besides distributed ensemble methods (FDDT, DRF, and DWF) presented and analyzed in Section 6, DiscoMLL contains implementations of several existing machine learning algorithms in summation form. Since to the best of our knowledge their pseudo code in MapReduce remains unpublished, but is of interest to the scientific community, we include it in this appendix. We present Naive Bayes, logistic regression, K-means clustering, linear regression, locally weighted linear regression, and support vector machine algorithms.

Naive Bayes for numeric attributes

Naive Bayes (NB) for discrete attributes was presented as an example in Section 5.5.1. Here we describe handling of numerical attributes. On numerical features the NB classifier (also called Gaussian Discriminative Analysis in this context) uses numerical features to learn the following statistics: mean, variance and prior probability $P(y)$. The map function (Algorithm 3) takes a training instance, breaks it into individual features and generates output key/value pairs. Each output pair contains the training label y and the feature index j as the key and the feature value x_j as the value. The occurrences of training labels are output in pairs, the training label as the key and 1 as the value. The combiner calculates local statistics (mean, variance and prior probability) for each map task to reduce network load. The reduce function (Algorithm 4) accepts partially calculated statistics for each attribute and combines them appropriately. The statistics are output and used to build a model, which is applied in the predict phase. The output of the predict phase was compared to the Naive Bayes algorithm implemented in the scikit-learn toolkit [44].

We combined the NB classifier for discrete and numeric features into a single algorithm. The computed conditional scores for discrete and numeric attributes are combined into a single score to predict the label $\hat{y}$ with the maximal score as stated in Eq. (A.1) below.

$$\hat{y} = \arg\max_y P(y)\left(\prod_{j\in Numeric} P(x_j|y)\right)\left(\prod_{j\in Discrete} P(x_j|y)\right). \quad (A.1)$$

Algorithm 3: The map function of the fit phase in the NB for numeric attributes.

```
function map(sample, params)
    x, y = sample
    for j = 0 to length(x)
        #key: label, attribute index
        #value: attribute value
        output((y, j), x_j)
    #mark label occurrence
    output(y, 1)
```

Algorithm 4: The reduce function of the fit phase in the NB for numeric attributes.

```
function reduce(iterator, params)
    y_dist = hashmap()
    for key, values in iterator
        if key has 1 element
            #count label occurrences
            y_dist.put(key, sum(values))
        else if key has 2 elements
            #combine local statistics
            mean = calculate_mean(values)
            var = calculate_variance(values)
            output(key, mean)
            output(key, var)
    prior = calculate_prior_probs(y_dist)
    output("prior", prior) #P(y)
```

Logistic regression

The logistic regression classifier is a binary classifier that uses numeric features. The classifier learns by fitting θ to the training data, using the hypothesis in the form $h_\theta(x) = g(\theta^T x) = 1/(1 + \exp(-\theta^T x))$. We use the Newton–Raphson method to update $\theta := \theta - H^{-1}\nabla_\theta \ell(\theta)$. For the summation form, we calculate the subgroups of gradients by map tasks, denoted as $\nabla_\theta \ell(\theta)$, by $\sum_{subgroup}(y - h_\theta(x))x_j$, and the Hessian matrix by $H(j,k) := H(j,k) + h_\theta(x)(h_\theta(x) - 1)\ x_j\ x_k$. The subgroups of the gradient and the Hessian matrix can be computed in parallel by map tasks as shown in Algorithm 5. The logistic regression updates θ in each iteration, where one iteration represents one MapReduce job. Before the execution of each MapReduce job, the θ are stored in object *params* and passed as argument. The map function calculates the hypothesis with x and θ from the previous iteration. The subgroups of gradient and Hessian matrix are calculated and output. The reduce function (Algorithm 6) takes an iterator over key/value pairs. Values with the same key are grouped together by the intermediate phase. The reduce function sums subgroups of gradients and Hessian matrix, updates θ and outputs it. This procedure takes place until convergence or a user-specified number of iterations. The output of the predict phase was compared with the logistic regression algorithm implemented in Orange [11].

Algorithm 5: The map function of the fit phase in the logistic regression.

```
function map(sample, params)
    x, y = sample
    h = calc_hypothesis(x, params.thetas)
    output("grad", calc_gradient(x, y, h))
    output("H", calc_hessian(x, h))
```

Algorithm 6: The reduce function of the fit phase in the logistic regression.

```
function reduce(iterator, params)
    for key, value in iterator
        if key == "H"
            H = sum(value)
        else
            grad = sum(value)
    thetas = params.thetas − inv(H) ∗ grad
    output("thetas", thetas)
```

K-means clustering

The *k*-means is a partitional clustering technique that aims to find a user-specified number of clusters (k) represented by their centroids. The computation of distances between the training instances and centroids can be parallelized. In the initial iteration, the map function randomly assigns data points to k clusters. The mean of data point values, assigned to a certain cluster, defines its centroid. The MapReduce procedure is repeated until it reaches a user-specified number of iterations. The map function (Algorithm 7) takes *sample* as the input parameter, which represents a data point. It computes the Euclidean distance between a data point and each centroid. It assigns each data point to the closest centroid and outputs the cluster identifier as key and the data point as value. The reduce function (Algorithm 8) recomputes centroids for each cluster and outputs the cluster identifier as key and the updated centroid as value. The k reduce tasks are parallelized across the cluster, where each task recalculates a certain centroid. The implementation of the k-means algorithm was taken from Disco examples and was adapted to work with DiscoMLL. The output of the predict phase was compared to the k-means implementation in the scikit-learn toolkit [44].

Algorithm 7: The map function of the fit phase in the k-means.

```
function map(sample, params)
    distances = calc_distances(sample,
                               params.centers)
    center_id = min(distances)
    output(center_id, sample)
```

Algorithm 8: The reduce function of the fit phase in the k-means.

```
function reduce(iterator, params)
    for center_id, samples in iterator
        update_center(params.centers[center_id],
                      samples)

    for center_id, samples in params.centers:
        output(center_id, average(samples))
```

Linear regression

The linear regression fits θ to training data with the equation $\theta^* = A^{-1}b$, where $A = \sum_{i=1}^{m}(x_i x_i^T)$ and $b = \sum_{i=1}^{m}(x_i y_i)$ with m training instances. To put these equations into summation form, the map function calculates $\sum_{subgroup}(x_i x_i^T)$ and $\sum_{subgroup}(x_i y_i)$ as shown in Algorithm 9. The reduce function (Algorithm 10) iterates over subgroups of A and b and sums them. Then it calculates the equation for θ^* and outputs the parameters.

Algorithm 9: The map function of the fit phase in the linear regression.

```
function map(sample, params)
    x, y = sample
    A = outer_product(x, x)
    b = inner_product(x, y)
    output("A", A)
    output("b", b)
```

Algorithm 10: The reduce function of the fit phase in the linear regression.

```
function reduce(iterator, params)
    for key, value in iterator
        if key == "A"
            A = sum(value)
        else:
            b = sum(value)
    thetas = inner_product(inv(A), b)
    output("thetas", thetas)
```

Locally weighted linear regression

Locally weighted linear regression (LOESS) stores training data and computes a linear regression at prediction time, separately for each testing instance. In the linear regression formula instances are weighted with their distances to the testing point, so that points closer to the given testing instance have a strong effect on prediction. This effect is modeled with parameter τ.

LOESS finds a solution of the equation $A\theta = b$, where

$$A = \sum_{i=1}^{m} w^{(i)}(x^{(i)}(x^{(i)})^T),$$

$$b = \sum_{i=1}^{m} w^{(i)}(x^{(i)}y^{(i)}),$$

where $w^{(i)}$ are distance based weights, $x^{(i)}$ is a training instance, and $y^{(i)}$ is function value of instance i. For summary form computation we use map function to compute subsets for A and b, while reduce sums subsets and finds solution to $\theta = A^{-1}b$.

LOESS makes one pass through training data for prediction of a single testing instance, which takes too much time for prediction of many instances. Our implementation computes *theta* parameters for several instances in one pass (Algorithm 11).

Algorithm 11: Computation of θ parameters with LOESS.

```
def get_thetas(train_set, test_set, tau = 1):
    result_url = [], test_set = {}
    read_test_set = read(test_set)
    for id, x in read_test_set:
        test_set[id] = x
        if below_max_capacity(test_set):
            params = Params(test_set, tau)
            # compute thetas for given subset
            thetas = compute(train_set,params)
            results_url.append(thetas)
            testing_set = {}
    return results_url
```

Function (Algorithm 12) uses dictionary of testing instances. Training instances are used to compute weights, and subset of matrices A and b for all testing instances. Function map returns as many pairs as there are testing instances in a dictionary.

Algorithm 12: The map function for LOESS

```
def map(instance, params):
    xi, y = instance
    for id, x in params.test_instances:
        w = weights(xi, x, params.tau)
        sub_A = w * outer_product(xi, xi)
        sub_b = w * xi * y
        yield (id, (sub_A, sub_b))
```

Function reduce (Algorithm 13) sums all subsets of matrices for test instances with the same id. It sorts pairs on the key and merges values based on the key (function *kvgroup*). For each test case we sum subsets of matrices A and b, and compute parameters θ.

Algorithm 13: Function reduce for LOESS.

```
def reduce(iterator, params):
    for id, value in kvgroup(iterator):
        A, b = 0, 0
        for sub_A, sub_b in value:
            A += sub_A
            b += sub_b
        thetas = vector_product(inverse(A),b)
        prediction = vector_product(thetas,
                            params.test_set[id])
        yield (id, (thetas, prediction))
```

Support Vector Machines

We implemented an incremental linear SVM, described in [83], which requires a single pass over the training set with time complexity $O(n^3)$ and space complexity $O(n^2)$. The method assumes numeric attributes, and a binary class encoded with -1 or $+1$. A training set with n instances and m attributes is stored in matrix A, and class values are stored in a diagonal matrix D. We compute matrix $E = [A - e]$, where e is a unit matrix of dimension $m \times 1$. For a user supplied parameter ν, we compute

$$\begin{bmatrix} w \\ \gamma \end{bmatrix} = \left(\frac{I}{\nu} + E^T E\right)^{-1} E^T De. \tag{A.2}$$

With the map function (Algorithm 14) we compute $E^T E$ and $E^T De$ in a distributed fashion.

Algorithm 14: Function map for linear SVM.

```
def map(sample, params):
    A, D  = sample
    e = unit_matrix(len(A), 1)
    E = column_merge(A, -e)
    ETE = inner_product($E^T$, $E$)
    ETDe = inner_product($E^T$, D, e)
    yield ("key", (ETE, ETDe))
```

In the reduce function (Algorithm 15) we sum *ETE* and *ETDe*, create the unit matrix I of size $(n+1) \times (n+1)$, which is divided by parameter ν. Using (A.2) we return parameters of linear SVM of size $(n+1) \times 1$.

Algorithm 15: Reduce function of linear SVM.

```
def reduce(iterator, params):
    sum_ETE, sum_ETDe = 0,0
    for key, value in iterator:
        if key == "ETE":
            sum_ETE += value
        else:
            sum_ETDe += value
    I = unit_matrix(sum_ETE.dimension_x)
    sum_ETE += I/params.nu
    yield ("key", vector_product(
                inverse(sum_ETE), sum_ETDe))
```

Each test instance x is extended with 1 in the prediction phase, $z = [{x \atop -1}]$and we get the prediction y as

$$y = sign\left(z^T \left(\frac{I}{\nu} + E^T E\right)^{-1} E^T De\right). \tag{A.3}$$

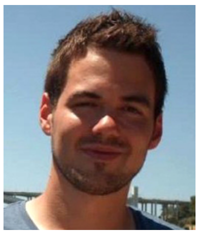
References

[1] J.W. Tukey, The future of data analysis, Ann. Math. Statist. 33 (1) (1962) 1–67.
[2] Q. Yang, X. Wu, 10 challenging problems in data mining research, Int. J. Inf. Technol. Decis. Mak. 05 (04) (2006) 597–604.
[3] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, T. Widener, The KDD process for extracting useful knowledge from volumes of data, Commun. ACM 39 (1996) 27–34.
[4] J. Chambers, Computing with data: Concepts and challenges, Amer. Statist. 53 (1999) 73–84.
[5] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Burlington, Massachusetts, ISBN: 1-55860-238-0, 1993.
[6] J. He, Advances in data mining: History and future, in: Third International Symposium on Intelligent Information Technology Application, IITA 2009, Vol. 1, 2009, pp. 634–636.
[7] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, third ed., Morgan Kaufmann, Amsterdam, ISBN: 978-0-12-374856-0, 2011.
[8] M.M. Burnett, Visual Programming, John Wiley & Sons, Inc., New York, 2001, pp. 275–283.
[9] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler, YALE: Rapid prototyping for complex data mining tasks, in: L. Ungar, M. Craven, D. Gunopulos, T. Eliassi-Rad (Eds.), KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2006, pp. 935–940.
[10] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, KNIME: The Konstanz information miner, in: C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (Eds.), GfKl, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, ISBN: 978-3-540-78239-1, 2007, pp. 319–326.
[11] J. Demšar, B. Zupan, G. Leban, T. Curk, Orange: From experimental machine learning to interactive data mining, in: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), PKDD, in: Lecture Notes in Computer Science, vol. 3202, Springer, ISBN: 3-540-23108-0, 2004, pp. 537–539.
[12] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows: A cloud based scientific workflow platform, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 7524, Springer, ISBN: 978-3-642-33485-6, 2012, pp. 816–819.
[13] Executable Paper Challenge, 2016. URL http://www.executablepapers.com/ (accessed: 18.01.16).
[14] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM (ISSN: 0001-0782) 51 (1) (2008) 107–113. http://dx.doi.org/10.1145/1327452.1327492, URL http://doi.acm.org/10.1145/1327452.1327492.
[15] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows sources, 2016. URL http://mloss.org/software/view/513/ (accessed: 20.01.2016).

[16] D. Talia, P. Trunfio, O. Verta, Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids, in: A. Jorge, L. Torgo, P. Brazdil, R. Camacho, J. Gama (Eds.), PKDD, in: Lecture Notes in Computer Science, vol. 3721, Springer, 2005, pp. 309–320. ISSN 3-540-29244-6.
[17] V. Podpečan, M. Zemenova, N. Lavrač, Orange4WS environment for service-oriented data mining, Comput. J. 55 (1) (2012) 89–98.
[18] D. Hull, K. Wolstencroft, R. Stevens, C.A. Goble, M.R. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Res. 34 (Web-Server-Issue) (2006) 729–732.
[19] G. Decker, H. Overdick, M. Weske, Oryx - An open modeling platform for the BPM community, in: M. Dumas, M. Reichert, M.-C. Shan (Eds.), Business Process Management, in: Lecture Notes in Computer Science, vol. 5240, Springer, Berlin/ Heidelberg, ISBN: 978-3-540-85757-0, 2008, pp. 382–385.
[20] D. Blankenberg, G.V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, J. Taylor, Galaxy: A web-based genome analysis tool for experimentalists, in: Frederick M. Ausubel, et al. (Eds.), Current Protocols in Molecular Biology, 2010, ISSN1934-3647 (Chapter 19).
[21] R. Rak, A. Rowley, W. Black, S. Ananiadou, Argo: an integrative, interactive, text mining-based workbench supporting curation, Database: J. Biol. Databases Curation (2012).
[22] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R.F. da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, Future Gener. Comput. Syst. (ISSN: 0167-739X) 46 (2015).
[23] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflow management in condor, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for E-Science, Springer, London, ISBN: 978-1-84628-519-6, 2007, pp. 357–375.
[24] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wieczorek, ASKALON: A development and grid computing environment for scientific workflows, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for E-Science, Springer, London, ISBN: 978-1-84628-519-6, 2007, pp. 450–471.
[25] E. Elmroth, F. Hernandez, J. Tordsson, Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment, Future Gener. Comput. Syst. (ISSN: 0167-739X) 26 (2) (2010) 245–256. URL http://www.sciencedirect.com/science/article/pii/S0167739X09001174.
[26] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, Kepler: an extensible system for design and execution of scientific workflows, in: Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004, 2004. pp. 423–424, http://dx.doi.org/10.1109/SSDM.2004.1311241.
[27] I. Taylor, M. Shields, I. Wang, A. Harrison, The Triana Workflow Environment: Architecture and applications, in: Workflows for E-Science, 1, 2007, pp. 320–339.
[28] T. White, Hadoop: the Definitive Guide, O'Reilly, 2012.
[29] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, R. Sears, MapReduce online, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 21–21. URL http://dl.acm.org/citation.cfm?id=1855711.1855732.
[30] C.T. Chu, S.K. Kim, Y.A. Lin, Y. Yu, G.R. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, in: B. Schölkopf, J.C. Platt, T. Hoffman (Eds.), NIPS, MIT Press, 2006, pp. 281–288. URL http://www.cs.stanford.edu/people/ang//papers/nips06-mapreducemulticore.pdf.
[31] Apache Mahout: Scalable machine learning and data mining, 2016. URL http://mahout.apache.org/ (accessed: 20.01.16).
[32] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Vol. 10, 10, 2010.
[33] Z. Prekopcsák, G. Makrai, T. Henk, C. Gáspár-Papanek, Radoop: Analyzing big data with rapidminer and hadoop, in: Proceedings of the 2nd RapidMiner Community Meeting and Conference, RCOMM 2011, 2011.
[34] P. Mundkur, V. Tuulos, J. Flatow, Disco: a computing platform for large-scale data analytics, in: Proceedings of the 10th ACM SIGPLAN Workshop on Erlang, ACM, 2011, pp. 84–89.
[35] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: Distributed stream computing platform, in: 2010 IEEE International Conference on Data Mining Workshops, (ICDMW), IEEE, 2010, pp. 170–177.
[36] N. Marz, Storm, distributed and fault-tolerant realtime computation, 2016. URL http://storm-project.net/ (accessed: 20.01.16).
[37] G.D.F. Morales, SAMOA: a platform for mining big data streams, in: L. Carr, A.H.F. Laender, B.F. Lóscio, I. King, M. Fontoura, D. Vrandecic, L. Aroyo, J.P.M. de Oliveira, F. Lima, E. Wilde (Eds.), WWW (Companion Volume), International World Wide Web Conferences Steering Committee /, ACM, ISBN: 978-1-4503-2038-2, 2013, pp. 777–778.
[38] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.
[39] P. Reutemann, J. Vanschoren, Scientific workflow management with ADAMS, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), ECML/PKDD (2), in: Lecture Notes in Computer Science, vol. 7524, Springer, ISBN: 978-3-642-33485-6, 2012, pp. 833–837.
[40] J. Vanschoren, H. Blockeel, A community-based platform for machine learning experimentation, in: W. Buntine, M. Grobelnik, D. Mladenič, J. Shawe-Taylor (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 5782, Springer Berlin, Heidelberg, ISBN: 978-3-642-04173-0, 2009, pp. 750–754.
[41] C. Kiourt, D. Kalles, A platform for large-scale game-playing multi-agent systems on a high performance computing infrastructure, Multiagent Grid Syst. 12 (1) (2016) 35–54.
[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow. org.
[43] Microsoft, Microsoft Azure Machine Learning, 2016. URL https://azure.microsoft.com/en-us/services/machine-learning/ (accessed: 04.04.16).
[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[45] JPype - Java to Python Integration, 2016. URL http://jpype.sourceforge.net/ (accessed 20.01.16).
[46] M. Lichman, UCI Machine Learning Repository, 2013. URL http://archive.ics.uci.edu/ml.
[47] B. Sluban, D. Gamberger, N. Lavrač, Ensemble-based noise detection: noise ranking and visual performance evaluation, Data Min. Knowl. Discov. (2013) 1–39.
[48] J. Kranjc, V. Podpecan, N. Lavrac, Real-time data analysis in ClowdFlows, in: 2013 IEEE International Conference on Big Data, IEEE, 2013, pp. 15–22.
[49] jQuery, 2016. URL http://jquery.com/ (accessed 20.01.16).
[50] Usage Statistics and Market Share of JavaScript Libraries for Websites, January 2014, 2016. URL http://w3techs.com/technologies/overview/javascript_library/all (accessed 20.01.16).
[51] Django: The Web framework for perfectionists with deadlines, 2016. URL https://www.djangoproject.com/ (accessed 20.01.16).
[52] A. Richardson, Introduction to RabbitMQ, Google UK, Sep 25.
[53] Python Simple SOAP library, 2016. URL https://code.google.com/p/pysimplesoap (accessed 20.01.16).
[54] E. Ikonomovska, S. Loskovska, D. Gjorgjevik, A survey of stream data mining, in: Proceedings of 8th National Conference with International Participation, ETAI, 2007, pp. 19–21.
[55] D. Rusu, B. Fortuna, M. Grobelnik, D. Mladenič, Semantic graphs derived from triplets with application in document summarization, Informatica (Slovenia) 33 (3) (2009) 357–362.
[56] PyTeaser source code, 2016. URL https://github.com/xiaoxu193/PyTeaser (accessed: 20.01.16).
[57] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, D. Mladenic, Triplet extraction from sentences, in: Proceedings of the 10th International Multiconference, Information Society-IS, 2007, pp. 8–12.
[58] Stanford Parser, 2016. URL http://nlp.stanford.edu/software/lex-parser.shtml (accessed: 20.01.16).
[59] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, O'Reilly Media Inc., 2009.
[60] G.A. Miller, WordNet: a lexical database for English, Commun. ACM 38 (11) (1995) 39–41.
[61] M. Bostock, V. Ogievetsky, J. Heer, D^3 data-driven documents, IEEE Trans. Vis. Comput. Graphics 17 (12) (2011) 2301–2309.
[62] T. Dwyer, Scalable, versatile and simple constrained graph layout, in: Computer Graphics Forum, Vol. 28, Wiley Online Library, 2009, pp. 991–998.
[63] R. Orač, Disco Machine Learning Library, 2015. URL https://github.com/romanorac/discomll (accessed: 10.01.15).
[64] T.E. Oliphant, Python for scientific computing, Comput. Sci. Eng. 9 (3) (2007) 10–20.
[65] C. Chu, S.K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, Adv. Neural Inf. Process. Syst. 19 (2007) 281.
[66] M. Kearns, Efficient noise-tolerant learning from statistical queries, J. ACM 45 (6) (1998) 983–1006.
[67] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, ACM, 2006, pp. 161–168.
[68] M. Robnik-Šikonja, Data generators for learning systems based on RBF networks, IEEE Trans. Neural Netw. Learn. Syst. 27 (5) (2016) 926–938.
[69] L. Breiman, Random forests, Mach. Learn. J. 45 (2001) 5–32.
[70] A. Verikas, A. Gelzinis, M. Bacauskiene, Mining data with random forests: A survey and results of new tests, Pattern Recognit. 44 (2) (2011) 330–349.
[71] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, X. Wu, MReC4.5: C4.5 ensemble classification with MapReduce, in: Fourth ChinaGrid Annual Conference, IEEE, 2009, pp. 249–255.
[72] J. Basilico, M. Munson, T. Kolda, K. Dixon, W. Kegelmeyer, COMET: A recipe for learning and using large ensembles on massive data, in: 2011 IEEE 11th International Conference on Data Mining, (ICDM), IEEE, 2011, pp. 41–50.
[73] B. Panda, J. Herbach, S. Basu, R. Bayardo, PLANET: Massively parallel learning of tree ensembles with MapReduce, Proc. VLDB Endow. 2 (2) (2009) 1426–1437.
[74] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, 1993.
[75] L. Breiman, Pasting small votes for classification in large databases and on-line, Mach. Learn. 36 (1–2) (1999) 85–103.
[76] I. Kononenko, On biases in estimating multi-valued attributes, in: Proceedings of the International Joint Conference on Artificial Intelligence, (IJCAI'95), Morgan Kaufmann, 1995, pp. 1034–1040.
[77] L. Breiman, Bagging predictors, Mach. Learn. J. 26 (2) (1996) 123–140.
[78] M. Robnik-Šikonja, Improving Random Forests, in: Machine Learning: Proceedings of ECML 2004, 2004.

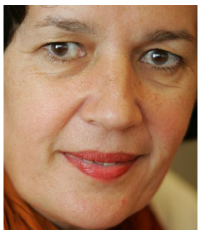
[79] J. Gower, A general coefficient of similarity and some of its properties, Biometrics (1971) 857–871.
[80] A. Frank, A. Asuncion, UCI Machine Learning Repository, 2010. URL http://archive.ics.uci.edu/ml.
[81] O. Kurasova, V. Marcinkevičius, V. Medvedev, A. Rapečka, Data mining systems based on web services, Informacijos mokslai 65 (2013) 66–74.
[82] M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, N. Lavrač, TextFlows: A visual programming platform for text mining and natural language processing, Spec. Issue Knowl.-Based Softw. Eng. Sci. Comput. Program. 121 (2016) 128–152.
[83] G. Fung, O.L. Mangasarian, Incremental support vector machine classification, in: SDM, SIAM, 2002, pp. 247–260.

Janez Kranjc started his graduate studies at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. He is enrolled in the Ph.D. programme "Information and Communication Technologies" and works as a research assistant at the Department of Knowledge Technologies, Jožef Stefan Institute. His research in the field of computer science mostly focuses on developing novel cloud-based approaches to data mining and knowledge discovery using graphical scientific workflows and developing applications and infrastructure to facilitate continuous mining of data streams and vast amounts of data dispersed on the cloud. The applications of the developed platform and workflows span several fields such as data mining, text mining, natural language processing, and sentiment analysis.



Roman Orač completed his B.Sc. in computer science in 2011 and his M.Sc. in 2014. He is a big data enthusiast who finds interest in the field of large-scale data mining. He is currently employed as an analytics engineer at Iddiction.

Vid Podpečan, Ph. D. is a researcher at the Department of Knowledge technologies at the Jožef Stefan Institute, Ljubljana, Slovenia. He received his Ph.D degree in computer science in 2013. His research interests lie mostly within robotics, artificial intelligence and data analysis, with a focus on humanoid robots and their applications, computational creativity, and data mining topics such as network analysis and bioinformatics. His research work has resulted in 45+ scientific publications and several software tools (e.g., Orange4WS, SegMine). Recently, he became involved in promoting robotics with Aldebaran's NAO humanoid robot to younger generations, especially children.



Nada Lavrač is Head of Department of Knowledge Technologies of Jozef Stefan Institute, Ljubljana, Slovenia. She is Professor at Jozef Stefan International Postgraduate School in Ljubljana and at University of Nova Gorica. She was initiator and co-founder of Centre for Knowledge Transfer in Information Technologies at JSI. She is Deputy Head of Information and Communication Technologies M.Sc. and Ph.D. Programs at Jozef Stefan International Postgraduate School in Ljubljana. Her main research interests are in Knowledge Technologies, an area of Information and Communication Technologies. Her particular research interests are machine learning, data mining, text mining, knowledge management and computational creativity.

Marko Robnik-Šikonja received his Ph.D. in computer science in 2001 from the University of Ljubljana. He is an Associate Professor at the University of Ljubljana, Faculty of Computer and Information Science. His research interests include machine learning, data and text mining, knowledge discovery, cognitive modeling, and their practical applications. He is a (co)author of more than 60 publications in scientific journals and international conferences and three open-source analytic tools.

Chapter 3

Adaptations of ClowdFlows

In this chapter we present several adaptations of the ClowdFlows platform. As the ClowdFlows platform is available as open-source software with a permissive license, it is possible for users to modify it and create their own adapted versions of ClowdFlows. First we describe the procedure for creating a new version of ClowdFlows, then we present two specific adaptations of the ClowdFlows platform—ConCreTeFlows [46] and TextFlows [47]. ConCreTeFlows is a platform for performing computational creativity tasks, while TextFlows is focused on text mining and natural language processing. Both platforms are presented with publications. The publication focused on ConCreTeFlows presents an illustrative use case of a novel computational creativity task. The chapter concludes with a publication which describes the development of TextFlows and outlines the differences between TextFlows and ClowdFlows.

3.1 Creating an Adaptation of the ClowdFlows Platform

ClowdFlows is released under the MIT license, which allows it to be modified and its modifications made public. The main drawback of ClowdFlows is a large number of workflow components which are often incompatible with each other. To remedy this, we propose collections of widgets to be joined in separate platforms. In this section we briefly describe the steps required to create an adaptation of the ClowdFlows platform and conclude with an example of an adaptation that was not created by the author of the thesis.

3.1.1 Setting up a development version

The ClowdFlows source code is available on GitHub[1]. GitHub is a web-based Git or version control repository. It offers all the distributed version control and source code management (SCM) functionalities of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests and task management.

Using Git, it is possible to download the source code to a development machine where a local installation can be set up. The steps needed to be taken to achieve this are as follows:

- As ClowdFlows is a Python project, it is customary (but not required) to create a virtual environment to keep all software requirements separate from other Python projects on the same machine. This eliminates the problem of required libraries version conflicts.

[1]https://github.com/xflows/clowdflows/

- The code is copied from GitHub using Git. This can be done using a command line or a graphical user interface in the target operating system.
- As all the software requirements are listed in a parsable text file, a single command line script is used to install the libraries needed to successfully execute and run ClowdFlows.
- The local version of ClowdFlows needs to be configured by editing a configuration file. A sample configuration file for local development is provided with the source code.
- The database is created and all workflow components are loaded from the file-based database into the relational data base that was set up in the configuration step.
- Finally, using the command line the ClowdFlows server can be started and accessed via web browsers on a local address.

All the above steps are listed with snippets of commands in the ClowdFlows documentation[2] available at `http://clowdflows.readthedocs.io/`.

3.1.2 Creating ClowdFlows widgets and packages

An adaptation of ClowdFlows would include different workflow components, named widgets. ClowdFlows widgets are contained in packages, which are collections of widgets that perform similar tasks.

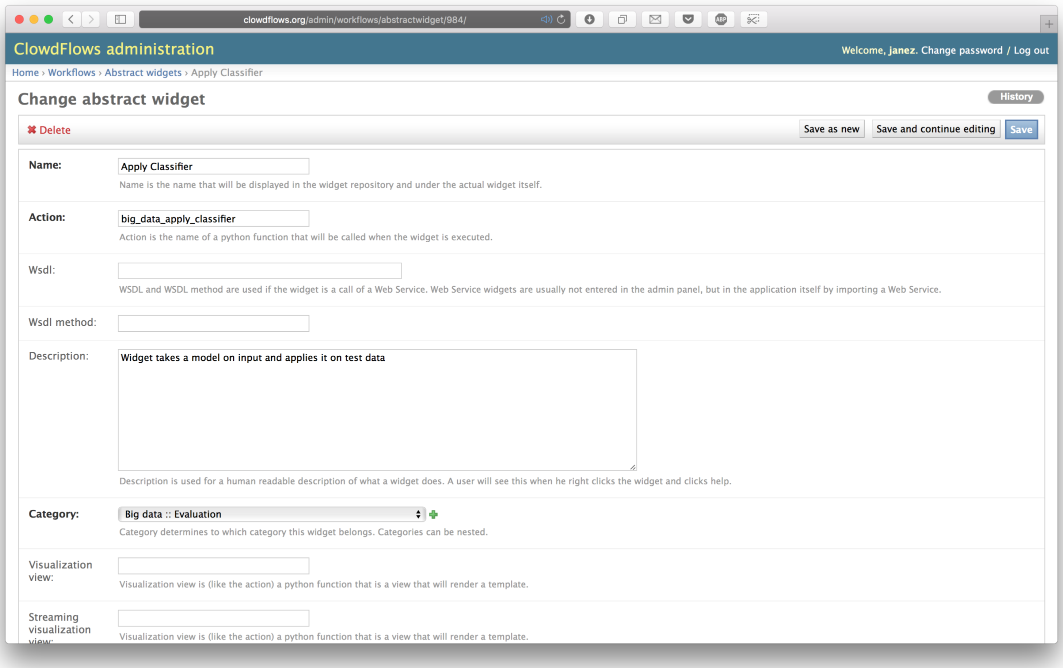
In the technical sense a ClowdFlows package is a collection of two types of files—widget source codes and package metadata. The source code determines what kind of task the widget performs. The metadata is automatically generated using a form in the ClowdFlows administration panel (accessible from the local development version). The form is shown in Figure 3.1.

ClowdFlows provides a command line tool that creates a package and includes a boilerplate code for creating widgets based on a template that is included with the source code. When a package is created, a unique name must be chosen which should not conflict with any existing Python package names to prevent unexpected behavior. The newly created package needs to be added to the local configuration (in the section where all activated packages are defined).

One of the created files is a special file entitled `library.py`. This file contains the source code of the widgets. Each widget is represented by a static method in this file. Each method receives a dictionary (a native Python data type) of inputs and should return a dictionary of outputs. The inputs, outputs, widget name, description and other meta data are entered in the ClowdFlows administration panel. When a widget is created in the administration panel it can immediately be used in the local ClowdFlows installation.

In order to transfer a widget from one ClowdFlows installation to another (e.g., from a development version to a live production version) the meta data files need to be created. They are created using the ClowdFlows package manager, which provides two command line tools: one for importing packages and another for exporting them. The export package command on the development server examines all widgets in the database (these were either imported or entered using the administration panel) and compares them to widgets in the data files. Missing widgets are added to the files (or updated if changed). On the target ClowdFlows installation the import command must be executed. Conversely, this

[2]http://clowdflows.readthedocs.io/en/latest/cf_dev_wiki/setting-up-your-development-version-of-clowdflows.html


clowdflows.org/admin/workflows/abstractwidget/984/
ClowdFlows administration
Welcome, janez. Change password / Log out
Home › Workflows › Abstract widgets › Apply Classifier
History
Delete
Save as new
Save and continue editing
Save

Change abstract widget

Name: Apply Classifier
Name is the name that will be displayed in the widget repository and under the actual widget itself.

Action: big_data_apply_classifier
Action is the name of a python function that will be called when the widget is executed.

Wsdl:
WSDL and WSDL method are used if the widget is a call of a Web Service. Web Service widgets are usually not entered in the admin panel, but in the application itself by importing a Web Service.

Wsdl method:

Description: Widget takes a model on input and applies it on test data
Description is used for a human readable description of what a widget does. A user will see this when he right clicks the widget and clicks help.

Category: Big data :: Evaluation
Category determines to which category this widget belongs. Categories can be nested.

Visualization view:
Visualization view is (like the action) a python function that is a view that will render a template.

Streaming visualization view:
Visualization view is (like the action) a python function that is a view that will render a template.

Figure 3.1: The ClowdFlows administration panel.

command compares data files with the database and adds (or removes if desired) widgets to the database.

It is possible to create external ClowdFlows packages which can be installed with Python's package manager and included into ClowdFlows. Using this method it is possible to have separate repositories for different packages. With such an architecture it is easy to maintain different package configurations. An example of such an external package is the RDM (Relational Data Mining) package [48]. This package is described in Chapter 4.

3.1.3 ClowdFlows Unistra

It is customary to rebrand an adapted platform so that it can be distinguished from the original one. An example adaptation is ClowdFlows Unistra[3], which is hosted at the University of Strasbourg, France. It features a separate authentication database and algorithms developed at the University of Strasbourg, such as the rule discovery tool Tertius [49], the first-order Bayesian classifiers 1BC and 1BC2 [50], and Propositionalisation using Relaggs, cardinalisation and quantiles [51].

The existence of this platform shows that the ClowdFlows documentation is sufficient and following it, it is possible to create a separate entity with personalized workflow components. The documentation includes instructions for all the steps and procedures described in this section along with several examples.

ClowdFlows Unistra and other adaptations of ClowdFlows can also benefit from ClowdFlows upgrades. As long as the maintainers of the ClowdFlows adaptations merge ClowdFlows changes with updates regularly, they will benefit from all the security fixes and new features.

[3]https://clowdflows.unistra.fr

3.2 ConCreTeFlows

This section presents the ConCreTeFlows platform—an adaptation of ClowdFlows which supports computational creativity. First, computational creativity systems are briefly introduced. This is followed by the inclusion of a conference proceedings publication which describes the platform and provides a novel computational creativity use case.

3.2.1 Background and motivation

Computational creativity is a subfield of artificial intelligence research, studying how to engineer software that exhibits behaviors which would reasonably be deemed creative [52]. These software systems are applied in domains associated with creative people, such as mathematics and science, poetry and story telling, musical composition and performance, video game, architectural, industrial and graphic design, and the visual arts.

Computational creativity systems use as their basic ingredients different types of resources, including musical, pictorial and textual. Such computational creativity systems include poetry generation [53], metaphor creation [54], generation of narratives, creation of fictional ideas [55] and conceptual blending [46]. These are computational creativity tasks which request manipulation of text resources that are provided as inputs.

Infrastructures supporting text-based creative systems are scarce. A text-based computational creativity system would automatically build creative artefacts from the given text resources, which end users would inspect and potentially adapt to their needs. An attempt in this direction is the FloWr system for automated flowchart construction, optimisation and alteration [56]. While getting software to write computational creativity code directly is a long-term research goal, that line of research is still in its infancy stage.

To this end, ClowdFlows was used to create ConCreTeFlows[4] as an easy-to-use workflow management system for computational creativity, allowing users to compose complex processing pipelines in a modular visual programming manner.

3.2.2 Related publication

The main feature that distinguishes ConCreTeFlows from ClowdFlows are the workflow components. The developed workflow components in ConCreTeFlows may be used to produce e.g., a conceptual blending workflow [46]. This workflow involves blending of texts from different domains, blending of corresponding images and poetry generation from texts.

The implementation details and the description of the workflow and all included components are presented in the following conference proceedings publication:

M. Žnidaršič, A. Cardoso, P. Gervás, P. Martins, R. Hervás, A. O. Alves, H. G. Oliveira, P. Xiao, S. Linkola, H. Toivonen, J. Kranjc, and N. Lavrač, "Computational creativity infrastructure for online software composition: A conceptual blending use case," in *Proceedings of the 7th International Conference on Computational Creativity*, ICCC, 2016, pp. 371–378.

In this publication we achieve the following:

- We present the ConCreTeFlows platform and illustrate its use in a specific use case of conceptual blending.
- We provide an introduction to conceptual blending theory.

[4]http://concreteflows.ijs.si

- We present an executable workflow that aims to conduct conceptual blending in three ways: conceptually, textually and visually. Given two descriptions of arbitrary concepts in natural language, the presented approach provides conceptual graph representations of both concepts and their blend, a textual description of the blended concept and even a set of possible visual blends.

The authors contributions are as follows. Martin Žnidaršič was the main developer and implementator of conceptual creativity workflows which are available in ConCreTeFlows and contributed to the development of available widgets. Amílcar Cardoso, Pedro Martins, and Ana Oliveira Alves contributed to the conceptual blending theory and the corresponding widgets. Hugo Gonçalo Oliveira authored the PoeTryMe workflow component. Pablo Gervas and Raquel Hervas were leading the work on the components for creation of text from graph representations. Ping Xiao, Simo Linkola and Hannu Toivonen served as the authors of the components for visual blending. Janez Kranjc collaborated with other authors on developing computational creativity widgets and provided necessary support for integrating third party libraries and software into ConCreTeFlows. Nada Lavrač contributed the idea of developing an adapted ClowdFlows platform for supporting computational creativity workflows and supervised the work.

Computational Creativity Infrastructure for Online Software Composition: A Conceptual Blending Use Case

Martin Žnidaršič[1], Amílcar Cardoso[2], Pablo Gervás[4], Pedro Martins[2], Raquel Hervás[4], Ana Oliveira Alves[2], Hugo Gonçalo Oliveira[2], Ping Xiao[3], Simo Linkola[3], Hannu Toivonen[3], Janez Kranjc[1], Nada Lavrač[1]

[1]Jožef Stefan Institute, Ljubljana, Slovenia
[2]CISUC, DEI, University of Coimbra, Coimbra, Portugal
[3]Department of Computer Science and HIIT, University of Helsinki, Finland
[4]Universidad Complutense de Madrid, Spain

Abstract

Computational Creativity is a subfield of Artificial Intelligence research, studying how to engineer software that exhibits behaviors which would reasonably be deemed creative. This paper shows how composition of software solutions in this field can effectively be supported through a Computational Creativity (CC) infrastructure that supports user-friendly development of CC software components and workflows, their sharing, execution and reuse. The infrastructure allows CC researchers to build workflows that can be executed online and be reused by others with a single click on the workflow web address. Moreover, it allows building of procedures composed of software developed by different researchers from different laboratories, leading to novel ways of software composition for computational purposes that were not expected in advance. This capability is illustrated on a workflow that involves blending of texts from different domains, blending of corresponding images, poetry generation from texts as well as construction of narratives. The paper concludes by presenting plans for future work.

Introduction

Computational creativity (CC) systems use as their basic ingredients different types of resources, including musical, pictorial and textual, to name a few. This paper focuses on infrastructure support to CC systems that base their creativity on textual resources. Such CC systems include poetry generation, metaphor creation, generation of narratives, creation of fictional ideas and conceptual blending, which all represent CC tasks which request manipulation of text resources that are provided as inputs.

Infrastructures supporting text-based creative systems are scarce. Ideally, a text-based CC system would automatically build creative artefacts from the given text resources, which the end user would then inspect and potentially adapt to their needs. An attempt in this direction is the FloWr system for automated flowchart construction, optimisation and alteration (Charnley, Colton, and Llano 2014). While getting software to write CC code directly is a long-term research goal, that line of research is—with the exception of FloWr—still in its infancy stage. A substantially more mature area of research concerns the development of infrastructures supporting modular development, sharing and execution of code used in text mining tasks. Text mining has numerous open source algorithms and natural language processing (NLP) software libraries available (such as NLTK (Bird 2006) and scikit-learn (Pedregosa et al. 2011)). However, even text mining and NLP experiments are still difficult to reproduce, including the difficulty of systematic comparison of algorithms. To this end, a number of attempts have been made to develop easy-to-use workflow management systems, allowing users to compose complex processing pipelines in a modular visual programming manner.

Related work As regards the work related to the platform presented in this paper, we first mention myGrid[1] which is used primarily for bioinformatics research, having in mind experiment replication. It is currently probably the most advanced workflow management system, although, due to its complexity, not very easy to use. The most important part of myGrid is Taverna, which is conceived as a suite of tools used to design and execute scientific workflows. A multilingual Internet service platform Language Grid[2], which is based on a service-oriented architecture and supports a web-oriented version of the pipeline architecture typically employed by NLP tools, is open source, but it is quite complex to install and use. The ARGO platform[3] is a more recent development, which enables workflows to have interactive components, where the execution of the workflow pauses to receive input from the user, but ARGO is not open source and does not have sophisticated utilities for cataloguing the available web services or workflows, nor a system of access permissions.

Our recently developed platform ClowdFlows[4] (Kranjc, Podpečan, and Lavrač 2012) is web-based thus requiring no local installation, is simple to use and install, and available as open source under the MIT Licence. While ClowdFlows is mainly devoted to data mining, its fork TextFlows[5] is focused on text mining and NLP workflows. A fork platform for facilitation and reuse of computational creativ-

[1]`http://www.mygrid.org.uk/`
[2]`http://langrid.org/`
[3]`http://argo.nactem.ac.uk/`
[4]`http://clowdflows.org`
[5]`http://textflows.org`

ity software is called ConCreTeFlows[6]. It is an independent platform with a specific backend that is being continuously adapted to computational creativity tasks and tools. As forks of ClowdFlows, TextFlows and ConCreTeFlows benefit from its service-oriented architecture, which allows users to utilize web-services as workflow components. The distinguishing feature of these platforms is the ease of sharing and publicizing workflows, together with an ever growing roster of reusable workflow components and entire workflows. As completed workflows, data, and results can be made public by the author, the platform can serve as an easy-to-access integration platform for data mining, text mining or computational creativity processes. Each public workflow is assigned a unique URL that can be accessed by anyone to either replicate the experiment, or use the workflow as a template to design new similar workflows.

Contributions In this paper we present ConCreTeFlows and illustrate its use in a specific use case of conceptual blending (introduction to blending theory is provided on page 3). This example employs multiple software components that are being developed by various members of the computational creativity community. The presented composition of software aims to conduct conceptual blending conceptually, textually and visually. Given two descriptions of arbitrary concepts in natural language, the presented approach provides conceptual graph representations of both concepts and their blend, a textual description of the blended concept and even a set of possible visual blends.

The paper is structured as follows. The first section presents ConCreTeFlows as a special purpose workflow management platform aimed at supporting computational creativity tasks. In the next section is the core of this paper. It provides a description of the use case and the basics of its theoretical foundations, followed by presentation of all the important methods and software components that are applied for its purpose. Last part of this section is devoted to critical discussion and ongoing work on the presented components. The paper concludes with a brief summary and plans for further work.

Software Infrastructure

This section briefly describes the main components of the ConCreTeFlows. It is a special purpose workflow management platform, aimed at supporting (primarily text-based) computational creativity tasks.

Like ClowdFlows, ConCreTeFlows can also be used in a browser, while the processing is performed in a cloud of computing nodes. The backend of ConCreTeFlows uses Django[7], which is an open source web framework. The graphical user interface is implemented in HTML and JavaScript, using jQuery[8] and jQuery-UI[9] libraries. ConCreTeFlows is easily extensible by adding new packages and workflow components. Workflow components of several types allow graphical user interaction during run-time, and visualization of results by implementing views in any format that can be rendered in a web browser. Below we explain the concept of workflows in more detail and describe the basic concepts of ConCreTeFlows.

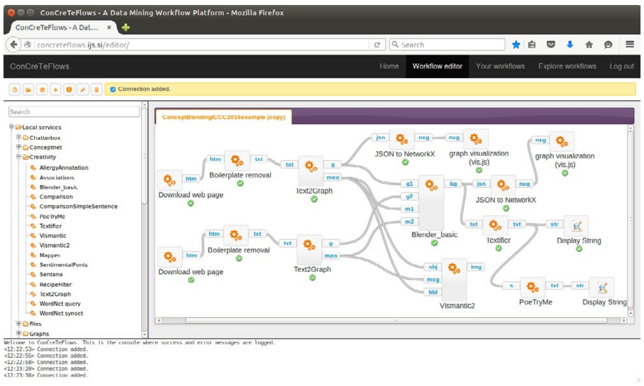
The workflow model is the main component of the ConCreTeFlows platform and consists of an abstract representation of workflows and workflow components. The graphical user interface for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The basic unit component in a ConCreTeFlows workflow is a processing component, which is graphically represented as a widget. Considering its inputs and parameters every such component performs a task and stores the results on its outputs. Different processing components are linked via connections through which data is transferred from a widget's output to another's input. An alternative widget input for a widget are parameters, which the user enters into widget's text fields. The graphical user interface implements an easy-to-use way of arranging widgets on a canvas to form a graphical representation of a complex procedure. Construction of new workflows thus requires no expertise, apart from knowing (usually from widget documentation) the inputs and outputs of the widgets to ensure their compatibility. Incorporation of new software components, on the other hand, requires basic programming skills in Python or SOAP web-service development in any programming language.

ConCreTeFlows implements its own workflow execution engine. Currently there are no ways to reuse the workflows using third party software. We plan to implement special widgets that will define inputs and outputs for REST API endpoints which will allow execution of workflows on variable inputs by any third party software.

The ConCreTeFlows graphical user interface is shown in Figure 1. On the top of the graphical user interface is a toolbar where workflows can be saved, deleted, and executed. Underneath on the left is the widget repository, which is a list of available widgets grouped by their functionality. Click on a widget in the repository adds it to the workflow construction canvas on the right. A console for displaying success and error messages is located on the bottom.

Workflows in ConCreTeFlows are processed and stored on remote servers from where they can be accessed from anywhere, requiring only an internet connection. By default each workflow can only be accessed by its author, although one may also chose to make it publicly available. ConCreTeFlows generates a specific URL for each workflow that has been saved as public. The users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by another user, a copy of the workflow is created on the fly and added to his private workflow repository. The workflow is copied together with widgets' parameter settings, as well as all the data, in order to ensure the experiments can be repeated. In this way the user is able to tailor the workflow to his needs without modifying the original workflow.

[6]`http://concreteflows.ijs.si`
[7]`https://www.djangoproject.com`
[8]`http://jquery.com`
[9]`http://jqueryui.com`



```
Welcome to ConCreTeFlows. This is the console where success and error messages are logged.
<12:22:53> Connection added.
<12:22:55> Connection added.
<12:22:58> Connection added.
<12:23:20> Connection added.
<12:23:30> Connection added.
```

Figure 1: A screenshot of the ConCreTeFlows graphical user interface opened in the Mozilla Firefox Web browser, presenting a motivational CC use case.

Conceptual Blending Online

The elements of the conceptual blending (CB) theory (Fauconnier and Turner 2002) are an inspiration to many algorithms and methodologies in the field of computational creativity (Veale and O'Donoghue 2000; Pereira 2005; Thagard and Stewart 2010; Schorlemmer et al. 2014). A key element in the theory is the *mental space*, a partial and temporary structure of knowledge built for the purpose of local understanding and action (Fauconnier 1994). To describe the CB process, the theory makes use of a network of four mental spaces (Figure 2). Two of these correspond to the *input spaces*, i.e., the content that will be blended. The process starts by finding a partial *mapping* between elements of these two spaces that are perceived as similar or analogous in some respect. A third mental space, called *generic*, encapsulates the conceptual structure shared by the input spaces, generalising and possibly enriching them. This space provides guidance to the next step of the process, where elements from each of the input spaces are *selectively projected* into a new mental space, called the *blend space*. Further stages of the process elaborate and complete the blend.

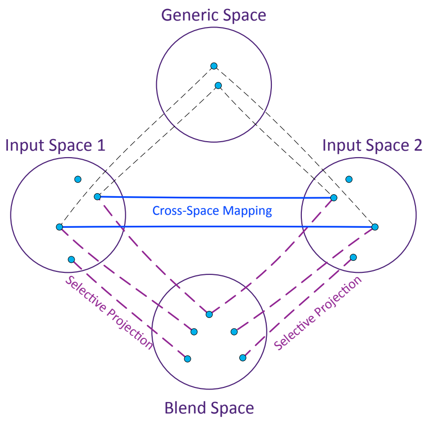


Figure 2: The original four-space conceptual blending network (Fauconnier and Turner 2002).

In most computational approaches to CB, the input and blended spaces are represented as computational versions of *Conceptual Maps* (Novak 1998), i.e., graphs where nodes are concepts and arcs are relations between them (see Figure 3).

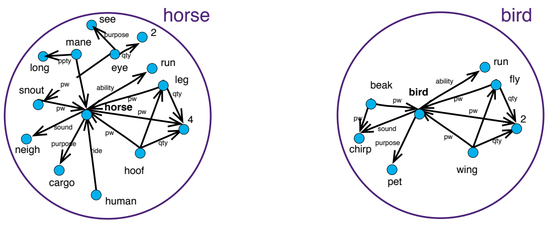


Figure 3: Concept maps of horse and bird.

Graph representations of concept blends are useful for automated analysis and further processing, but are not very suitable and appealing for human perception of the blended spaces. To improve on this aspect of conceptual blending, we have developed methodologies and algorithms for visual blending and for textual representation of concept graphs. Using these new techniques, we designed a CB process that results in conceptual blends that are described in natural language and enriched with visual representations. The process is sketched in Figure 4, where the boxes represent the main (software) components and the arrows indicate the flow of data from inputs to outputs.

Each of the main process components (that is, each box from the sketch in Figure 4) is implemented as an independent software solution and represented as a widget or a group of widgets in ConCreTeFlows.

In the following, we describe the process components and their implementations in detail, with a presentation of the whole workflow and some exemplary results at the end.

Construction of Conceptual Networks

The *Concept Network Builder* component from Figure 4 accepts a textual description of a concept in natural language and on its basis produces a conceptual graph.The set of possible concepts and relations in the resulting graph is open and not limited to a particular fixed set (such as relations in ConceptNet[10]) or linguistic characteristic. We decided to represent also relations as concepts, which allows treating a particular entity as both a concept or relation, depending on the context. For example, the concept of *eating* can be used to relate the concepts of *cows* and *grass*, but it can also be a concept related through *is a* with *animal activity*.

The software component that creates these conceptual graphs from text is implemented in ConCreTeFlows as the Text2Graph widget. This component first uses the Ollie triplet extractor (Mausam et al. 2012) to extract the triplets from the given text. The only text transformation before triplet extraction is uncapitalization of sentences. The resulting triplets are used to create a graph. In this process, the entities in the triplets can be lemmatized (this choice is

[10]http://conceptnet5.media.mit.edu/

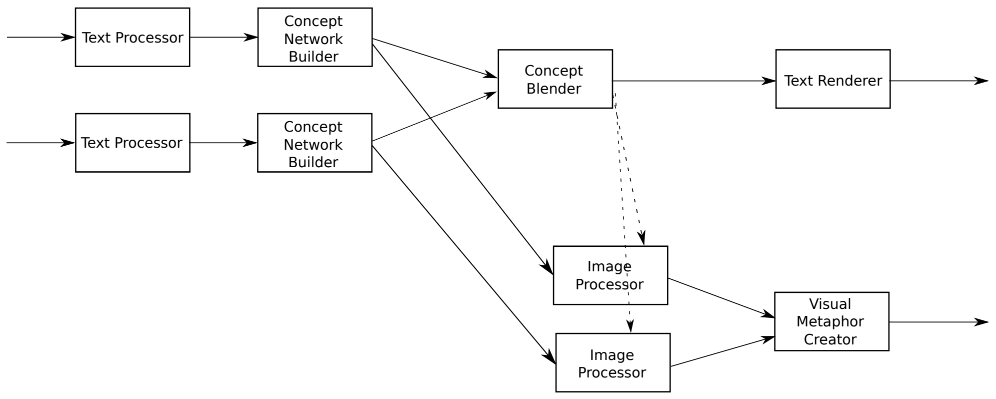


Figure 4: Sketch of the workflow for conceptual blending with visual and textual representations of blends.

left to the user). If the *Main size* parameter of the resulting graph is set, the graph is filtered to contain only a limited neighborhood around the *Main entity*, that is, the node in the graph that a user might be most interested in. The *Main entity* can be set by the user, but if it is not, it is selected automatically as the graph node with the highest out-degree.

Conceptual Blending

The *Concept Blender* component takes care of blending two elements that are represented as graphs. In our design of the process (Figure 4), the mapping between the elements from the input spaces is to be done by a human as an initial step (to select the two inputs to take part in blending) or to be done in the *Concept Blender* on all possible pairs of elements from the two input spaces. Following our representation, these would be pairs of conceptual graphs.

In the current baseline implementation, the *Concept Blender* expects only one pair of input elements, which it fully blends (merges the two conceptual graphs) without any influence of the *Generic Space*. An elaborate concept blending component, that is based on Divago framework, is in development as described in subsection on further work.

Visual Blending

In order to generate visual blends, the visual module, consisting of the *Image Processor* and the *Visual Metaphor Creator* in Figure 4, takes inputs from either the Concept Network Builder or the Concept Blender. From the first, it can take two concepts from two concept graphs (in this version, their main entities) as inputs to be visually blended. The resulting visual blend is not a representation of a blend created by Concept Blender, but an independent visual blend of the two concepts. For this purpose the visual module first finds photos tagged with the two concepts in Flickr, respectively. For ensuring the relevance and quality of the photos, we use a set of image analysis methods bundled together in QualiPy[11]. The image processor separates the subject and the background of each photo, and inpaints the background to hide any marks of the subject. The visual metaphor creator implements three visual operations: juxtaposition, replacement and fusion, as described by Xiao and Linkola (2015). In effect, it puts one object in the context of another, or gives an object the texture of another object (see Figure 5 for an example).

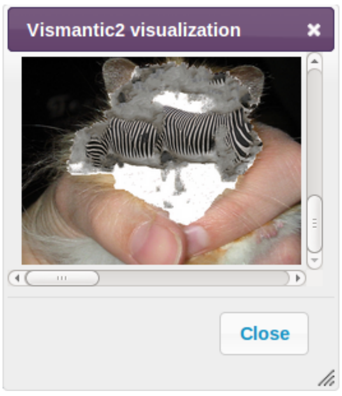
The visual module can also take input from the concept blender, which indicates a specific way of blending. Specifically, the input may indicate a choice between the replacement and fusion operations. For instance, a frequent conceptual blend is placing an object in an unusual environment, which suggests that the replacement operator shall be used.

In ConCreTeFlows, such blending is available in two versions (generations) as Vismantic and Vismantic2 widgets.

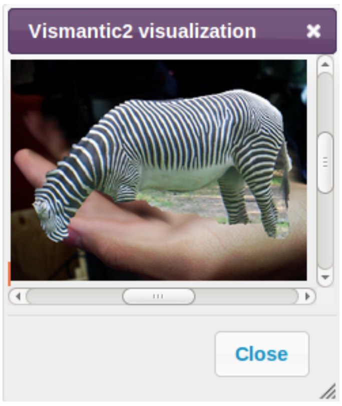
Text Generation

In order to generate a textual description of the blends obtained, a *Text Renderer* widget called Textifier has been added to the workflow. Textifier is a natural language generation tool that transforms data represented in a graph into a natural language text. It carries out stages of content determination, document planning and surface realization (Reiter and Dale 2000) and then translates the result into plain text. Content determination processes input to select and adapt what might be rendered. The input graph must contain pairs of source and target nodes with information, and the system will create all possible paths and represent them in a tree. Textifier first groups related information that refers to the same concept by combining nodes that contain the same subject and discarding duplicated nodes. Nodes that represent information with granularity inappropriate for textual rendering – such as verb-preposition groups represented as single strings – are rewritten to make all information explicit in the knowledge structure. Lastly, Textifier can prune the tree if only branches of a certain length need to be considered. Currently we are working with branches that are three nodes long, after detecting that they tend to contain more promising information. Document planning is basic at present but will play a larger role once the graphs of blends are processed. The surface realization stage transforms the tree into text. Figure 6 shows an example of Textifier in operation over a graph constructed from a given input text.

[11] `https://github.com/vismantic-ohtuprojekti/qualipy`

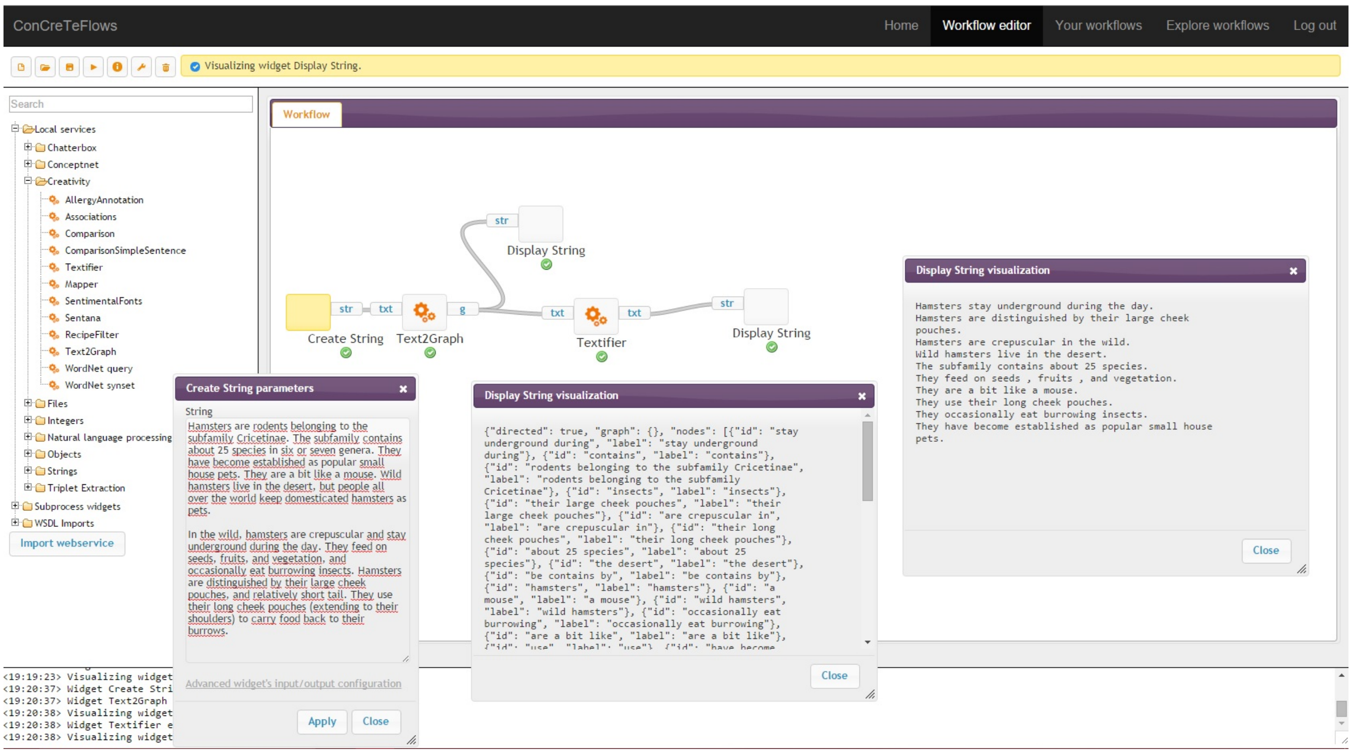


(a)



(b)

Figure 5: Two (out of four combinations of exchanging context and texture) outputs of the Vismantic2 widget for the example of blending the concepts of *hamster* and *zebra*. Figure 5a shows result of exchanging texture: hamster with a zebra's texture. In Figure 5b is an example of exchanging context: zebra is put in the usual visual context of a hamster.



Hamsters are rodents belonging to the subfamily Cricetinae. The subfamily contains about 25 species in six or seven genera. They have become established as popular small house pets. They are a bit like a mouse. Wild hamsters live in the desert, but people all over the world keep domesticated hamsters as pets.

In the wild, hamsters are crepuscular and stay underground during the day. They feed on seeds, fruits, and vegetation, and occasionally eat burrowing insects. Hamsters are distinguished by their large cheek pouches, and relatively short tail. They use their long cheek pouches (extending to their shoulders) to carry food back to their burrows.

```
{"directed": true, "graph": {}, "nodes": [{"id": "stay
underground during", "label": "stay underground
during"}, {"id": "contains", "label": "contains"},
{"id": "rodents belonging to the subfamily Cricetinae",
"label": "rodents belonging to the subfamily
Cricetinae"}, {"id": "insects", "label": "insects"},
{"id": "their large cheek pouches", "label": "their
large cheek pouches"}, {"id": "are crepuscular in",
"label": "are crepuscular in"}, {"id": "their long
cheek pouches", "label": "their long cheek pouches"},
{"id": "about 25 species", "label": "about 25
species"}, {"id": "the desert", "label": "the desert"},
{"id": "be contains by", "label": "be contains by"},
{"id": "hamsters", "label": "hamsters"}, {"id": "a
mouse", "label": "a mouse"}, {"id": "wild hamsters",
"label": "wild hamsters"}, {"id": "occasionally eat
burrowing", "label": "occasionally eat burrowing"},
{"id": "are a bit like", "label": "are a bit like"},
{"id": "use", "label": "use"}, {"id": "have become
```

```
Hamsters stay underground during the day.
Hamsters are distinguished by their large cheek
pouches.
Hamsters are crepuscular in the wild.
Wild hamsters live in the desert.
The subfamily contains about 25 species.
They feed on seeds , fruits , and vegetation.
They are a bit like a mouse.
They use their long cheek pouches.
They occasionally eat burrowing insects.
They have become established as popular small house
pets.
```

```
<19:19:23> Visualizing widget
<19:20:37> Widget Create Stri
<19:20:37> Widget Text2Graph
<19:20:38> Visualizing widget
<19:20:38> Widget Textifier e
<19:20:38> Visualizing widget
```

Figure 6: Example of Textifier working on input graph obtained from text.

Integration in a Workflow

The software components that implement the functionalities sketched in Figure 4 were implemented and integrated in ConCreTeFlows either as internal (Python) functions, wrapped standalone programs or as Web services. In addition, we implemented some additional components that support the user interaction and data processing. These are: (I) components for Web page content retrieval and filtering and (II) components for graph reformatting and visualization.

By connecting these software components, we composed a ConCreTeFlows workflow that conducts a basic conceptual, textual and visual concept blending. The workflow is presented in Figure 7 and is publicly available from: `http://concreteflows.ijs.si/workflow/137/` where it can be executed, changed and appended with additional functionality.

In this workflow, two textual inputs are transformed into conceptual graphs by a series of the Download web page, Boilerplate removal and Text2Graph widgets. The first one

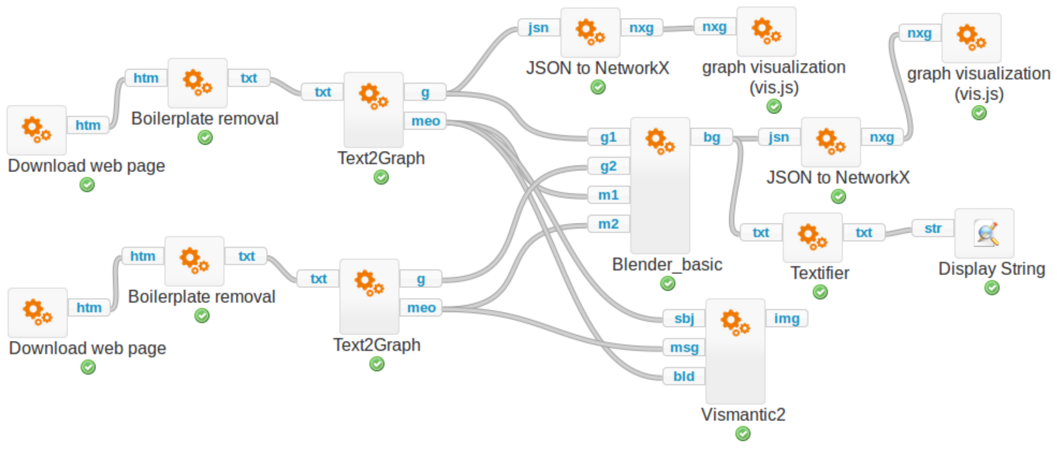


Figure 7: Workflow implementation in ConCreTeFlows (available at: `http://concreteflows.ijs.si/workflow/137/`).

obtains the Web page source from a given URL. In the example presented in this paper, these are the Wikipedia pages for two animals: hamster and zebra. The second widget removes the headers, menus, navigation and similar non-relevant content from the source. Finally, Text2Graph transforms the textual content into conceptual graphs (output *g*), which are available to other widgets with separately provided main entity (output *meo*). In the workflow, one of the graphs is reformatted and visualized with the graph visualization widget. All outputs of Text2Graph widgets enter the Blender_basic which blends the two graphs together and outputs a combined blended graph (output *bg*). This one gets served to the Textifier widget, which produces a textual description of the blend. Its output is presented by a standard Display String widget. The two main entities from Text2Graph widgets enter also the Vismantic2, which either changes the texture of one to the texture of the other (see Fig. 5a), or puts one in the usual surroundings of the other (Fig. 5b). This way it creates four candidates for visual blends. This widget takes somewhat longer to run, as it is in fact a call to a computationally intensive Web service. Upon completion, the outcome is shown in an output similar to the ones shown in Figure 5.

Workflow dissemination, reuse and extension

Any ConCreTeFlows workflow can either remain private or be made public for the purposes of dissemination and reproducibility of work. The workflow from the previous subsection is available from a public URL. This means that anyone can open it in ConcreTeFlows. Everytime this happens, a dedicated copy of the original workflow is made for that particular user. This allows any user not only to run the workflow with its original data and parameters, but also to change the inputs, parameters and redesign the structure of software components without affecting the original workflow.

Changing and extending a workflow is easy, but it requires some insight on the format of data that is exchanged among the widgets. This is usually made available in widget documentation, but can also be seen by observing the raw results of a widget (right-click and *Results*).

In the following we describe a simple exemplary extension of our workflow from Figure 7.

Exemplary addition: PoeTryMe widget The system named PoeTryMe (Gonçalo Oliveira and Cardoso 2015) is a poetry generation platform with a modular architecture that may be used to produce poetry in a given form, based on a set of seed words. Semantically-coherent lines are generated using the seeds or related words, and are produced by exploiting the knowledge in a semantic network and a grammar with textual renderings of the covered relations. A generation strategy selects some of the produced lines and organises them to suit the given form.

The PoeTryMe widget is limited to some of the features of the full system. Nevertheless, it can produce poetry in three languages (Portuguese, Spanish and English), given one of the available target forms (block of four, sonnet, ...), an open set of seeds, and a surprise factor, between 0 and 1, with implications on the selection of more or less semantically-distant words.

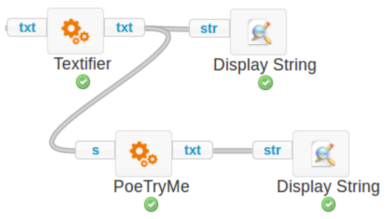


Figure 8: Addition of the PoeTryMe widget to the workflow.

In our workflow, the PoeTryMe widget can be appended to the Textifier widget (as shown in Figure 8) in order to get also a poem inspired by the resulting blend. Here is an example of a poem from a blend of *hamster* and *zebra*:

when the coat paints the water white and black
stadiums here make song each stand has his
have not yet grown by the familiar crack
will mine and leave where the great love is

Discussion and Future Work on Components

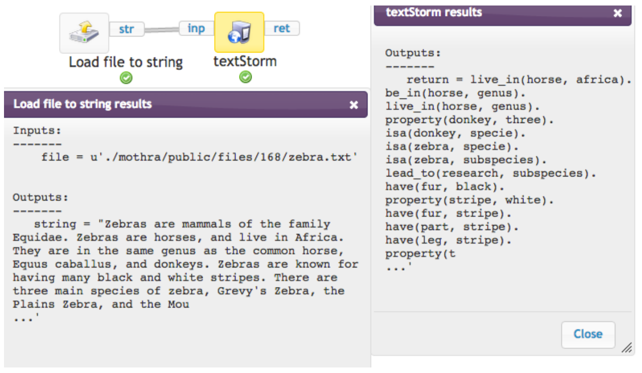
In the following, we discuss some of the encountered issues and shortcomings of the components and processes that are presented in this paper, as well as present some ongoing work on their improvements and additions.

Graph representations and formats The use of graphs for representing knowledge presents advantages in as much as it is a simple format with significant expressive power. In this sense it acts as a useful communication format for the various components in the flows envisaged. However, it has certain disadvantages in the sense that the graphs as considered at present do not have a unique semantic interpretation. Some of the modules produce graphs where relations are represented as edges between nodes representing objects, and others rely on graphs that represent relations as nodes occurring in the path of the graph between nodes representing objects. Even when the same approach to knowledge representation in a graph is used, problems may arise depending on the type of string used to label the nodes. Examples of problematic cases are: inflected verb forms used as well as verbs in infinitive, nouns used in singular and/or plural form, complex actions of the form `stay_at_home`... At present the content determination stage of the Textifier module is carrying out complex transformations to handle these various inputs in a uniform fashion when it comes to the final rendering. This requires the development of different version of the content determination stage for receiving input from different modules. It would be beneficial to make progress towards a unified approach to graph representation to allow blending operations to be carried out fruitfully between outputs generated by different modules. However, a certain flexibility is desirable in these content determination modules, so that they can tolerate inputs not altogether conforming to expectations. This is largely due to the open nature of the ConCreTeFlows platform, which may see the addition of new modules that do not conform to any standards set on graph representation, but also because the results of conceptual blending operations may not always produce output conforming to standards, even when the inputs to the conceptual blending process do conform.

TextStorm Conceptual Maps TextStorm (Oliveira, Pereira, and Cardoso 2001) is an NLP tool based on a Definite Clause Grammar (DCG) to extract binary predicates[12] from a text file using syntactic and discourse knowledge, not needing any preview knowledge about the discussed domain. The resulting set of predicates constitute a Conceptual Map. This tool can be used as an alternative to the Text2Graph.

TextStorm receives text as initial base of the open information extraction. After applying Part-of-Speech tagging and querying WordNet (Miller 1995), it builds predicates that map relations between two concepts from parsing of sentences. Its goal is to extract from utterances like Cows,

[12]These predicates have the common Prolog form: functor(Argument 1, Argument 2).



```
Inputs:
-------
    file = u'./mothra/public/files/168/zebra.txt'


Outputs:
-------
   string = "Zebras are mammals of the family
Equidae. Zebras are horses, and live in Africa.
They are in the same genus as the common horse,
Equus caballus, and donkeys. Zebras are known for
having many black and white stripes. There are
three main species of zebra, Grevy's Zebra, the
Plains Zebra, and the Mou
...'
```

```
Outputs:
-------
   return = live_in(horse, africa).
be_in(horse, genus).
live_in(horse, genus).
property(donkey, three).
isa(donkey, specie).
isa(zebra, specie).
isa(zebra, subspecies).
lead_to(research, subspecies).
have(fur, black).
property(stripe, white).
have(fur, stripe).
have(part, stripe).
have(leg, stripe).
property(t
...'
```

Figure 9: Imported Textstorm SOAP Web service used in ConCreTeFlows.

as well as rabbits, eat only vegetables, while humans eat also meat, the predicates eat(cow, vegetables), eat(rabbit, vegetables), eat(human, vegetables), eat(human, meat) which will form its concept map. Since concepts in text are not named every time the same way, TextStorm uses WordNet's synonymy semantic relationship to identify the concepts that were already referred before with a different name.

Textstorm operates as a standards compliant SOAP Web service and as such can be imported on-the-fly to ConCreTeFlows (see Figure 9).

Divago concept blender The concept blending method that is currently used in the workflow from Figure 7 is very basic.We are currently working on the adaptation of a more elaborated blender, the pre-existing Divago (Pereira 2005), to offer its main functionalities as webservices in ConCreTeFlows. This blender adopts the same graph format as TextStorm, i.e., the Conceptual Map format, for the input and blended mental spaces.

The new blender, the *DivagoFlow*, is itself a flowchart composed of two modules, the *Mapper* and the *Blend Factory* The first is responsible for finding analogy mappings between two Input Spaces using structural alignment. More precisely, it computes the largest isomorphic pair of sub-graphs contained in the Input Spaces. The output mapping is, for each pair of sub-graphs, the list of crossover relations between nodes of each of the input spaces. The *Blend Factory* takes these mappings as input, as well as the Input Spaces and a Generic Space. For each mapping, it performs a selective projection into the Blend Space, which leads to the construction of a *Blendoid*, an intermediate graph that subsumes the set of all possible blends. This Blendoid feeds an evolutionary process that explores the space of all possible combinations of projections of the Input Spaces taking into account the Generic Space. This module uses an implementation of the CB theory optimality principles (a set of principles that ensure a coherent and integrated blend) as fitness measure. When an adequate solution is found or a pre-defined number of iterations is attained, the Blend Factory stops the execution and returns the best Blend.

Conclusions

We have presented the ConCreTeFlows platform for online composition of computational creativity solutions. It is entirely Web based and does not require installation for its use. New processes in the platform can be designed as workflows of software components, which are either made available in the platform or even imported on-the-fly in case of SOAP Web services[13]. Workflows can be either private or shared, which makes for an elegant solution to dissemination and reuse of one's work and repeatability of experiments.

The main focus of the paper is on a use case, which shows how the platform can be used in practice and presents several computational creativity software components that were combined in a collaborative effort to implement an interesting conceptual blending solution. Namely, the resulting blends are not only conceptual but also visual and textual. The benefits of a unifying workflow for blending are twofold: a user can get blends of various kinds through the same user-interface and the components can affect one another to produce a more coherent and orchestrated set of multimodal blending results. While some of the presented components are currently being updated from implementing basic to more elaborate methods, the presented prototype solution is fully operational and serves as a proof of concept that such an approach to multimodal conceptual blending is possible. Potential for use of such an approach is for example in creation of news stories. Such a tool could form an entire automated article on a funny and humorous or a serious and thought-provoking blend of topics. All the components of an article are there: the text, the picture, as shown, one could even add a poem. Other potential uses of the approach could be in art, advertising and human creativity support.

To make these things possible, as described in section *Future Work on Components*, our future work will include improvement of the components and the workflow presented in this paper. We will also continue with development and improvement of the presented platform to make creation of this and other computational creativity solutions further more efficient, collaborative and fun.

Acknowledgments

This research was partly funded by the Slovene Research Agency and supported through EC funding for the project ConCreTe (grant number 611733) that acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission.

References

Bird, S. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, 69–72. Association for Computational Linguistics.

Charnley, J.; Colton, S.; and Llano, M. T. 2014. The FloWr Framework: Automated Flowchart Construction, Optimisation and Alteration for Creative Systems. In *Fifth International Conference on Computational Creativity (ICCC-2014)*, 315–323.

Fauconnier, G., and Turner, M. 2002. *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books.

Fauconnier, G. 1994. *Mental Spaces: Aspects of Meaning Construction in Natural Language*. New York: Cambridge University Press.

Gonçalo Oliveira, H., and Cardoso, A. 2015. Poetry generation with PoeTryMe. In Besold, T. R.; Schorlemmer, M.; and Smaill, A., eds., *Computational Creativity Research: Towards Creative Machines*, Atlantis Thinking Machines. Atlantis-Springer. chapter 12, 243–266.

Kranjc, J.; Podpečan, V.; and Lavrač, N. 2012. Clowdflows: A cloud based scientific workflow platform. In Flach, P. A.; Bie, T. D.; and Cristianini, N., eds., *ECML/PKDD (2)*, volume 7524 of *Lecture Notes in Computer Science*, 816–819. Springer.

Mausam; Schmitz, M.; Bart, R.; Soderland, S.; and Etzioni, O. 2012. Open language learning for information extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*.

Miller, G. A. 1995. Wordnet: A lexical database for english. *Commun. ACM* 38(11):39–41.

Novak, J. 1998. *Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Corporations*. Mahwah, NJ: Erlbaum.

Oliveira, A.; Pereira, F. C.; and Cardoso, A. 2001. Automatic reading and learning from text. In *Proceedings of the International Symposium on Artificial Intelligence*, 69–72.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research* 12:2825–2830.

Pereira, F. C. 2005. *Creativity and AI: A Conceptual Blending approach*. Ph.D. Dissertation, Dept. Engenharia Informática da FCTUC, Universidade de Coimbra, Portugal.

Reiter, E., and Dale, R. 2000. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

Schorlemmer, M.; Smaill, A.; Kühnberger, K.-U.; Kutz, O.; Colton, S.; Cambouropoulos, E.; and Pease, A. 2014. COINVENT: Towards a computational concept invention theory. In *Proceedings of the 5th Int. Conference on Computational Creativity, ICCC-14, Ljubljana, Slovenia*, 288–296.

Thagard, P., and Stewart, T. C. 2010. The AHA! experience: Creativity through emergent binding in neural networks. *Cognitive Science* 35(1):1–33.

Veale, T., and O'Donoghue, D. 2000. Computation and blending. *Cognitive Linguistics* Special Issue on Conceptual Blending:253–282.

Xiao, P., and Linkola, S. 2015. Vismantic: Meaning-making with images. In *Proceedings of the Sixth International Conference on Computational Creativity*, ICCC2015, 158–165.

[13]REST services can currently only be wrapped into native widgets.

3.3 TextFlows

This section presents the TextFlows platform which is an adaptation of the ClowdFlows platform and focuses on mining textual data and natural language processing (NLP). The section starts with a brief overview of workflow management software aimed at text mining and NLP. Some drawbacks of ClowdFlows are presented which form a motivation for creating TextFlows. The section and chapter are concluded with the inclusion of a journal publication which describes the platform along with several use cases. The main results and scientific contributions of the paper are highlighted before its inclusion.

3.3.1 Background and motivation

Text mining is a research area that deals with the construction of models and patterns from text resources, aiming at text categorization and clustering, taxonomy construction, sentiment analysis, etc. [57]. This research area, also known as text data mining or text analytics, is usually considered a subfield of data mining (DM) research [58], but can be viewed also more generally as a multidisciplinary field drawing its techniques from data mining, machine learning, natural language processing (NLP), information retrieval (IR), information extraction (IE) and knowledge management.

Workflow Management Systems have in the last years become a very hot topic, mostly in the field of bioinformatics and other natural sciences. Lately, however, this trend has also spread to text mining and NLP. In the area of Workflow Management Systems for text mining and NLP, established systems are slowly starting to be used for text analytics, while at the same time, new ones are being developed, specifically targeted to NLP. Among the workflow managament software platforms used for text mining and NLP are Taverna [19], PANACEA [59], ARGO [18], WebLicht [60], Language Grid [61] and LAPPS Grid [62]. We examined these systems with regard to being open-source, simplicity of use, workflow sharing and I/O protocols, and determined there exists a lack of an open source platform with a web-based graphical user interface that allows sharing of text mining and natural language processing workflows.

ClowdFlows as a general data mining platform incorporates workflow components for numerous data mining and machine learning projects while severely lacking text mining and natural language processing components. Moreover, despite being very useful for workflow construction by the informed developers and end-users, ClowdFlows currently suffers from a somewhat disorganized roster of workflow components which may be incompatible with one another due to their sheer numbers and diverse origins. As a result, new users sometimes struggle to construct functioning workflows as there are too many mutually incompatible components.

The motivation to create an adaptation of ClowdFlows for text mining and NLP is therefore two-fold. First, there is an opportunity to develop a widely used tool for workflow management systems, which is simplified by the fact that ClowdFlows already provides a graphical user interface and workflow engine along with all its other features. Secondly, limiting the roster of workflow components to a specific scientific field would increase the cohesion of incorporated workflow components. The new platform, named TextFlows, requires a redesigned roster of workflow components, which are highly compatible with each other within the platform and easier to use for the expert and novice users alike. In contrast to ClowdFlows, where there are no guidelines for sorting workflow components into categories, TextFlows sorts the components based on their functionality. As a result, all widgets that perform similar functions are located in the same category. The user interface was enriched with extra information about the workflow components and documentation was added for the provided workflow components and their inputs and outputs.

3.3.2 Related publication

The implementation details and description of the platform are included in the following journal publication:

M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, and N. Lavrač, "TextFlows: A visual programming platform for text mining and natural language processing," *Science of Computer Programming*, vol. 121, pp. 128–152, 2016.

In this publication we achieve the following:

- We provide a survey and detailed TextFlows comparison to five other natural language processing platforms.
- We present the TextFlows workflow management web platform for text mining and natural language processing and explain its relationship to ClowdFlows.
- We present the development of a unified data type object that is accepted and used by all workflow components in the platform—the annotated document corpus (ADC).
- We enable simple evaluation of algorithms from NLTK [63], LATINO [64] and scikit-learn [65] libraries.
- We demonstrate the features of the platform on three use cases: comparison of document classifiers and part-of-speech taggers on a text categorization problem, and outlier detection in document corpora.
- We show that LATINO's Max Entropy classifier achieves best results in document categorization.
- We show that Part-Of-Speech tagging improves the accuracy of document classification.

The authors' contributions are as follows. The integration of text mining widgets was implemented by Matic Perovšek. Janez Kranjc altered the ClowdFlows platform to support the execution of all types of widgets on different types of computation servers. Tomaž Erjavec investigated the related work and provided a comparison of TextFlows to other text mining platforms. Bojan Cestnik and Nada Lavrač supervised the work and provided ideas for the use cases. All authors contributed to the publication text.

Science of Computer Programming 121 (2016) 128–152

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

TextFlows: A visual programming platform for text mining and natural language processing

Matic Perovšek[a,b,*], Janez Kranjc[a,b], Tomaž Erjavec[a,b], Bojan Cestnik[a,c], Nada Lavrač[a,b,d]

[a] Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[c] Temida d.o.o., Ljubljana, Slovenia
[d] University of Nova Gorica, Nova Gorica, Slovenia

A R T I C L E I N F O

Article history:
Received 14 March 2015
Received in revised form 28 December 2015
Accepted 5 January 2016
Available online 14 January 2016

Keywords:
Text mining
Natural language processing
Web platform
Workflows
Human-computer interaction

A B S T R A C T

Text mining and natural language processing are fast growing areas of research, with numerous applications in business, science and creative industries. This paper presents TextFlows, a web-based text mining and natural language processing platform supporting workflow construction, sharing and execution. The platform enables visual construction of text mining workflows through a web browser, and the execution of the constructed workflows on a processing cloud. This makes TextFlows an adaptable infrastructure for the construction and sharing of text processing workflows, which can be reused in various applications. The paper presents the implemented text mining and language processing modules, and describes some precomposed workflows. Their features are demonstrated on three use cases: comparison of document classifiers and of different part-of-speech taggers on a text categorization problem, and outlier detection in document corpora.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Text mining [9] is a research area that deals with the construction of models and patterns from text resources, aiming at solving tasks such as text categorization and clustering, taxonomy construction, and sentiment analysis. This research area, also known as text data mining or text analytics, is usually considered as a subfield of data mining (DM) research [12], but can be viewed also more generally as a multidisciplinary field drawing its techniques from data mining, machine learning, natural language processing (NLP), information retrieval (IR), information extraction (IE) and knowledge management.

From a procedural point of view, text mining processes typically follow the CRISP-DM reference process model for data mining [7], which proposes six phases when working on a DM project: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. Text mining can be distinguished from general data mining by special procedures applied in the data preparation phase, where unstructured or poorly structured text needs to be converted into organized data, structured as a table of instances (rows) described by attributes (columns). In the modeling phase, such a table of instances can be used by the standard or slightly adapted data mining algorithms to uncover interesting

* Corresponding author.
E-mail addresses: matic.perovsek@ijs.si (M. Perovšek), janez.kranjc@ijs.si (J. Kranjc), tomaz.erjavec@ijs.si (T. Erjavec), bojan.cestnik@temida.si (B. Cestnik), nada.lavrac@ijs.si (N. Lavrač).

http://dx.doi.org/10.1016/j.scico.2016.01.001
0167-6423/© 2016 Elsevier B.V. All rights reserved.

information hidden in the data. Two typical approaches are using clustering algorithms to find groups of similar instances or classification rule learning algorithms to categorize new instances.

The TextFlows platform described in this paper is a new open source, web-based text mining platform that supports the design and composition of scientific procedures implemented as executable workflows. As a fork of ClowdFlows [28], TextFlows has inherited its service-oriented architecture that allows the user to utilize arbitrary web services as workflow components. TextFlows is oriented towards text analytics and offers a number of algorithms for text mining and natural language processing. The platform is implemented as a cloud-based web application and attempts to overcome various deficiencies of similar text analytics platforms, providing novel features that should be beneficial to the text mining community. In contrast to existing text analytics workflow management systems, the developed platform is the only one with all the following properties. It is simple (i.e., enables visual programming, is web-based and requires no installation), enables workflow sharing and reuse, and is open source. Moreover, the platform enables combining workflow components (called "widgets") from different contexts (e.g., using clustering in relational data mining) and from different software sources (e.g., building ensemble classifiers from different libraries). To do so, it provides a unified input-output representation, which enables interoperability between widget libraries through automated data type conversion. It uses a common text representation structure and advocates the usage of 'hubs' for algorithm execution.

TextFlows is publicly available at http://textflows.org, while its source code is available at https://github.com/xflows/textflows under the MIT License. Detailed installation instructions are provided with the source code. After setting up a local TextFlows instance, advanced users can also implement and test their own algorithms. Improvements to the code can also be pushed to the main Git code repository via pull requests. The committed changes are reviewed by the TextFlows core team and merged into the master branch.

TextFlows is a web application which can be accessed and controlled from anywhere while the processing is performed in a cloud of computing nodes. TextFlows differs from most comparable text mining platforms in that it resides on a server (or cluster of machines) while its graphical user interface for workflow construction is served as a web application through a web browser. The distinguishing feature is the ease of sharing and publicizing the constructed workflows, together with an ever growing roster of reusable workflow components and entire workflows. As not only widgets and workflows, but also data and results can be made public by the author, TextFlows can serve as an easy-to-access integration platform both for various text mining workflows but also for experiment replicability. Each public workflow is assigned a unique URL that can be accessed by anyone to either repeat the experiment, or to use the workflow as a template to design another, similar, workflow.

Workflow components (widgets) in TextFlows are organized into packages which allows for easier distributed development. The TextFlows packages implement several text mining algorithms from LATINO[1] [11], NLTK[2] [3] and scikit-learn[3] [32] libraries. Moreover, TextFlows is easily extensible by adding new packages and workflow components. Workflow components of several types allow graphical user interaction during run-time and visualization of results by implementing views in JavaScript, HTML or any other format that can be rendered in a web browser (e.g., Flash, Java Applet).

The rest of the paper is structured as follows. Section 2 presents the technical background and implementation details of the TextFlows platform, along with its key text mining components. The architecture of the system is presented in detail along with specific data structures that allow efficient text mining in a workflow environment. The concept of workflows, their implementation, execution and sharing are presented in Section 3, while Section 4 describes the widget repository and the implemented modules of the platform. The advanced features of TextFlows are demonstrated in Section 5 on three use cases: a comparison of document classifiers on a classification problem, a comparison of part-of-speech taggers on a text categorization (classification) problem, and outlier detection in document corpora. Section 6 presents the related work, where we describe comparable text mining platforms together with their similarities and differences compared to the TextFlows platform. Section 7 concludes the paper by presenting a summary and some directions for further work.

2. The TextFlows platform

This section presents the TextFlows platform, together with its architecture and main components of the system. We also introduce the graphical user interface and describe the concept of workflows. Like its predecessor data mining platform ClowdFlows [28], TextFlows can also be accessed and controlled from a browser, while the processing is performed on a cloud of computing nodes. In this section we explain the relationship between TextFlows and ClowdFlows, present the architecture of the TextFlows platform and describe the key text mining concepts of TextFlows in more detail.

2.1. Platform architecture

In software engineering, terms *front-end* and *back-end* are used to distinguish the separation between a presentation layer (the client side) and a data access layer (the server side), respectively. Fig. 1 shows the TextFlows architecture, which

[1] LATINO (Link Analysis and Text Mining Toolbox) is open-source—mostly under the LGPL license—and is available at http://source.ijs.si/mgrcar/latino.
[2] http://www.nltk.org.
[3] http://scikit-learn.org.

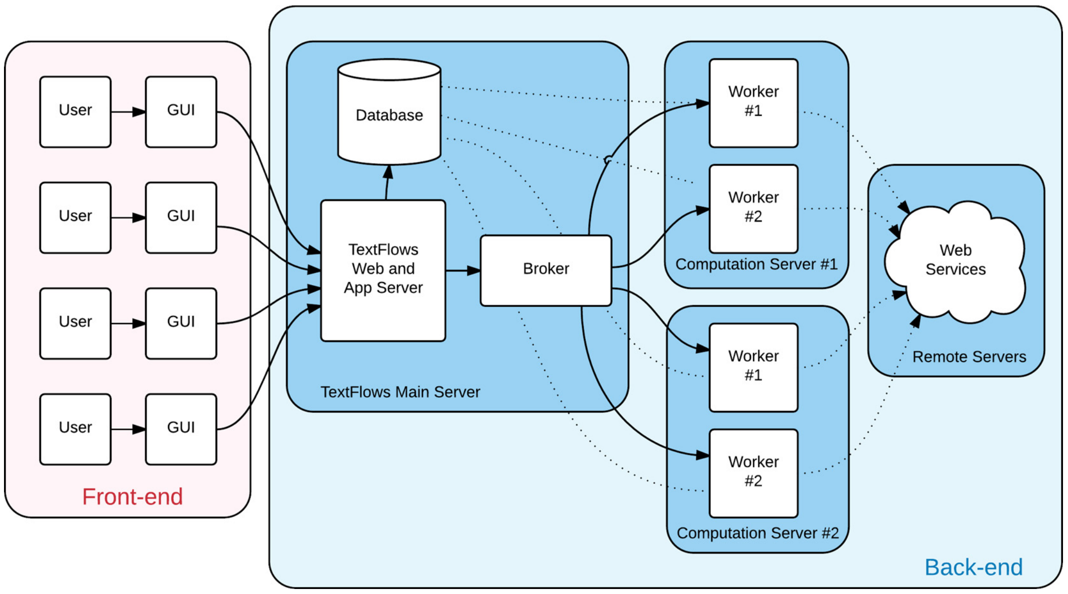


Fig. 1. An overview of the TextFlows architecture, separated into the front-end (in pink) and the back-end (in blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

is logically separated into the front-end (in pink) and the back-end (in blue). The back-end comprises a relational database for storing workflows, workers for task execution and a broker for delegating tasks from different queues to workers which can reside on different clusters of machines. The front-end is designed for user interaction with workflows through the graphical user interface in a web browser.

The back-end of the TextFlows platform uses Django,[4] a Python-based open source high-level web framework. Django follows the model–view–controller architectural design, which encourages rapid development, and includes an object-relational mapper and a template engine for efficient code generation. The object-relational mapper provides an interface which links objects to a database. This provides support for several database systems, such as PostgreSQL, MySQL, SQLite and Oracle databases. PostgreSQL is used in the public installation of TextFlows.

Without additional extensions Django is synchronous, sometimes also described as blocking. This means that a HTTP request will not be returned until all processing is complete. Though this is the expected behavior usually required in web applications, in the case of TextFlows we need tasks to run in the background without blocking. Furthermore, different system environment requirements of implemented libraries dictate that the TextFlows platform must be distributed across multiple machines (e.g., the LATINO library uses the. Net framework and performs best on Windows operating systems). TextFlows task queues are used as a mechanism to distribute work across threads and machines. This is performed via Celery,[5] which is a task queue system based on distributed message passing. Celery is focused on real-time operation, but supports scheduling as well. Dedicated worker processes monitor task queues for new work to perform and active workers execute different tasks concurrently on multiple servers. Tasks can be executed asynchronously (in the background) or synchronously (wait until ready). A Celery system can consist of multiple workers and brokers, thus supporting high availability and horizontal scaling.

Celery communicates via messages and uses a message broker to mediate between clients and workers. To initiate a task, a client adds a message to the queue, which the broker then delivers to a worker. The system used as a broker in TextFlows is RabbitMQ,[6] a complete and highly reliable enterprise messaging system based on the Advanced Message Queuing Protocol (AMQP). It offers not only exceptionally high reliability, but also high availability and scalability, which is vital for the TextFlows platform.

TextFlows uses the PySimpleSoap library[7] for integrations of web services as workflow components. PySimpleSoap is a lightweight Python library which provides an interface for client and server web service communication. Using PySimple-Soap we can not only import WSDL web services as workflow components, but also expose entire workflows as WSDL web services.

The client side of the TextFlows platform consists of operations that involve user interaction primarily through the graphical user interface (GUI) in a modern web browser. The graphical user interface is implemented in HTML and JavaScript,

[4] https://www.djangoproject.com.
[5] http://www.celeryproject.org.
[6] http://www.rabbitmq.com.
[7] https://code.google.com/p/pysimplesoap/.

with an extensive use of the jQuery library.[8] The jQuery library was designed to simplify client-side scripting and is the most popular JavaScript library in use today.[9] On top of jQuery we use the interaction library jQuery-UI,[10] which is a collection of GUI modules, animated visual effects, and themes. This library supports the option to make elements in the graphical user interface draggable, droppable, and selectable, which are the features supported by the TextFlows workflow canvas (cf. Section 3).

2.2. Key text mining concepts in TextFlows

The key concepts used in text mining and natural language processing are a document collection (or corpus), a single document (or text), and document features (or annotations) [9]. The following sections describe the model of corpora, documents and annotations in TextFlows. When designing TextFlows, emphasis was given to common representations that are passed among the majority of widgets: each TextFlows document collection is represented by an instance of the *AnnotatedDocumentCorpus (ADC)* class, a single text is an instance of the *AnnotatedDocument* class, while the features are instances of the *Annotation* class.

2.2.1. Annotated corpus

A document collection is any grouping of text documents that can be used in text analytics. Even though the size of a collection may vary from a few to millions of documents, from the text analysis perspective, more is better. In TextFlows, the Python class that represents a corpus of documents is called *AnnotatedDocumentCorpus (ADC)*. Every ADC instance contains not only a collection of documents which are part of this corpus but also the features that provide additional information about the corpus (e.g., authors, date of collection, facts and notes about the dataset, etc.). The features are stored in a simple key-value Python dictionary, where keys are strings and the values can store any Python object.

2.2.2. Annotated document

In TextFlows a single textual data unit within a collection—a document—is represented by the class *AnnotatedDocument*. An *AnnotatedDocument* object contains the text of the entire document, which may vary in size, e.g., from a single sentence to a whole book.

Similarly to *AnnotatedDocumentCorpus*, *AnnotatedDocument* instances in TextFlows also contain features which may provide information about a single document (e.g., author, date of publication, document length, assigned keywords, etc.).

2.2.3. Annotation

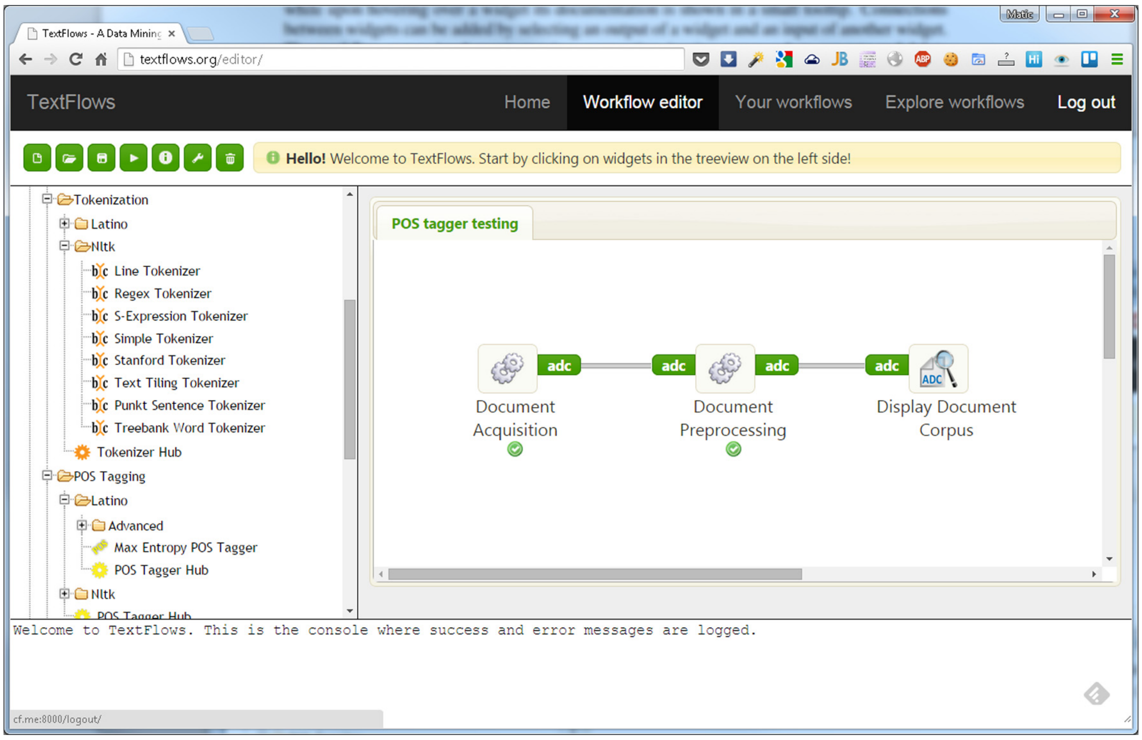
In TextFlows *Annotation* instances are used to mark parts of the document, e.g., words, sentences or terms. Every *Annotation* instance has two pointers: one to the start and another to the end of the annotation span in the document text. These pointers are represented as the character offset from the beginning of the document. *Annotation* instances also have a type attribute, which is assigned by the user and is used for grouping annotations of similar nature. As described in detail in Section 4, annotations can also contain key-value dictionaries of features, which are used by various taggers to annotate parts of document with a specific tag, e.g., annotations of type “token” that have a feature named “StopWord” with value “true”, represent stop words in the document.

3. The concept of workflows

The workflow model in the TextFlows platform consists of an abstract representation of workflows and workflow components. A workflow is an executable graphical representation of a complex procedure. The graphical user interface used for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The most basic unit component in a TextFlows workflow is a processing component, which is represented as a widget in the graphical representation. Considering its inputs and parameters every such component performs a task and outputs the results. Different processing components are linked via connections through which data is transferred from a widget’s output to another widget’s input. Additional inputs for a widget are its parameters, which the user enters into the widget text fields. The graphical user interface implements an easy-to-use way of arranging widgets on a canvas to form a graphical representation of a complex procedure.

The TextFlows graphical user interface, illustrated in Fig. 2, consists of a widget repository and a workflow canvas. The widget repository is a set of widgets ordered in a hierarchy of categories. Upon clicking on a widget in the repository, the widget is added as a building block to the canvas. While hovering over a widget its documentation is shown in as a tooltip. Connections between widgets can be added by clicking on an output of a widget and then on an input of another widget. The workflow canvas implements moving, connecting, and issuing commands to execute or delete widgets. Every action on the workflows canvas causes an asynchronous HTTP POST request to be sent to the server. After the requested operation is

[8] http://jquery.com.
[9] http://w3techs.com/technologies/overview/javascript_library/all.
[10] http://jqueryui.com.



Hello! Welcome to TextFlows. Start by clicking on widgets in the treeview on the left side!

```
Welcome to TextFlows. This is the console where success and error messages are logged.
```

Fig. 2. A screenshot of the TextFlows GUI opened in Google Chrome. On the top there is a toolbar for saving, deleting entire workflows. On the left is the widget repository giving a list of available widgets grouped by their functionality. By clicking on a widget in the repository it is added to the workflow construction canvas which is on the right. The console for displaying success and error message is located on the bottom of the interface.

validated on the server, a success or error message with additional information is passed to the user interface. An example of such a validation is checking for cycles in the workflows when connecting widgets.

3.1. *Workflow execution*

The workflow execution engine is responsible for executing the workflow widgets in the predefined order. Two such engines are integrated into the TextFlows platform, one implemented in Python and another in JavaScript. Sub-parts of workflows in subprocesses and loops are executed by the Python implementation, while the top-level workflows executed from the user interface (when the user actually needs to see the order of the executed widgets in real time) are processed by the JavaScript implementation. The former shows results only after their complete execution, while the later allows showing results of the execution in real time. With the Python implementation the server receives only one HTTP request for the entire part of the workflow, therefore multiprocessing had to be implemented manually. On the other hand, when a workflow is running with the JavaScript engine, it perpetually checks for widgets that are executable and executes them. Executable widgets are widgets which either have no predecessors or their predecessors have already been successfully executed. Whenever two or more independent widgets can be executed at the same time they are asynchronously executed in parallel. Each widget is executed through a separate asynchronous HTTP request. Every request is handled by the server separately and executes a single widget. When a widget is prepared to be executed, a task is added to a suitable task queue; as some widgets have special library requirements or even system requirement they are executed on a separated task queue with its dedicated workers. Celery communicates via messages and uses a message broker to find a suitable worker to which the task is delivered to. Dedicated worker processes monitor task queues for new work to perform. When the task is executed its result is saved into the database and returned to the client. The execution of a workflow is considered complete when there are no executable or running widgets.

3.2. *Workflow sharing*

Workflows in TextFlows are processed and stored on remote servers from where they can be accessed from anywhere, requiring only an Internet connection. By default each workflow can only be accessed by its author, although (s)he may also choose to make it publicly available. The TextFlows platform generates a specific URL for each workflow that has been saved as public. Users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by another user, a copy of the workflow is created on the fly and added to their private workflow repository. The workflow is copied with all the data to ensure that the experiments can be repeated. In this way, the users are able to tailor the workflow to their needs without modifying the original workflow.

4. The widget repository

This section presents the TextFlows widget repository. First we describe different types of widgets, followed by the presentation of widgets based on their functionality as they appear in the TextFlows widget repository.

Widgets in TextFlows are separated into four groups based on their type:

- *Regular widgets* perform specific tasks that transform the data from their inputs and parameters to data on their outputs, and provide success or error messages to the system. In the back-end such widgets are represented as Python functions, which receive (on every widget execution) a Python dictionary of inputs and parameters as widget arguments, perform a specific task and return a dictionary of outputs. The function is called each time the widget is executed. Widgets that implement complex long-running procedures can also display a progress bar, which shows the progress to the user in real time.
- *Visualization widgets* are extended versions of regular widgets as they also provide the ability to render an HTML template with JavaScript to the client's browser, which is useful for data visualizations and presentation of a more detailed feedback to the user. Visualization widgets differ from regular widgets by a secondary Python function which controls the rendering of the template. This function is only invoked when the widget is executed using the JavaScript execution engine, i.e., when it is not part of a subprocess.
- *Interactive widgets* are extensions of regular widgets as they pop-up an additional window during execution through which the user can interact with or manipulate the data (an example of an interactive widget is shown in Fig. 12). The entire procedure is implemented using three Python functions. The first function receives the widget's inputs and initialization parameters as its arguments and prepares the data for the second function. The second function renders (using an HTML template) a pop-up window that prompts the user to manipulate the data. The final function uses the user inputs, as well as the widget's inputs and parameters to produce the final output of the widget.
- *Workflow control widgets* provide additional workflow controls which allow the user to combine different workflow components into subprocesses, and provide different types of iterations through data (e.g., iteration through a list of classifiers, applying a classifier to all folds in cross-validation, etc.). This group of widgets consists of: *Subprocess*, *Input*, *Output*, *For Input*, *For Output*, *Cross Validation Input* and *Cross Validation Output* widgets. Whenever a *Subprocess* widget is added to a workflow, an initially empty workflow with no inputs and outputs is created. Then, when an *Input* or *Output* widget is attached to a subprocess workflow, an input or output is automatically added to the subprocess widget itself. Workflows can be indefinitely nested this way.
  Two additional variations of the input and output widgets exist in TextFlows. When a subprocess contains the *For Input* and *For Output* widgets, the workflow execution engine will emulate a for loop by attempting to break down the object on the input and executing the subprocess workflow once on every iteration. Using these controls a subprocess can be iteratively executed on a list. Similarly, if the user opts to use the *Cross Validation Input* and *Cross Validation Output* widgets the input data will be divided into the training and test dataset according to the selected number of folds; if the input data is labeled, stratified cross-validation [40] is performed.

The widget repository shows the hierarchy of all the widgets currently available in the TextFlows platform, grouped by their functionality. There are four top-level categories:

- *Text mining widgets:* a collection of implemented text mining widgets; these widgets are further divided based on their text mining functionality.
- *Basic widgets:* widgets that are responsible for creating and manipulating simple objects such as strings and integers.
- *Subprocess widgets:* a collection of workflow control widgets, which are required for visual programming of complex procedures.
- *WSDL imports:* workflow components representing the WSDL web-services that the user has imported.

In the following sections we present in more detail the text mining widgets based on their functionality in the order of appearance in the TextFlows widget repository.

4.1. Corpus and vocabulary acquisition

Document acquisition (import) is usually the first step of every task, where TextFlows employs various widgets to enable loading document corpora, labeling of documents with domain labels and converting them into the *AnnotatedDocumentCorpus* structure. We identified the following text document acquisition scenarios, which are also supported by the developed widgets:

- *Loading locally stored files in various application dependent formats*. In TextFlows document corpora can be uploaded from local files using the *Load Document Corpus* interactive widget. The entire dataset can be either a single text file (.doc, .docx, .pdf file formats are supported), where each line represents a separate document, or a zip of files in which a document is represented as a file. Apart from text, the files may optionally contain document titles, as well as multiple labels, which are encoded by the first word within a document prefixed by an exclamation mark, e.g., "!positive" is used to denote that the document belongs to the "positive" document category.
- *Acquiring documents using the WSDL+SOAP web services*. The user can integrate third-party web services as workflow components using the *Import webservice* button obtained from the bottom of the widget repository. Such integration allows for the inclusion of database web services into the workflow (e.g., PubMed offers a SOAP web service interface to access their database). TextFlows currently supports WSDL as the interface definition language for describing connections and requests to the web service and SOAP as the format for sending and receiving messages. The output of the imported web service widget can be connected to the *Load Document Corpus* widget that transforms the plain text input documents into the *AnnotatedDocumentCorpus* structure.
- *Selecting documents from SQL databases*. The TextFlows platform supports loading data only from MySQL databases via the *Load Document Corpus from MySQL* widget. Before execution the user enters the information which is required to connect to a database (e.g., user credentials, database address, database name, table name, column names, etc.) in order to retrieve the data from a MySQL database server. This widget then connects to the specified MySQL database server and returns the input columns representing document titles, texts and labels from the selected table. The final output of the widget is an automatically constructed *Annotated Document Corpus* object.
- *Crawling the Internet for gathering documents from web pages*. The *Crawl URL links* widget receives as an input a list of URLs, e.g., Wikipedia pages. First, every page is visited by an automatic crawler in order to gather the website's (HTML) content. Next, the Boilerpipe library [27] is used to extract the linguistically relevant textual content from the web page source. There are several content extraction methods available which can be selected by the user from the widget's properties. Finally, the *Crawl URL links* outputs the *Annotated Document Corpus* where document titles are represented with URLs and the extracted website texts become the document texts.
- *Collecting documents from snippets returned from web search engines*. In TextFlows the user can search the web using the *Search with Bing* and *Search with Faroo* widgets, which use the Microsoft Bing[11] and Faroo[12] as their web search engines, respectively. Both widgets require the user to enter the search query as well as the number of search results that the widget should return. The output of both widgets is a list of URLs which are returned by the web search engine. The execution the output can be connected to the *Crawl URL links* widget which will extract the web content for every URL.

The most straightforward technique to incorporate background knowledge about the documents and their domain is to use a controlled vocabulary. A controlled vocabulary is a lexicon of all terms that are relevant for a given domain. TextFlows allows the users not only to upload their own local vocabulary files but also gives them the possibility to use one of the implemented vocabulary construction tools, such as the *MeSH filter* widget which constructs a vocabulary containing all the terms that belong to the user selected descriptors from the MeSH hierarchy.[13]

4.2. Corpus manipulation and visualization

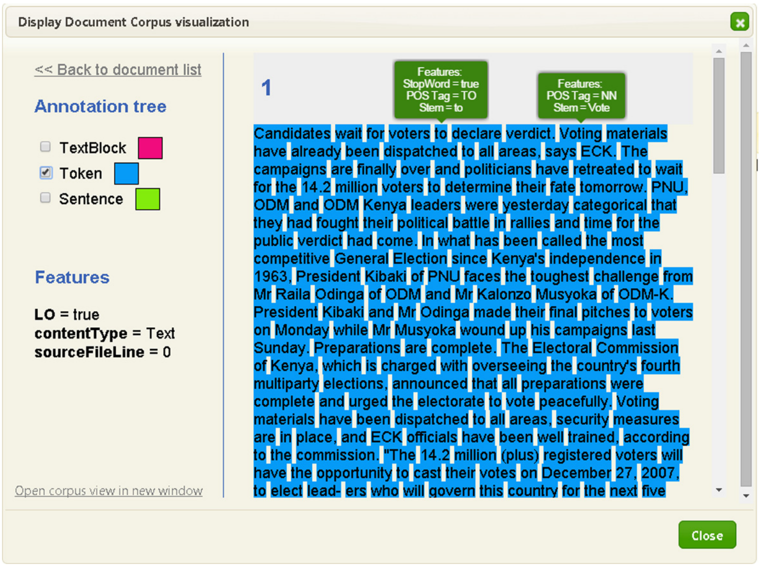
TextFlows implements widgets which allow the manipulation of *AnnotatedDocumentCorpus* (ADC) data objects. They allow the user to add new features, extract existing features from a document corpus, split document corpora (by either specifying conditions or by indices), merge different corpora, etc.

A special widget in the platform is *Document Corpus Viewer*, which visualizes the ADC data objects (note that TextFlows emphasizes the importance of the common ADC representation which is passed among the majority of widgets). As shown in Fig. 3, the *Document Corpus Viewer* interactive widget allows the user to check the results of individual widgets by visualizing the ADC data object from their outputs. Through its interactive interface the widget allows the user to select an individual document from the list of document snippets by simply clicking on it. This opens a new page with a detailed view of the selected document, as shown in Fig. 3.

4.3. Text preprocessing widgets

Preprocessing is a very important part of any form of knowledge extraction from text documents. Its main task is the transformation of unstructured data from text documents into a predefined well-structured data representation. In general, the task of preprocessing is to construct a set of relevant features from text documents. The set of all features selected for a given document collection is called a representational model [42,43,9]. Each document is represented by a vector of

[11] http://www.bing.com/.
[12] http://www.faroo.com/.
[13] MeSH (Medical Subject Headings) is a controlled vocabulary thesaurus used for indexing articles in PubMed, a database designed by The National Library of Medicine. The MeSH database is available online http://www.nlm.nih.gov/mesh.


Display Document Corpus visualization

<< Back to document list

Annotation tree

- [ ] TextBlock 🟥
- [✓] Token 🟦
- [ ] Sentence 🟩

Features

LO = true
contentType = Text
sourceFileLine = 0

Open corpus view in new window

1

Features:
StopWord = true
POS Tag = TO
Stem = to

Features:
POS Tag = NN
Stem = Vote

Candidates wait for voters to declare verdict. Voting materials have already been dispatched to all areas, says ECK. The campaigns are finally over and politicians have retreated to wait for the 14.2 million voters to determine their fate tomorrow. PNU, ODM and ODM Kenya leaders were yesterday categorical that they had fought their political battle in rallies and time for the public verdict had come. In what has been called the most competitive General Election since Kenya's independence in 1963, President Kibaki of PNU faces the toughest challenge from Mr Raila Odinga of ODM and Mr Kalonzo Musyoka of ODM-K. President Kibaki and Mr Odinga made their final pitches to voters on Monday while Mr Musyoka wound up his campaigns last Sunday. Preparations are complete. The Electoral Commission of Kenya, which is charged with overseeing the country's fourth multiparty elections, announced that all preparations were complete and urged the electorate to vote peacefully. Voting materials have been dispatched to all areas, security measures are in place, and ECK officials have been well trained, according to the commission. "The 14.2 million (plus) registered voters will have the opportunity to cast their votes on December 27, 2007, to elect lead- ers who will govern this country for the next five

Fig. 3. Document Corpus Viewer widget visualizes the *AnnotatedDocumentCorpus* data objects. This figure shows the document exploration page which displays a detailed view for a selected document. On the left side a list of applied tokenizations is shown, while on the right the document's full text is displayed. When the user selects a tokenization from the list, the tokens are highlighted directly on the text of the document with an alternative background color. As shown in the figure, annotation features are shown by hovering over a token.

numerical values, one for each feature of the selected representational model. Using this construction, we get the most standard text mining document representation, called feature vectors, where each numeric component of a vector is related to a feature and represents a weight related to the importance of the feature in the selected document. Typically, such text-based feature vectors are very sparse, as the majority of weights are equal to zero [9]. The goal of text preprocessing widgets is to extract a high quality feature vector for every document in a given document corpus.

Technically, our implementation employs the LATINO (Link Analysis and Text Mining Toolbox) software library [11], NLTK (Natural Language Toolkit) library [3] and scikit-learn library [32] for the text preprocessing needs. Together they contain the majority of elementary text preprocessing procedures as well as a large number of advanced procedures that support the conversion of a document corpus into a table of instances, thus converting every document into a table row representation of an instance.

In the TextFlows widget repository preprocessing techniques are based on standard text mining concepts [9] and are implemented as separate categories. Every category possesses a unique hub widget, which has the task of applying an given preprocessing technique from its category to the *AnnotatedDocumentCorpus* data object. Every such widget is library independent, meaning that it can execute objects from either LATINO, NTLK or scikit-learn libraries.

A standard collection of preprocessing techniques is listed below, together with sets of functionalities implemented in our platform:

1. *Tokenization*. In tokenization, meaningful tokens are identified in the character stream of the document, such as words or terms. TextFlows offers a large set of tokenizers: from LATINO, such as *Max Entropy Tokenizer* (word and sentence), *Unicode*, *Simple* and *Regex* tokenizers; various tokenizers from the NLTK library, from simpler ones, such as *Line*, *Regex*, *S-Expression*, *Simple*, to more complex ones, like *Stanford Tokenizer* and *Treebank Word Tokenizer*. Every tokenizer can be applied on a document corpus using the *Tokenizer Hub* widget. This hub receives as an input an ADC data object and a tokenizer instance, as well as two parameters entered by the user: the type of annotations to be tokenized (e.g., "TextBlock") and the type of annotations to be produced (e.g., "Sentence", "Token"). The *Tokenizer Hub* finds annotations of the input type and tokenizes them using the input tokenizer. The output of the hub is a new ADC object, which now contains the annotations of the new type. As described in the previous section, the results of any corpus tokenization can be visualized using the *Display Document Corpus* widget, as shown in Fig. 3.
2. *Stop word removal*. Stop words are predefined words from a language that do not carry relevant semantic information (e.g., articles, prepositions, conjunctions, etc.); the usual practice is to ignore them when building a feature set. In TextFlows we have three widgets which are used for stop word tagging: *Get StopWord Set* (outputs a predefined list of stop words for the user selected language—stop word lists for 18 languages, taken from Snowball,[14] are included in our library), *StopWords Tagger* (receives as an input a list of stop words and outputs a constructed tagger object, which tags the words from the input list as stopwords), *StopWord Tagger Hub* (responsible for applying a stop word tagger on a

[14] Snowball: A small string processing language designed for creating stemming algorithms: http://snowball.tartarus.org/.

document corpus). Similarly to the *Tokenization Hub*, the *Stop Word Tagger Hub* receives on its inputs an ADC data object and a stop word tagger instance. The user is able to enter two parameters: the type of annotations to be tagged (as a stop word) and a feature name, which is added (with a default value of 'true') to every annotation of the selected type, which the tagger marks as a stop word. The output of the hub is a new ADC object. Fig. 3 shows the visualization of a selected document from the output ADC data object using the *Display Document Corpus* widget. The stop word annotation features are shown by hovering over the documents tokens.

3. *Part-of-speech tagging*. Tagging annotates words with the appropriate part-of-speech (PoS) tags based on the context in which they appear. The TextFlows platform includes the LATINO's *Max Entropy PoS Tagger* and from NLTK the following taggers: *Affix PoS Tagger* (learns only on term affixes), *Brill's rule-based PoS Tagger* [5], *Classifier-based PoS Tagger* (requires a classifier and a pre-annotated dataset to learn the PoS tags) and a *PoS N-gram tagger* (learns on a pre-annotated dataset the most frequent tag for every n-gram). PoS tags are applied to ADC data using the *PoS Tagger Hub*. The *PoS Tagger Hub* requires, besides the usual element annotation type (default: "Token") and the PoS feature name (default: "PoS Tag"), an additional parameter input by the user: a sentence annotation type (default: "Sentence"). The hub tags element annotations in the context of sentence annotations by assigning them new features with values returned by the PoS tagger. Fig. 3 visualizes a selected document from the output ADC data object using the *Display Document Corpus* widget. The generated PoS tag features are shown when hovering over the document's tokens.
4. *Stemming and lemmatization*. This is the process of converting words/tokens into their stem or citation forms. The following stemmers/lemmatizers were taken from the LATINO library: *Stem Tagger Snowball* and the *Lemma Tagger LemmaGen* [23]. We have also implemented the following widgets which represent the corresponding algorithms from the NLTK library: *Porter Stemmer*, *Snowball Stemmmer*, *ISRI Stemmer*, *Regex Stemmer*, *RSLP Stemmer*, *Lancaster Stemmer*, and *WordNet Lemmatizer*. Analogous as in the stop word removal category, stemmers and lemmatizers can be applied using the *Stem/Lemma Tagger hub*. This widget receives as an input an ADC data object and a stemmer (or lematizer) instance and outputs a new ADC data object with an additional stemming added. The user can enter two parameters: the type of annotations to be stemmed ("Token" by default) and a feature name ("Stem" by default), which will be assigned to every annotation of the selected type as a key-value pair together with the stemmed value.

4.4. Bag-of-Words model

In the most general case, when dealing with raw text, the features are derived from text using only text preprocessing methods. The most common document representation model in text mining is the Bag-of-Words (BoW) model [9]. It uses all words (or, e.g., terms) as features, therefore the dimension of the feature space is equal to the number of different words in all of the documents. One of the most important characteristics of the described document features is the fact that they are usually very sparse [9], meaning that most of the features in a vector have zero weight. This sparsity is due to the fact that there are many different features (words, terms, concepts) in the document corpus; yet, a single document contains only a small subset of them. Consequently, the resulting feature matrix will have many (typically more than 99%) feature values that are zeros.

The TextFlows platform uses the Compressed Sparse Row (CSR) matrices, implemented in the scipy.sparse package[15] in order to be able to efficiently store the matrix of features in memory and also to speed up algebraic operations on vectors and matrices. The CSR matrices make no assumptions about the sparsity structure of the matrix, and do not store any unnecessary elements. Their main purpose is to put the subsequent non-zeros of the matrix rows in contiguous memory locations. Usually three vectors are created: one for storing floating-point numbers (*values*), and the other two for integers (*col_ ind, row_brk*). The values vector stores the values of the non-zero elements of the matrix, as they occur if reading through the matrix row by row. The *col_ind* vector stores the column indexes of the elements in the val vector, while the *row_brk* vector stores the locations in the *values* vector that start a new row in the dense original matrix. The storage savings using this approach are significant. Instead of storing $m * n$ elements, we only need to use $2 * \mathrm{nnz} + m$ storage locations, where m is the number of rows, n is the number of columns and *nnz* is the number of non-zeros in the dense matrix.

In the data mining modeling phase (i.e., document classification or text clustering), each document from the ADC structure needs to be represented as a set of document features it contains. In TextFlows the *Construct BoW Dataset and BoW Model Constructor* widget takes as an input an ADC data object and generates a sparse BoW model dataset (which can be then handed to a classifier). The widget takes as an input also several user defined parameters, which are taken into account when building the feature dataset:

- *Token Annotation.* This is the type of *Annotation* instances marking parts of the document (e.g., words, sentences or terms), which will be used for generating the vocabulary and the dataset.
- *Feature Name.* If present, the model will be constructed out of annotations' feature values instead of document text. For example, this is useful when we wish to build the BoW model using stems instead of the original word forms.

[15] http://docs.scipy.org/doc/scipy/reference/sparse.html.

- *Stop Word Feature Name.* This is an annotation feature name which is used to tag tokens as stop words. These tokens will be excluded from the BoW representational model. If the stop word feature name in not provided all tokens are included in the BoW space.
- *Label Document Feature Name.* This is the name of the document feature which will be used as a class label of the examples in the dataset. If blank, the generated sparse dataset will be unlabeled.
- *Maximum n-gram Length.* The upper bound of the range of n-values for different n-grams to be extracted. All values of n such that $1 \leq n \leq$ max_ngram will be used.
- *Minimum Word Frequency.* Cut-off frequency value for including an item into the vocabulary.
- *Word Weighting Type.* The user can select among various weighting models for assigning weights to features:
  – Binary. A feature weight is 1 if the corresponding term is present in the document, or zero otherwise.
  – Term occurrence. A feature weight is equal to the number of occurrences of the corresponding term. This weight is sometimes better than a simple binary value since frequently occurring terms are likely to be more relevant to the given text.
  – Term frequency. A weight is derived from the term occurrence by dividing the vector by the sum of all vector's weights.
  – TF-IDF. Term Frequency-Inverse Document Frequency [42] is the most common scheme for weighting features. For a given term w in document d from corpus D, the TF-IDF measure is defined as follows:

$$\mathrm{tfIdf}(w,d) = \mathrm{tf}(w,d) \times \log \frac{|D|}{|\{d \in D : w \in d\}|}, \tag{1}$$

  where $\mathrm{tf}(w,d)$ represents the number of times term w appears in document d. The reasoning behind the TF-IDF measure is to lower the weight of terms that appear in many documents as this is usually an indication of them being less important (e.g., stop-words). The appropriateness of this scheme was confirmed in numerous text mining problem solutions [14,9].
  – Safe TF-IDF. For a given term w in document d from corpus D, the Safe TF-IDF measure is defined as follows:

$$\mathrm{safeTfIdf}(w,d) = \mathrm{tf}(w,d) \times \log \frac{|D|}{|\{d \in D : w \in d\}| + 1}, \tag{2}$$

  This approach smoothens IDF weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. This prevents the occurrence of divisions by zero.
  – TF-IDF with sublinear TF scaling. It often seems unlikely that twenty occurrences of a term in a document truly carry twenty times the significance of a single occurrence. Accordingly, there has been considerable research into variants of term frequency that go beyond counting the number of occurrences of a term [31]. A common modification is to use the logarithm of the term frequency instead of *tf*, defined as:

$$\mathrm{wf}(w,d) = \begin{cases} 1 + \log \mathrm{tf}(w,d), & \text{if } \mathrm{tf}(w,d) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

- *Normalize Vectors.* The weighting methods can be further modified by vector normalization. If the user opts to use it in TextFlows the $L2$ regularization [30] is performed.

Besides the sparse BoW model dataset the *Construct BoW Dataset and BoW Model Constructor* also outputs a *BowModelConstructor* instance. This additional object contains settings which allow repetition of the feature construction steps on another document corpus. These settings include the input parameters, as well as the learned term weights and vocabulary.

An important widget in the Bag-of-Words category is the *Create BoW Dataset using BoW Model Constructor*. Its task is to apply the input *BowModelConstructor* instance to an input ADC data object and create a sparse feature dataset. This is useful, for instance, in every cross-validation fold where you need to build the test dataset's sparse matrix using the same settings (also including IDF term weights) used for building the training sparse matrix.

4.5. Document classification

Document classification (also called text categorization) refers to automated assigning of predefined categories to natural language texts. A primary application of text categorization systems is to assign subject categories to documents to support information retrieval, or to aid human indexers in assigning such categories. Text categorization components are also increasingly being used in natural language processing systems for data extraction. Classification techniques have been applied to spam filtering, language identification, genre classification, sentiment analysis, etc. The common approach to building a text classifier is to manually label a selected set of documents to predefined categories or classes, and use them to train a classifier. The trained classifier can then be used to assign class labels to unseen documents based on the words they contain.

The term *supervised learning* refers to the above-described approach to automatically building a text classifier from training documents, which have been labeled (often manually) with predefined classes. The TextFlows platform currently contains only the supervised learning approaches from the LATINO, NTLK and scikit-learn libraries. Every widget contains input parameters which are used to construct the classifier object. From the LATINO library we integrated *Nearest Centroid Classifier, Naive Bayes Classifier, SVM (Binary and Multiclass) Classifier, Majority Classifier, Maximum Entropy Classifier, kNN (fast and regular version) Classifier*, while the NLTK library contributes an additional Naive Bayes Classifier. The following widgets represent classifiers from the scikit-learn library: *Decision Tree Classifier, Multinomial Naive Bayes Classifier, Gaussian Naive Bayes Classifier, k-Nearest Neighbors Classifier, SVM Linear Classifier, SVM Classifier*

For training and applying classifiers TextFlows offers two dedicated widgets: *Train Classifier Hub* and *Apply Classifier Hub*. The *Train Classifier Hub* receives on its inputs a sparse feature dataset object and an untrained classifier. Its function is to fit the classifier model according to the given training data. The final outcome of this widget is a trained classifier object.

The *Apply Classifier Hub* receives a trained classifier object and returns predicted class probabilities for every new document from the input test dataset, as well as the test dataset with a new predicted labels column.

4.6. Literature-based discovery

Literature-based discovery refers to the use of papers and other academic publications (the "literature") to find new relationships between existing knowledge (the "discovery"). [48] presented an approach to discovering unknown relations between previously unrelated concepts by identifying interesting bridging terms (b-terms) appearing in two separate document sets and bearing the potential to indirectly connect the two concepts under investigation.

The TextFlows platform includes a dedicated category with several widgets which support literature-based discovery. For example, literature-based discovery is supported through the incorporated *Explore In CrossBee* widget, which provides a link to the web application CrossBee (Cross-Context Bisociation Explorer) developed by [21]. The web application can be used for cross-domain knowledge discovery from a given pair of user-selected document corpora.

4.7. Noise detection

Noise filtering is frequently used in data preprocessing to improve the accuracy of induced classifiers. TextFlows incorporates an ensemble-based noise ranking methodology for explicit noise and outlier identification, named NoiseRank [45], which was modified to work with texts and TextFlows ADC data objects. Its main aim is to detect noisy instances for improved data understanding, data cleaning and outlier identification. NoiseRank was previously successfully applied to a real-life medical problem [45]. We show an example of using the NoiseRank methodology on a task of outlier detection in document corpora in Section 5.3.

4.8. Evaluation and visualization

The TextFlows platform enables users to create interactive charts for easy and intuitive evaluation of performance results. It includes standard performance visualizations used in machine learning, data mining, information retrieval, etc. Notably, the TextFlows platform includes a full methodology, named VIPER [45,46], i.e., a visualization approach that displays the results as points in the two dimensional precision-recall space. The platform contains several visual performance evaluation widgets, which result in interactive performance charts that can be saved and exported to several formats.

- *Scatter charts*. These include ROC space charts and PR space charts.
- *Curve charts*. These include PR curves, Cost curves, Lift curves, ROC curves, Kendall curves and Rate-driven curves.
- *Column charts*. These are general column charts for visualizing multiple performance measures for a set of algorithms.

While VIPER visualizations appear to be straightforward, this visualization toolbox is innovative and very useful for text analytics. An example is visual comparison of F-value results of different text analysis tools, including F-isoline-based text classifier comparison, which is not supported by any other visualization tool. We demonstrate several implemented visualization techniques in Section 5.

5. Selected use cases

In this section we demonstrate advanced features of TextFlows on three use cases: a comparison of classifiers from different libraries for a text categorization problem, a comparison of PoS taggers on the same categorization problem and outlier detection in document corpora.

In our experiments we used a corpus of documents, presented by [35] and [36], which was originally collected by the IPrA Research Center, University of Antwerp. The document corpus contains 464 articles (about 320,000 words) concerning Kenyan presidential and parliamentary elections, held on 27th December 2007, and the crisis following the elections. The documents originate from six different daily newspapers in English, covering the time period from 22nd December 2007 to 29th February 2008. Articles from the US and British press (The New York Times, The Washington, The Independent, The Times and Post) form the class label "Western" (WE) and articles from local Kenyan newspapers Daily Nation and The Standard are categorized as "Local" (LO).

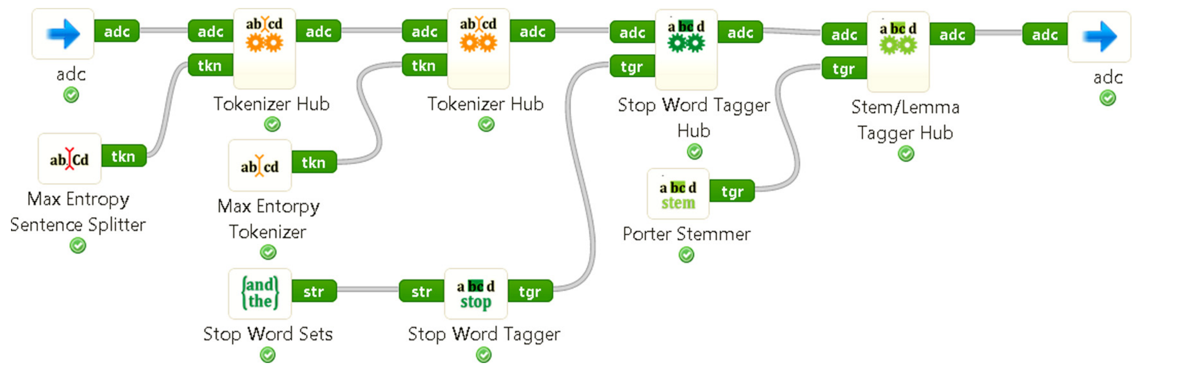


Fig. 4. Document preprocessing workflow, available at http://textflows.org/workflow/604/. The same workflow is implemented as a subprocess in the three use cases presented in this paper.

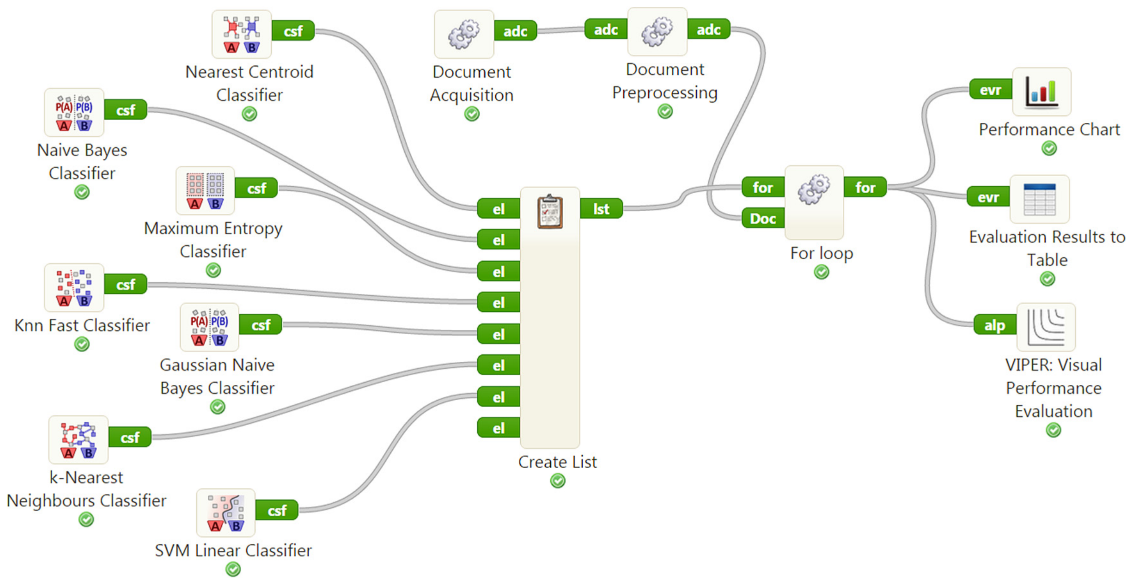

Nearest Centroid Classifier
Naive Bayes Classifier
Maximum Entropy Classifier
Knn Fast Classifier
Gaussian Naive Bayes Classifier
k-Nearest Neighbours Classifier
SVM Linear Classifier
csf
Document Acquisition
Document Preprocessing
adc
el
lst
Create List
for
Doc
For loop
evr
alp
Performance Chart
Evaluation Results to Table
VIPER: Visual Performance Evaluation

Fig. 5. The workflow for evaluating 7 classifiers from different text mining libraries. The workflow is available at http://textflows.org/workflow/350/.

The two common steps in the three presented use cases are the *Document acquisition* and *Document preprocessing* steps. These two steps were implemented as subprocesses in the main workflows for the three use cases, as shown in Figs. 5, 9 and 11.

In all three workflows, *Document acquisition* is the first step of the methodology and is responsible for, first, loading the locally stored text file (containing the Kenyan elections document corpus), then labeling the documents with appropriate domain labels and finally converting them into the *AnnotatedDocumentCorpus* data object.

Fig. 4 shows the subprocess in the TextFlows platform for the second step in all three presented methodologies: the *Document preprocessing* workflow. As described in Section 4, category specific hubs are used for applying different preprocessing objects to the *AnnotatedDocumentCorpus* data object. The documents are first split into sentences with LATINO's *Max Entropy Sentence Splitter* and then the sentences are split into tokens with LATINO's *Max Entropy Tokenizer*. Some of these tokens are tagged as stop words using the *Stop Word Tagger* with the predefined Snowball list of English stop words. Finally, the *Porter Stemmer* is used for converting tokens into their stems.

5.1. Classifier comparison for text categorization

In this section we propose a workflow for classifier evaluation on the Kenyan elections dataset. In our experiments we compared 7 different classifiers from different text mining libraries. As shown in Fig. 5, we compared 4 classifiers from LATINO (*Nearest Centroid Classifier, Naive Bayes Classifier, Maximum Entropy Classifier, kNN Fast Classifier*) and 3 classifiers implemented in the scikit-learn library (*Gaussian Naive Bayes Classifier, k-Nearest Neighbors Classifier, SVM Linear Classifier*).

Every algorithm was evaluated using 10-fold stratified cross-validation, as shown in Fig. 6. All cross-validations were performed using the same seed in order to ensure the data was equally split for different classifiers. Fig. 7 shows the

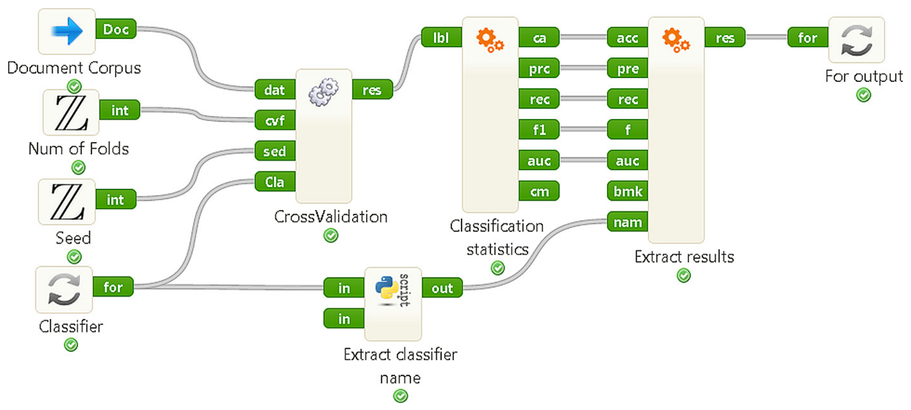


Fig. 6. The *For loop* subprocess which evaluates the input classifier using 10-fold stratified cross-validation (with a constant seed) and extracts the obtained results, which are then visualized as shown in Fig. 5.

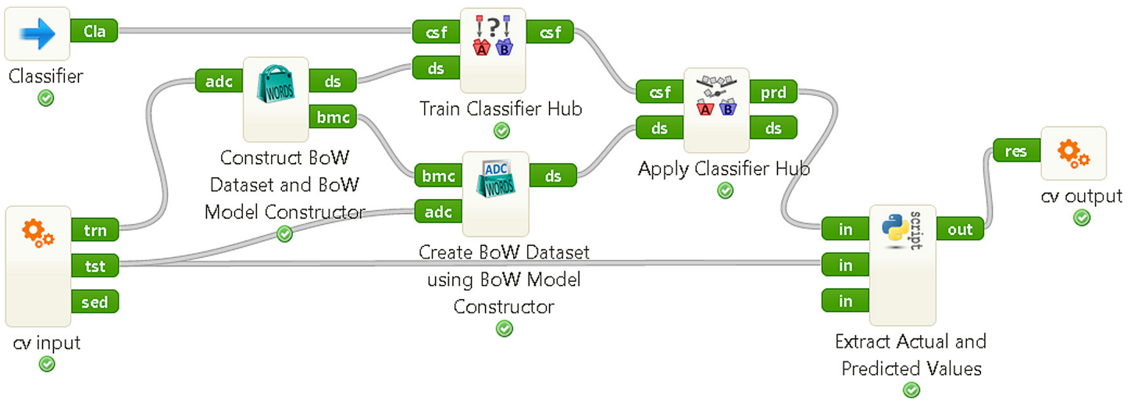

Cla
Classifier
csf
ds
csf
Train Classifier Hub
adc
ds
bmc
Construct BoW Dataset and BoW Model Constructor
bmc
adc
ds
Create BoW Dataset using BoW Model Constructor
csf
ds
prd
ds
Apply Classifier Hub
trn
tst
sed
cv input
in
in
in
out
Extract Actual and Predicted Values
res
cv output

Fig. 7. The subprocess workflow representing the methodology in every fold for the used 10-fold cross-validation. First, the construction of the training and testing sparse matrices is performed. Next, the input classifier model is fitted to the training dataset and used to predict class probabilities for every new document from the input test dataset. The subprocess returns constructed pairs of actual and predicted classes.

methodology behind the *CrossValidation* subprocess which was executed for every cross-validation fold. First, a sparse BoW model dataset and the *BowModelConstructor* instance were generated by the *Construct BoW Dataset and BoW Model Constructor*. We constructed unigrams (n-grams where $n = 1$) out of stemmed values from word tokens, while disregarding stop words in the process. The Bag-of-Words vector space was calculated using the TF-IDF weighting scheme.

The test sparse matrix is required to be constructed using the same parameter settings, IDFs and vocabulary as the training sparse matrix. Therefore, we applied the BoW constructor object (output of the *Construct BoW Dataset and BoW Model Constructor*) to the test ADC data object using the *Create BoW Dataset using the BoW Model Constructor*.

After calculating the training and testing sparse matrices, we fitted the input classifier model to the training dataset using the *Train Classifier Hub*. Next, the *Apply Classifier Hub* received the trained classifier object and returned predicted class probabilities for every new document from the input test dataset. The *Extract Actual and Predicted Values* widget used these probabilities and constructed pairs of actual and predicted classes. These pairs were returned from the *CrossValidation* subprocess and used for calculating different metrics, as shown in Fig. 6.

The results of cross-validation (precision, recall, F-score) were connected to the input of the *VIPER: Visual Performance Evaluation* widget, as shown in Fig. 5, while Fig. 8 presents its visualization of classifier evaluation in the precision-recall plane, where each point represents the result of an algorithm (for the selected target class “Lo”). Points closer to the upper-right corner have higher precision and recall values. F-measure values are presented as isolines (contour lines), which allows a simple comparison of algorithm performance.

Fig. 8 shows that in terms of the F-measure, scikit-learn’s *SVM Linear Classifier* and LATINO’s *Maximum Entropy Classifier* achieved the best results: both algorithms generally achieved a higher percentage of correctly classified examples (higher recall score), and also a slightly higher percentage of correctly classified examples of the target class (higher precision score) compared to other classifiers used. A somewhat lower performance was achieved using LATINO’s *Nearest Centroid Classifier* classifiers, while the k-nearest neighbor and Naive Bayes classifiers performed worse. Detailed results are presented in Table 1.

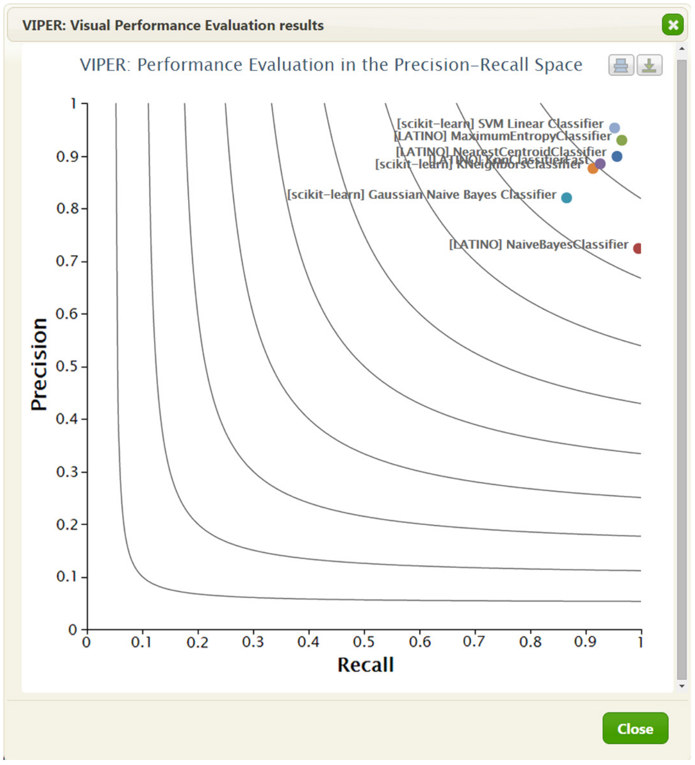


Fig. 8. The VIPER visualization showing evaluation of classifiers from different libraries. This visualization is the result of the workflow presented in Fig. 5.

Table 1
Classifier evaluation on the Kenyan elections dataset.

Library	Classifier	Recall	Prec.	F1 score	Classif. accuracy	AUC
LATINO	Nearest Centroid Classifier	0.96	0.90	0.93	92.42%	0.92
LATINO	Naive Bayes Classifier	1.00	0.72	0.84	80.74%	0.81
LATINO	Maximum Entropy Classifier	0.97	0.93	0.95	94.59%	0.95
LATINO	kNN Fast Classifier	0.93	0.88	0.90	90.26%	0.90
scikit-learn	Gaussian Naive Bayes Classifier	0.87	0.82	0.84	83.77%	0.84
scikit-learn	k-Nearest Neighbors Classifier	0.91	0.88	0.89	89.18%	0.89
scikit-learn	SVM Linear Classifier	0.95	0.95	0.95	95.24%	0.95

The workflow was run within a virtual machine with a dedicated 1 core (Intel i7 2600k) and 4 GB of RAM. The running time for the document acquisition and document preprocessing step is 161 seconds while the comparison of classifiers with the 10 fold cross-validation takes 1486 seconds. The runtime of the entire workflow was 1652 seconds.

5.2. Part-of-speech tagger comparison for text categorization

In this section we describe a workflow for the evaluation of different part-of-speech taggers on the Kenyan elections dataset. In the experiments we compared five different PoS taggers: *English Maximum Entropy PoS Tagger* (from LATINO library) and *PoS Affix Tagger, PoS N-gram Tagger, PoS Brill's rule-based Tagger, PoS Classifier-based Tagger* (from NLTK library). As shown in Fig. 9, PoS tagging widgets from NLTK require tagged sentence data as input, which is used for training the PoS tagger. The TextFlows platform already contains several tagged datasets, which come as a part of the NLTK library, and can be added to workflows through the *NLTK Document Corpus* widget. In this use case we used the annotated Brown corpus.[16]

The training process involves inspecting the tag of each word and storing the most likely tag for any word in a dictionary, which is stored inside the tagger. As it happens, once we have processed several thousand words of English text, most new words will be nouns. In cases when the NLTK PoS tagger is unable to assign a tag from its learned lookup table, it can use a backoff tagger from its input. As shown in Fig. 9 we have set the *PoS Default Tagger* as the backoff tagger for all NLTK

[16] Description of the Brown corpus is available at http://www.nltk.org/book/ch02.html#tab-brown-sources.

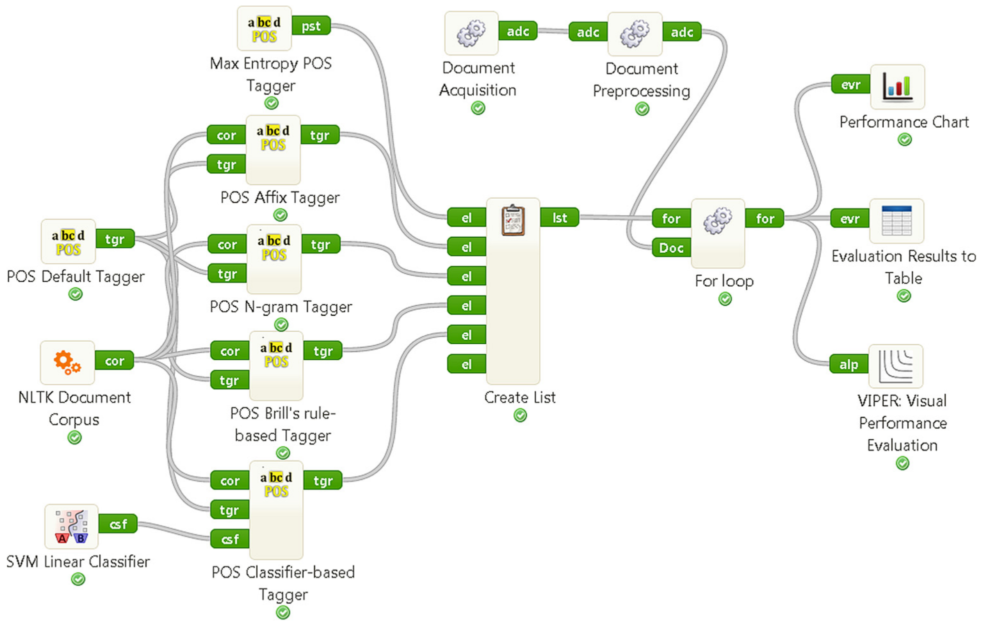


Fig. 9. The workflow used for evaluation of 5 different PoS taggers from various text mining libraries on a text categorization problem. The workflow is available at http://textflows.org/workflow/355/.

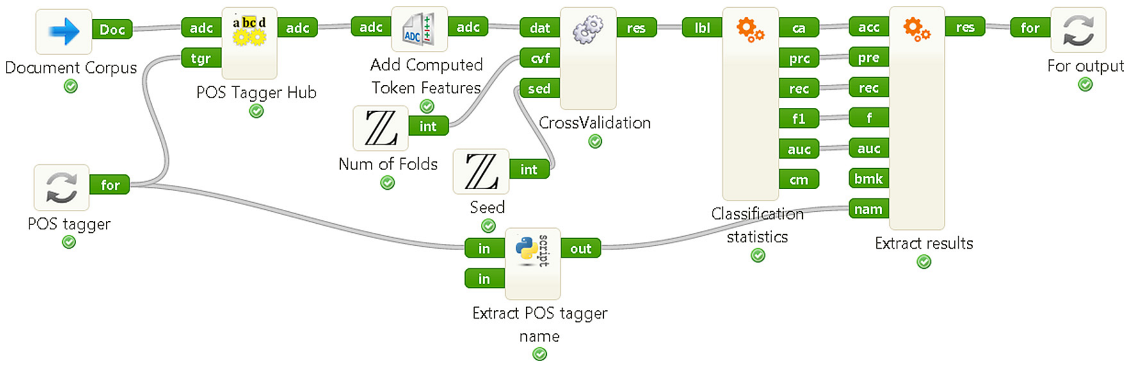


Fig. 10. The subprocess workflow representing the methodology in every fold for the used 10-fold cross-validation. First, the input PoS tagger is applied to the document corpus. Next, the construction of the training and testing sparse matrices is performed. Then, a linear SVM classifier is fitted to the training dataset and used to predict class probabilities for every new document from the input test dataset. Last, the subprocess returns constructed pairs of actual and predicted classes.

PoS taggers. The *PoS Default Tagger* assigns the input tag (e.g., "NN", which is the PoS tag representing a noun) to every single word. Whenever the initial PoS tagger cannot assign a tag to a token it will invoke the input backoff tagger and thus tag the token as a noun. This improves the robustness of the language processing system. The *PoS Classifier-based Tagger* widget also requires input of a classifier, which is learned to predict PoS tags based on the pre-annotated dataset. Every PoS tagger was applied to the document corpus, as shown in Fig. 10. In every iteration (over the list of PoS taggers) of the for loop the input PoS tagger is applied to the tokenized sentences of preprocessed ADC data object using the *PoS Tagger Hub* by generating new features with name "PoS tag" on every elementary (word) token. In order to use PoS tags together with stemmed values, we constructed (for every token) new features named "Stem with PoS" using the *Add Computed Token Features* widget. These features were later used in the CrossValidation subprocess to generate the BoW models. The values of the "Stem with PoS" features were constructed using the *Add Computed Token Features* widget as a combination of stems and PoS tags: "Stem_PoS tag".

Next, 10-fold stratified cross-validation was performed on the generated PoS tagged ADC data objects. Similarly as in the classifier evaluation use case, all cross-validations were performed using the same seed in order to ensure the data was equally split for all PoS taggers. The methodology behind the *CrossValidation* subprocess, which is executed on every cross-validation fold, is similar to the methodology presented in Fig. 7. The only two differences are that cross-validation does not receive a classifier on its input—instead it always uses scikit-learn's linear SVM classifier—and that the *Construct*

Table 2
PoS tagger evaluation on the Kenyan elections database.

Library	Tagger	Recall	Precision	F1 score	Classif. accuracy	AUC
	no PoS tagger	0.98	0.93	0.95	95.24%	0.95
LATINO	Maximum Entropy PoS Tagger	0.98	0.94	0.96	96.10%	0.96
NLTK	PoS Affix Tagger	0.98	0.94	0.96	95.67%	0.96
NLTK	PoS Ngram Tagger	0.98	0.95	0.96	96.10%	0.96
NLTK	PoS Brill Tagger	0.97	0.93	0.95	95.24%	0.95
NLTK+scikit-learn	PoS Classifier Based Tagger (using SVM Linear Classifier)	0.98	0.95	0.96	96.32%	0.96

BoW Dataset and BoW Model Constructor widget uses features constructed by the *Add Computed Token Features* widget instead of stemmed values.

Table 2 shows the results of the presented PoS tagger evaluation workflow. The first row in the table shows the classification results without applying a PoS tagger (see row 3 of Table 1). We see that the usage of a PoS tagger increases the performance of the classifier. The best results were obtained using the NLTK's *PoS Classifier Based Tagger*, which in combination with the LATINO's *Maximum Entropy Classifier* achieved a slightly higher classification accuracy on the Kenyan elections dataset compared to the other PoS taggers.

The experiments were run using the same resources as in the classifier evaluation example—a virtual machine with a setting of 1 core and 4 GB of RAM. The execution time of the entire workflow was 1913 seconds, where 158 seconds were used for document acquisition and preprocessing, while the for loop which compares PoS taggers took 1747 seconds to execute.

5.3. Outlier document detection in categorized document corpora

In this section we propose a workflow for detecting atypical, unusual and/or irregular documents on the Kenyan elections dataset. The idea behind irregularity detection in categorized document corpora is based on early noise filtering approaches by [6], who used a classifier as a tool for detecting noisy instances in data. Noise detection approaches identify irregularities and errors in data and are therefore suitable also for detecting atypical documents in categorized document corpora, which can be considered as outliers of their own document category.

The aim of the NoiseRank (ensemble-based noise detection and ranking) methodology, proposed by [47,45], is to support domain experts in identifying noisy, outlier or erroneous data instances. The user should be able to select the noise detection algorithms to be used in the ensemble-based noise detection process. We have implemented this methodology as a workflow in TextFlows, which now offers widgets implementing classification and saturation noise filters, and enables the inclusion of external user specific noise detection algorithms available as web services. Fig. 11 presents the NoiseRank workflow using the implemented classifiers used for class noise detection.

The NoiseRank methodology workflow returns a visual representation of a list of potential outlier instances, ranked according to the decreasing number of noise detection algorithms which identified an instance as noisy, due to its classification into a class different from its own class label. The ability of NoiseRank to obtain atypical documents was tested on the Kenyan elections corpus. The implemented voting-based irregularity detection method uses four different classifiers acting as noise detection algorithms by identifying misclassified instances.

As in the experiments of Section 5.1 and Section 5.2 we ran the workflow on a virtual machine with 1 CPU core and 4 GB of RAM. The execution time of the entire workflow was 202 seconds, where 148 seconds were used for document acquisition and preprocessing, while the BoW model construction and NoiseRank widget took 52 seconds to execute.

Fig. 12 shows the obtained set of atypical/irregular articles grouped and ranked according to the number of noise detection algorithms that identified them as irregular.

6. Related work

An extensive survey of workflow management platforms, including general data mining platforms—such as RapidMiner [29], Weka [49] and Orange [8]—is out of the scope of this paper, however, we do describe and compare to TextFlows other text mining and natural language processing platforms, starting with a comparison to its predecessor ClowdFlows [28]. In the rest of this section, we concentrate on the key features of the presented platforms: visual programming and execution of scientific workflows, diversity of workflow components, service-oriented architectures, remote workflow execution, big data processing, stream mining, and data sharing. This overview is followed by a comparison of the presented systems with TextFlows.

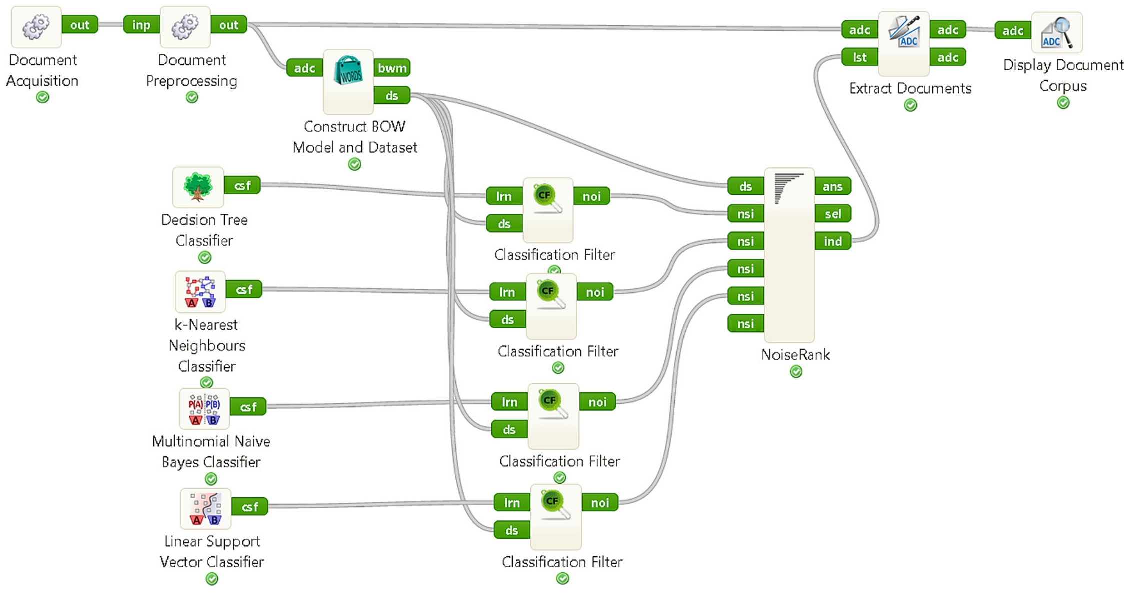
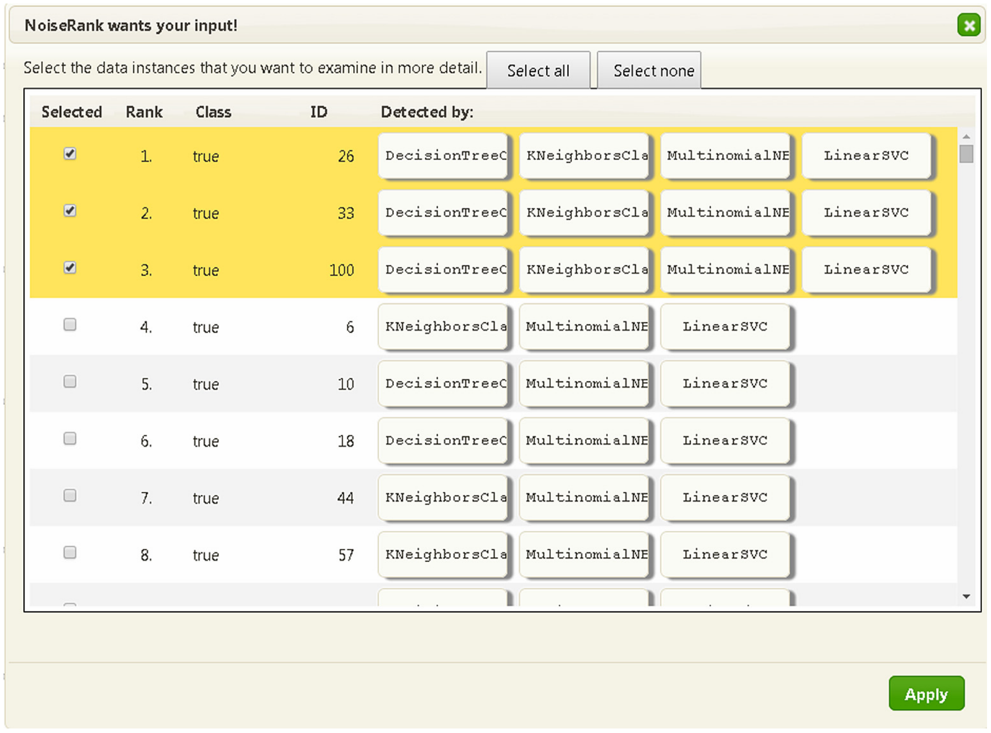


Fig. 11. Example workflow of the NoiseRank methodology, which is available at http://textflows.org/workflow/360/.



Select the data instances that you want to examine in more detail.

Selected	Rank	Class	ID	Detected by:			
☑	1.	true	26	DecisionTreeC	KNeighborsCla	MultinomialNB	LinearSVC
☑	2.	true	33	DecisionTreeC	KNeighborsCla	MultinomialNB	LinearSVC
☑	3.	true	100	DecisionTreeC	KNeighborsCla	MultinomialNB	LinearSVC
☐	4.	true	6	KNeighborsCla	MultinomialNB	LinearSVC	
☐	5.	true	10	DecisionTreeC	MultinomialNB	LinearSVC	
☐	6.	true	18	DecisionTreeC	MultinomialNB	LinearSVC	
☐	7.	true	44	KNeighborsCla	MultinomialNB	LinearSVC	
☐	8.	true	57	KNeighborsCla	MultinomialNB	LinearSVC	

Fig. 12. The NoiseRank interactive widget where the user gets a visual representation of a list of top-ranked potentially noisy instances, which are misclassified according to a decreasing number of elementary noise detection algorithms which identified the instance as noisy. The user can decide which documents he wishes to exclude from the document corpus.

6.1. Comparison with ClowdFlows

TextFlows is a heavily modified fork of the ClowdFlows data mining platform [28], which is a general workflow construction and execution tool for data mining and machine learning. The platform incorporates workflow components for numerous data mining and machine learning projects—including WEKA [49] and Orange [8]—while severely lacking text mining and natural language processing components. Moreover, despite being very useful for workflow construction by the informed developers and end-users, ClowdFlows currently suffers from a somewhat disorganized roster of workflow components which may be incompatible with each other. As a result, new users sometimes struggle to construct functioning workflows as there are too many mutually incompatible components.

We created a fork of ClowdFlows in order to maintain cohesion of incorporated text mining and natural language processing workflow components. TextFlows has a completely redesigned roster of workflow components, which are now highly compatible with each other within the platform and easier to use for the expert and novice users alike. In contrast to ClowdFlows, where there are no guidelines for sorting workflow components into categories, TextFlows sorts the components based on the functionality of the components As a result, all widgets that perform similar functions are located in the same category. We have also enriched the user interface with extra information about the workflow components and written the documentation for the provided workflow components and their inputs and outputs.

While we introduced a completely new common text representation structure (cf. Section 2.2), a new widget structure as well as numerous new text mining and new NLP components (cf. Section 4) and workflows (cf. Section 5 for selected examples), the underlying architectures of ClowdFlows and TextFlows remain similar. To summarize, TextFlows is built on top of ClowdFlows, meaning that both the widget execution engine and the core of the ClowdFlows user interface are present in TextFlows. TextFlows still benefits from all security updates, bug fixes and feature upgrades that ClowdFlows receives.

6.2. *Survey of workflow management environments for natural language processing*

Workflow Management Systems have in the last years become a very hot topic, mostly in the field of bioinformatics and other natural sciences. Lately, however, this trend has also spread to NLP, as evidenced also by recent workshops at the main NLP conferences, e.g., the Workshop “Language Technology Service Platforms: Synergies, Standards, Sharing” at the 9th Language Resources and Evaluation Conference (LREC 2014)[17] and the Workshop on Open Infrastructures and Analysis Frameworks for HLT[18] at the 25th Conference on Computational Linguistics, (COLING 2014).

The current situation with Workflow Management Systems for NLP is very fluid; some well-established systems are slowly starting to be used for NLP applications, while at the same time, new ones are being developed, specifically targeted to NLP. In this section we first overview some of the more important NLP-related Workflow Management Systems, where each platform/project is introduced and its most salient characteristics presented.

6.2.1. *Taverna*

The set of tools developed by the myGrid[19] team in the U.K. and used primarily for bioinformatics and other life sciences research (having in mind experiment replication) is currently probably the most advanced, richest and easiest to use Workflow Management (Eco)System, and consists of the following main components:

- SoapLab[20] [44] which provides a convenient way to generate web services for command-line software;
- Taverna[21] [16] with its Workflow editor and Server;
- BioCatalogue[22] [2], a registry (for life sciences) where web services can be shared, searched for, annotated with tags, etc.
- myExperiment[23] [41], a social network for sharing, reusing and repurposing public workflows.

As the most important part of the myGrid offerings, we here discuss Taverna, which is conceived as a suite of tools used to design and execute scientific workflows. It combines distributed Web Services and/or local tools into complex analysis pipelines, which can be executed on local desktop machines or through larger infrastructures, such as supercomputers, Grids or cloud environments, using the Taverna Server. The Server allows for workflow execution from web browsers, or through third-party clients; it supports WSDL, REST, GRID and Cloud services, local and distributed command-line scripts, as well as other types of services, such as R-Scripts.

Taverna is connected to myExperiment and BioCatalogue, as well as to other service registries, such as BioVeL,[24] the Biodiversity Virtual e-Laboratory. In addition, Taverna offers an Interaction Service, which enables scientists to select parameters and data during workflow execution, and the Provenance suite, which records service invocations, intermediate and final workflow results.

Taverna workflows can be designed and executed in several ways, to serve different types of workflow users. First, the Taverna Workbench—once downloaded to a local machine—provides an environment for scientists to develop new workflows and test new analysis methods, by either developing workflows from scratch, or by composing them from existing

[17] http://lrec2014.lrec-conf.org/.
[18] http://glicom.upf.edu/OIAF4HLT/.
[19] http://www.mygrid.org.uk/.
[20] http://soaplab.sourceforge.net/soaplab2/.
[21] http://www.taverna.org.uk/.
[22] https://www.biocatalogue.org/.
[23] http://www.myexperiment.org/.
[24] http://www.biovel.eu/.

workflows. Second, workflows can be executed directly through a Taverna Server, which is an environment for serving finished workflows to a larger community of scientists. Here, a single installation of the Server provides access to a collection of workflows, normally through a web interface, called the Taverna Player; in this case, no installation or in-depth knowledge of the workflows is required, but workflows cannot be changed nor can new workflows be added to the collections. The third mode of execution is via a Taverna Lite installation, which provides an intermediate solution, as it allows users not only to run workflows through the web but also to upload new workflows from e.g., myExperiment or other sources. This means that Taverna Lite installations require user authentication, but no local software installation by regular users, as workflow execution also occurs on a server.

The Taverna Workbench, as the most sophisticated means of composing workflows, needs to be first downloaded and installed on a local machine (Windows, Linux or Mac OS X). For third-party services that require a login, Taverna allows credentials to be added at run-time, or to be stored in a purpose-built credential manager. The Workbench allows users to identify and combine services by dragging and dropping them onto the workflow design panel. The services can be from third parties (using e.g., WSDL or REST), but typically also contain local scripts for formatting data and managing service compatibility, known as shim services. Most workflows will need shim services as the analysis services are not usually designed to work together and, therefore, often have incompatible input and output formats. A workflow can also contain nested workflows, so workflows can be components of other workflows. Nested workflows can, for example, control retrieval of data from asynchronous services. Here the nested workflow is executed repeatedly until results are available and the control links between its output and downstream services pause the remainder of the workflow until all preceding results are available. As the workflow runs, the results panel shows progress through the workflow and iterations over data, as well as any errors if there are problems with executions.

The Taverna Server, which executes workflows, can also be downloaded and configured to run with or without login restrictions. It is written in Java, has to be installed on Unix and uses Tomcat with, for secure mode, HTTPS and SSL host certificate. There are various public installations of the Taverna Server, with the best know being the already mentioned BioVeL portal with workflows from the area of biodiversity.

As mentioned, Taverna is currently the most developed and advanced (open source) Workflow Management System, with a host of features and connection capabilities, including fine-grained access management. It is also very popular, e.g., myExperiment currently contains over 2000 Taverna workflows. By far the largest part of the user community is from the field of bioinformatics and other life sciences, where Taverna workflows are typically used in the areas of high-throughput analyses or for evidence gathering methods involving data mining.

Given the power and versatility of Taverna and other myGrid platforms, it is surprising that—apart from a few basic workflows submitted by various individuals—there are few public NLP workflows have been so far implemented in it. In connection with biomedical informatics, there is one published experiment in text mining [25], which has given rise to further research and there is also one platform that makes use of the myGrid building blocks for the purposes of NLP, which is the subject of Section 6.2.2.

6.2.2. PANACEA

In PANACEA[25] [34], a FP7 project that ran 2010–2012, the objective was to automate the stages involved in the acquisition, production, updating and maintenance of language resources required by machine translation systems, and by other applications for processing of natural language.

The architecture of this factory is based on deploying NLP tools as Web Services using SoapLab to generate them for command-line software, this being the standard mode of invocation for most current NLP tools. These individual services can then be combined in the Taverna Workbench and deployed on a Taverna Server.

In the scope of PANACEA various enhancements have been made to the underlying technology, e.g., the possibility to limit in SoapLab the amount of direct data that SOAP messages can transfer; various bugs were also identified and reported to the developers. A common interchange format for the language resources (esp. annotated corpora) was also defined [33], in the first instance XCES [17], because that was previously used by most of the project partners, but in the final version the format moved to the more current Linguistic Annotation Format (LAF) and Graph-based Format for Linguistic Annotations (GrAF) developed in the scope of the ISO/TC37/SC4 technical committee [19]. Finally, the IPR and other legal issues connected to sharing possibly proprietary tools and esp. resources were also considered in PANACEA [1].

The concrete results of the project are made available via ELDA,[26] the European Evaluations and Language resources Distribution Agency, and consist of:

- the PANACEA Registry,[27] currently describing 163 services (not all freely available);
- the PANACEA MyExperiment[28] installation, which allows exploring workflows, but allows executing them only after registration.

[25] http://panacea-lr.eu/.
[26] http://www.elda.org/
[27] http://registry.elda.org/.
[28] http://myexperiment.elda.org/.

The actual web services mostly run on machines (SoapLab or Taverna servers) of the project partners. It should be noted that the ELDA installation is made with an eye to commercializing the platform.

While the PANACEA platform is in place and operational, there does not seem to be any great up-take of this service by the NLP community. An inspection of the PANACEA MyExperiment shows that the platform receives few new workflows or web services, and most of those fairly specific ones by the (ex)project partners.

6.2.3. ARGO

Tsujii lab at the University of Tokyo had a long tradition in combining NLP with biomedical informatics. For example, they were the creators of the GENIA corpus [26], the first and best known biomedical corpus annotated with linguistic information. In connection with their work on NLP for bioinformatics a workflow for text mining U-compare[29] [24], which includes NLP annotation services was developed. U-compare is implemented as an independent Java application, using the Apache UIMA[30] framework. This work was later integrated into Taverna, in the already mentioned work by [25].

Recently, a new workflow management system has been developed on the basis of U-compare, now in the context of the National Centre for Text Mining (NaCTeM) at the University of Manchester, called ARGO[31] [37,38]. ARGO offers the usual array of features, accessed through a browser based interface: the user can upload documents, compose workflows and execute them. A novelty introduced by ARGO is that workflows can have interactive components as well, where the execution of the workflow pauses to receive input from the user. This is useful for workflows which allow for manual annotation of corpora by the user, and ARGO offers several such workflows. However, ARGO does not seem to have any sophisticated utilities for cataloguing available web services or workflows, nor a system of access permissions.

As far as its architecture goes, ARGO continues to be based on UIMA, and uses REST for communication between the components, so other services or workflows can call ARGO defined web services as well. It supports import and export (deserializations and serializations) into RDF, which is the de-facto language of the Semantic Web. The requirement that web services need compatible I/O formats is in ARGO resolved with a powerful system based on cascaded finite-state transducers called Type Mapper [39], which allows for developing needed shim services.

6.2.4. WebLicht

The European Research Infrastructure CLARIN[32] aims to provide the researchers unified single sign-on access to a platform which integrates language-based resources and advanced tools. This is to be implemented by the construction and operation of a shared distributed infrastructure that aims at making language resources, technology and expertise available to the humanities and social sciences research communities. CLARIN is a distributed data infrastructure, with national sites in a number of European countries. These sites provide access to language data repositories, to services and workflows to work with language data, and to expertise that enables researchers to use these resources and tools.

In the scope of national CLARIN portals, various workflows have been developed, with the German WebLicht[33] [15] being the most advanced. While WebLicht is, in some respects, similar to PANACEA, the focus here is not on providing web services for human language technology research, but rather producing annotated corpora for use in linguistic research. Nevertheless, the methods are similar, as on both platforms most web services are devoted to the linguistic annotation of textual input.

WebLicht does not, as does PANACEA, use the myGrid tools and platforms but rather developed its own infrastructure, starting from the wrappers to make the included tools into RESTful web services, to its centralized repository, and the Web editor enabling the composition of the workflows (or toolchains, as they are known in WebLicht, as they are typically limited to one input and one output). As opposed to PANACEA, WebLicht does cover all the standard linguistic annotation steps, for a number of languages, and with often several tools being available for an annotation step. The WebLicht repository includes information about the type of inputs and outputs of individual services, which allows controlling the workflow construction process in the editor to allow connecting only tools that have matching I/O specifications. For example, a part-of-speech tagger needs a tokenizer to be applied to a text before it can be called. As in PANACEA, WebLicht also imposes standards on the I/O interchange format: it uses TCF (Text Corpus Format) [13], a format similar but slimmer than ISO LAF/GrAF, but also provides conversion to LAF/GrAF. With TCF the text and annotations are in one XML file, but with stand-off annotation, with each processing step adding a layer of annotation.

The Web (2.0) based WebLicht editor allows the construction of workflows and their invocation (after a CLARIN(-DE) recognized log-in) and viewing or saving the results. WebLicht has a dedicated viewer, which allows displaying an annotated corpus in tabular format (for tokens and their annotations), lists (for sentences) or as a graphic (for parse trees).

While the WebLicht platform is not open source, the initiative is open to adding new partners who are interested in contributing tools and services. This typically involves wrapping the tools to be contributed to make them RESTful and to take care of I/O requirements, installing this web service on a local machine and then registering them to the WebLicht

[29] http://u-compare.org/.
[30] https://uima.apache.org/.
[31] http://argo.nactem.ac.uk/
[32] http://www.clarin.eu/.
[33] weblicht.sfs.uni-tuebingen.de/.

central repository. Currently, such third-party web services are provided for Finnish, by the University of Helsinki, and Romanian, by their Academy of Sciences.

6.2.5. Language Grid

The Language Grid[34] [20] is a multilingual Internet service platform, developed by Language Grid Project (started in 2006) at the Japanese National Institute of Information and Communications Technology. The Language Grid is based on the service-oriented architecture (SOA), a web-oriented version of the pipeline architecture typically employed by NLP tools. As other Workflow Management Systems it provides three main functions: language service registration and deployment, language service search, and language service composition and execution. Importantly, Language Grid also offers access to a large number of language resources such as online dictionaries and bi-lingual corpora.

In contrast to e.g., myGrid, geared towards running and reproducing scientific experiments, the Language Grid is much more application oriented, with a focus on enabling communication in multilingual communities (via machine translation), the best known example being the support of farming in rural communities in Vietnam, by enabling computer mediated communication between Vietnamese youth and Japanese experts in agriculture. This is also reflected in its architecture, where the users, services and workflows (or "composite services") are centrally administered. And while running such web services is easy with the provided graphical interface, constructing them is more complicated: workflows are composed using WS-BPEL (Web Service Business Process Execution Language) as XML files, rather than in a dedicated Web based or locally installed visual editor. Although Eclipse does provide a visual BPEL editor, which can be used for this process, workflow construction is still more complicated than with e.g., Taverna.

The Language Grid is Open source and the first European node, known as Linguagrid[35] [4] was established in Italy in 2010.

6.2.6. LAPPS Grid

The Language Grid Server also serves as the basis for the U.S. Language Application (LAPPS) Grid project[36] [18]. LAPPS is especially interesting in three aspects. First, it uses advanced standards for data encoding and exchange, in particular the JSON-based serialization for Linked Data (JSON-LD). The JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format that defines a small set of formatting rules for the portable representation of structured data. Because it is based on the W3C Resource Definition Framework (RDF), JSON-LD is simple to map to and from other graph-based formats, in particular the already mentioned ISO LAF/GrAF [19]. It also enables services to reference categories and definitions in web-based repositories and ontologies, such as those of ISOcat.[37]

Second, it uses the Open Advancement approach (developed in the making of IBM's Watson [10]) for component- and application-based evaluation, that has been successful in enabling rapid identification of frequent error categories within modules and documents, thus contributing to more effective investment of resources in both research and application assembly. The LAPPS Grid thus envisions scenarios where it is possible for users to rapidly (re)configure and automatically evaluate a (changed) pipeline on a chosen dataset and metrics.

Third, workflows are planned to incorporate a comprehensive model for addressing constraints on the intellectual property used in the LAPPS Grid, making it maximally open to users ranging from open source developers to commercial users of language services.

6.3. Comparison with TextFlows

The previous sections presented the more widely used Workflow Management Systems, with a focus on those that are also or primarily used for NLP and that support distributed and/or remote processing. So, for example, we do not treat Moa[38] (not to be confused with MOA,[39] an open source framework for data stream mining) and similar systems that are meant to develop workflows on a local server. In this section we compare the overviewed systems, together with TextFlows along several dimensions that affect the usefulness of each system. In the following section we first present an overview table and then discuss the salient dimension.

6.3.1. Open source

The first dimension, summarized in Table 3, concerns the question whether the Workflow Management Systems is open source, i.e., whether it is possible to download the complete system and install it on local server(s). This is important in cases where the system is to be an internal one, not accessible to third parties, e.g., for data privacy protection or where local modifications to the system are desired. In the general case, this option is not really needed, as it is much easier to simply use the official system. Nevertheless, it is desirable to at least have this option available. Of the surveyed systems, Taverna

[34] http://langrid.org/.
[35] http://www.linguagrid.org/.
[36] http://lapps.anc.org/.
[37] http://www.isocat.org/
[38] http://moa.readthedocs.org/.
[39] http://moa.cms.waikato.ac.nz/

Table 3
Comparison of NLP platforms regarding open source.

Taverna	Yes (Java)
PANACEA	No
ARGO	No
WebLicht	No
Language Grid	Yes (Java, PostgreSQL, Tomcat)
TextFlows	Yes (Python)

Table 4
Comparison of NLP platforms regarding workflow sharing.

Taverna	With myExperiment, BioCatalogue, etc.
PANACEA	Own installation of BioCatalogue and MyExperiment
ARGO	Local registry
WebLicht	Local registry
Language Grid	Local registry
TextFlows	Local registry

Table 5
Comparison of NLP platforms regarding simplicity of use.

Taverna	Difficult
PANACEA	Difficult
ARGO	Easy for workflow composition
WebLicht	Easy for workflow composition
Language Grid	Difficult
TextFlows	Easy for workflow composition

is available under OS, and can be installed on all major platforms (Windows, Mac, Linux) with precompiled distributions or in source Java and has few prerequisites; Language Grid (available from SourceForge) also runs on all platforms and requires PostgreSQL and Tomcat, with the set-up being rather complicated. TextFlows is also open source and publicly available with all the prerequisites wrapped in the installation script.

6.3.2. Workflow sharing

All systems offer workflow sharing, as this is the essence of building such systems, however, they differ in whether the platform itself provides the registry and exploration service of registries or whether they make use of associated services for workflow sharing and discovery. As can be seen from Table 4, most systems provide their own sharing platform, with the exception of Taverna and PANACEA, both of which use myExperiment as a social network platform and BioCatalogue (and, in the case of Taverna, other catalogues as well) as a registry for public web services.

6.3.3. Simplicity of use

This dimension, summarized in Table 5 describes how difficult it is to start using a particular system, not so much from the point of view of executing ready-made workflows but of designing workflows and adding new web services to the platforms. Taverna provides a dedicated Workflow Management System, which is, however, mostly due to the variety of options, reportedly quite difficult to master, also because the details of web service communication have to be understood by the user and complex types decomposed into atomic parts. It is also possible to add new web services using SoapLab. The composition of workflows in PANACEA follows that of Taverna, so the same (dis)advantages apply. ARGO provides a Web-based interface, which allows for graphical composition of workflows. It also provides a sophisticated system for adding new web services, including a testing environment. WebLicht supports web-based workflow composition, with the registry constraining which web services can be attached to the workflow, depending on their I/O requirements; however, new web services cannot be added by users. Language Grid does enable composition of new workflows, but this is an off-line and rather complicated process. Finally, TextFlows offers, via a web interface, easy composition of workflows. Adding new web SOAP-based services with WSDL available is also trivial (based on the user-supplied URL of WSDL, new workflow components are created automatically from service's functions).

6.3.4. I/O protocols

This dimension, summarized in Table 6, concerns input/output protocols. The options supported by the communication protocols between the web services in a workflow have mostly to do with the age of the system: the older ones typically prefer WSDL and SOAP, while the newer ones choose the simpler REST.[40]

[40] Note that the WSDL protocol referred to here is version 1.0 or 1.1; WSDL 2.0 offers (limited) support for RESTful web services.

Table 6
Comparison of NLP platforms regarding I/O protocols.

Taverna	WSDL+SOAP, REST
ARGO	REST
PANACEA	WSDL+SOAP
WebLicht	REST
Language Grid	WSDL+SOAP
TextFlows	WSDL+SOAP, REST, JSON-WSP

When dealing with NLP workflows, standards become very important. NLP components in the main perform pipeline processing, where each tool/service adds another layer of annotation to the base text where it typically needs to have access to previous annotations. Furthermore, the input text might itself contain annotations, such as document structure. The wish for interoperability lead to the development of standards for text (corpus) annotation; however there exist a number of such standards and best practices, and different systems use different ones. TAVERNA, not being targeted toward NLP, is standard agnostic, and relies on implementers of workflows to provide the necessary conversion (shiv) services, i.e., conversion routines to the format that the downstream service(s) expect. In the context of NLP it is also possible to develop shiv services that convert not only outputs but also inputs, taking one of the commonly accepted standards as the pivot encoding. This is the route taken by PANACEA, where each web service is wrapped to accept and produce an output: for primary data this is an XCES encoded file, while text annotations use the LAF/GrAF standard with stand-off annotations. ARGO is based on UIMA and does not enforce any NLP standards in its pipelines. However, ARGO did pioneer a new method of aligning I/O requirements of component web services: rather than relying on shiv services, which need programming skills, it supports a Type Mapper, i.e., a dedicated rule-based analytic for transcribing feature structures between types needed for particular web services. It also supports export and type mapping to RDF.

WebLicht uses a local format TCF, which is, however, quite similar to ISO LAF/GrAF and conversion to these is provided. In WebLicht all web services are expected to use TCF, where the conversion is typically implemented as a wrapper around each annotation tool. Furthermore, the WebLicht Registry included information about the prerequisites for each web service and allows only chaining web services that do not violate these constraints. Language Grid has a very different scheme from most other services, and the details of how the interfaces are to be configured are rather hard to come by. In general, it defines an ontology of Web services, which then specifies also their I/O formats. As the focus is on dictionaries and machine translation, it is these kinds of formats that are defined. Finally, TextFlows, as Taverna, does not impose any standards in its I/O formats. Mostly plain text, JSON, XML and, for efficiency, serialized Python data structures and objects (in particular, the *AnnotatedDocumentCorpus* instances) are exchanged between workflow components.

7. Conclusions and directions for future work

This paper presented TextFlows, an open source platform featuring workflow components for text mining and natural language processing. TextFlows provides a common graphical user interface and joins several text mining, visualization and machine learning libraries under a single unifying platform, expanding their usability for researchers, practitioners and non-experts. The architecture of the platform is scalable, allows for many users and workflow and dataset sharing.

The usability of the platform was demonstrated on three illustrative use cases, showing that components developed for different systems and in different programming languages can be run within a single coherent system and may be represented as a visually constructed workflow available on the web. The ease of sharing the completed workflows and results over the web allows for TextFlows to become a central integration platform for many text mining and natural language processing tasks. We have compared our work to related platforms and shown the differences, benefits, and drawback of using TextFlows instead of other text mining platforms.

There are several directions for future work. We plan to expand the TextFlows widget repository with additional text preprocessing components (such as chunking, term extraction, syntactic parsing), extend the batch of weighting schemes for generating the BoW models, and include various algorithms for clustering. We also plan to extend the literature-based discovery package to include more widgets for cross-domain bridging term discovery. Furthermore, we will connect the platform to various external tools to better assist the user in the process of exploration and visualization of results, such as TopicCircle [22] for cluster visualization.

Second, by using public data on user workflows, submitted to the public version of the TextFlows platform, we will construct a recommender system based on the data on previously executed workflows that will enable computer-assisted construction of text mining workflows and bring the platform even closer to non-experts in terms of usability.

A current weakness of TextFlows is the inability to deal with very large data. As part of future work we plan to integrate the stream mining and big data mining features of ClowdFlows to the TextFlows platform. This will allow performing natural language processing on a larger scale, as well as preforming different tasks on streams of data such as website news feeds, or real time data from social networks such as Twitter and Facebook.

Finally, we wish to simplify the installation procedures of TextFlows to private servers by providing one-click deployment to services such as Amazon Elastic Compute Cloud (EC2) and Microsoft Azure.

We have released the sources of the TextFlows platform under an open source MIT license, available at https://github.com/xflows/textflows. Detailed deployment instructions are provided with the source code. A public installation of the platform is available for online use at http://textflows.org.

Acknowledgements

We are grateful to Borut Sluban and Senja Pollak for their previous work on the Kenyan elections document corpus. This work was supported by the Slovenian Research Agency and the FP7 European Commission FET projects MUSE (grant no. 296703) and ConCreTe (grant no. 611733).

References

[1] V. Arranz, O. Hamon, On the way to a legal sharing of web applications in NLP, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 2965–2970.
[2] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, C.A. Goble, BioCatalogue: a universal catalogue of web services for the life sciences, Nucleic Acids Res. 38 (Web-Server-Issue) (2010) 689–694.
[3] S. Bird, NLTK: the natural language toolkit, in: Proceedings of the COLING/ACL on Interactive Presentation Sessions, Association for Computational Linguistics, 2006, pp. 69–72.
[4] A. Bosca, L. Dini, M. Kouylekov, M. Trevisan, Linguagrid: a network of linguistic and semantic services for the Italian language, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 3304–3307.
[5] E. Brill, A simple rule-based part of speech tagger, in: Proceedings of the Workshop on Speech and Natural Language, Association for Computational Linguistics, 1992, pp. 112–116.
[6] C.E. Brodley, M.A. Friedl, Identifying mislabeled training data, J. Artif. Intell. Res. 11 (1999) 131–167.
[7] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth, CRISP-DM 1.0 step-by-step data mining guide, http://www.crisp-dm.org/CRISPWP-0800.pdf, 2000.
[8] J. Demšar, B. Zupan, G. Leban, T. Curk, Orange: from experimental machine learning to interactive data mining, in: Proceedings of Knowledge Discovery in Databases, PKDD, 2004, pp. 537–539.
[9] R. Feldman, J. Sanger, The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data, Cambridge University Press, 2007.
[10] D.A. Ferrucci, E.W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J.M. Prager, N. Schlaefer, C.A. Welty, Building Watson: an overview of the DeepQA project, AI Mag. 31 (3) (2010) 59–79.
[11] M. Grčar, Mining text-enriched heterogeneous information networks, Ph.D. thesis, Jožef Stefan International Postgraduate School, 2015.
[12] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, USA, 2006.
[13] U. Heid, H. Schmid, K. Eckart, E. Hinrichs, A corpus representation format for linguistic web services: the D-SPIN text corpus format and its relationship with ISO standards, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation, LREC'10, Valletta, Malta, European Language Resources Association (ELRA), 2010, pp. 494–499.
[14] D. Hiemstra, A probabilistic justification for using TF-IDF term weighting in information retrieval, Int. J. Digit. Libr. 3 (2) (2000) 131–139.
[15] M. Hinrichs, T. Zastrow, E. Hinrichs, WebLicht: web-based LRT services in a distributed eScience infrastructure, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation, LREC'10, Valletta, Malta, European Language Resources Association (ELRA), 2010.
[16] D. Hull, K. Wolstencroft, R. Stevens, C.A. Goble, M.R. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Res. 34 (Web-Server-Issue) (2006) 729–732.
[17] N. Ide, P. Bonhomme, L. Romary, XCES: an XML-based encoding standard for linguistic corpora, in: Proceedings of the Second International Language Resources and Evaluation Conference, Paris, European Language Resources Association, 2000, pp. 825–830.
[18] N. Ide, J. Pustejovsky, C. Cieri, E. Nyberg, D. Wang, K. Suderman, M. Verhagen, J. Wright, The language application grid, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC'14, Reykjavik, Iceland, European Language Resources Association (ELRA), 2014, pp. 22–30.
[19] N. Ide, K. Suderman, GrAF: a graph-based format for linguistic annotations, in: Proceedings of the Linguistic Annotation Workshop, LAW '07, Stroudsburg, PA, USA, Association for Computational Linguistics, 2007, pp. 1–8.
[20] T. Ishida, The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability, Springer Science & Business Media, 2011.
[21] M. Juršič, B. Cestnik, T. Urbančič, N. Lavrač, Cross-domain literature mining: finding bridging concepts with CrossBee, in: Proceedings of the 3rd International Conference on Computational Creativity, 2012, pp. 33–40.
[22] M. Juršič, B. Cestnik, T. Urbančič, N. Lavrač, HCI empowered literature mining for cross-domain knowledge discovery, in: Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data, Springer, 2013, pp. 124–135.
[23] M. Juršič, I. Mozetič, T. Erjavec, N. Lavrač, Lemmagen: multilingual lemmatisation with induced ripple-down rules, J. Univers. Comput. Sci. 16 (9) (2010) 1190–1214.
[24] Y. Kano, W. Baumgartner, L. McCrohon, S. Ananiadou, K. Cohen, L. Hunter, J. Tsujii, U-Compare: share and compare text mining tools with UIMA, Bioinformatics 25 (15) (2009) 1997–1998.
[25] Y. Kano, P. Dobson, M. Nakanishi, J. Tsujii, S. Ananiadou, Text mining meets workflow: linking U-Compare with Taverna, Bioinformatics 26 (19) (2010) 2486–2487.
[26] J.-D. Kim, T. Ohta, Y. Tateisi, J. Tsujii, GENIA corpus – a semantically annotated corpus for bio-textmining, in: ISMB (Supplement of Bioinformatics), 2003, pp. 180–182.
[27] C. Kohlschütter, P. Fankhauser, W. Nejdl, Boilerplate detection using shallow text features, in: Proceedings of the Third ACM International Conference on Web Search and Data Mining, ACM, 2010, pp. 441–450.
[28] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows: a cloud based scientific workflow platform, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), Proceedings of Machine Learning and Knowledge Discovery in Databases, ECML/PKDD (2), Springer, 2012, pp. 816–819.
[29] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler, Yale: rapid prototyping for complex data mining tasks, in: L. Ungar, M. Craven, D. Gunopulos, T. Eliassi-Rad (Eds.), Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, 2006, pp. 935–940.
[30] A.Y. Ng, Feature selection, L1 vs. L2 regularization, and rotational invariance, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 78.
[31] G. Paltoglou, M. Thelwall, A study of information retrieval weighting schemes for sentiment analysis, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2010, pp. 1386–1395.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[33] M. Poch, N. Bel, Interoperability and technology for a language resources factory, in: Proceedings of the Workshop on Language Resources, Technology and Services in the Sharing Paradigm, Chiang Mai, Thailand, Asian Federation of Natural Language Processing, 2011, pp. 32–40.
[34] M. Poch, A. Toral, O. Hamon, V. Quochi, N. Bel, Towards a user-friendly platform for building language resources based on web services, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 1156–1163.
[35] S. Pollak, Text classification of articles on Kenyan elections, in: Proceedings of the 4th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics, 2009, pp. 229–233.
[36] S. Pollak, R. Coesemans, W. Daelemans, N. Lavrač, Detecting contrast patterns in newspaper articles by combining discourse analysis and text mining, Pragmatics 21 (4) (2013) 674–683.
[37] R. Rak, A. Rowley, W. Black, S. Ananiadou, Argo: an integrative, interactive, text mining-based workbench supporting curation, Database, J. Biol. Databases Curation (2012), http://database.oxfordjournals.org/content/2012/bas010.full.
[38] R. Rak, A. Rowley, J. Carter, S. Ananiadou, Development and analysis of NLP pipelines in Argo, in: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Association for Computational Linguistics, 2013, pp. 115–120.
[39] R. Rak, A. Rowley, J. Carter, R.T.B. Batista-Navarro, S. Ananiadou, Interoperability and customisation of annotation schemata in Argo, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC'14, European Language Resources Association (ELRA), 2014, pp. 3837–3842.
[40] P. Refaeilzadeh, L. Tang, H. Liu, Cross-validation, in: Encyclopedia of Database Systems, Springer, 2009, pp. 532–538.
[41] D.D. Roure, C. Goble, R. Stevens, The design and realisation of the myExperiment virtual research environment for social sharing of workflows, Future Gener. Comput. Syst. 25 (2009) 561–567.
[42] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, Inf. Process. Manag. 24 (5) (1988) 513–523.
[43] S. Scott, S. Matwin, Feature engineering for text classification, in: Proceedings of 16th International Conference on Machine Learning, ICML'99, 1999, pp. 379–388.
[44] M. Senger, P. Rice, T. Oinn, Soaplab: a unified sesame door to analysis tools, in: UK e-Science All Hands Meeting, National e-Science Centre, 2003, pp. 509–513.
[45] B. Sluban, D. Gamberger, N. Lavrač, Ensemble-based noise detection: noise ranking and visual performance evaluation, Data Min. Knowl. Discov. 28 (2) (2014) 265–303.
[46] B. Sluban, N. Lavrač, ViperCharts: visual performance evaluation platform, in: Proceedings of Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 650–653.
[47] B. Sluban, S. Pollak, R. Coesemans, N. Lavrač, Irregularity detection in categorized document corpora, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, 2012, pp. 1598–1603.
[48] N. Smalheiser, D. Swanson, Using ARROWSMITH: a computer-assisted approach to formulating and assessing scientific hypotheses, Comput. Methods Programs Biomed. 57 (3) (1998) 149–154.
[49] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed., Morgan Kaufmann, Amsterdam, 2011.

Chapter 4

Use Case: Wordification in ClowdFlows

This chapter presents a use case scenario where ClowdFlows is used as a test bed and experimental environment to validate a novel propositionalization methodology entitled *wordification*.

The first section of the chapter presents the problem of relational data mining and introduces the *wordification* methodology. This is followed by a description of ClowdFlows features that allow for experimentation in a workflow environment. Finally, the core of the chapter is the journal publication which presents the *wordification* approach in detail along with its validation and comparison with other Inductive Logic Programming (ILP) and Relational Data Mining (RDM) algorithms, which have been implemented as executable ClowdFlows workflows.

4.1 Problem Definition and Motivation

Standard machine learning and data mining algorithms look for patterns and induce models from a single data table, where one example corresponds to a row in the table. Inductive Logic Programming [66] and Relational Data Mining [67] algorithms and approaches induce hypotheses from multiple tables, e.g., data stored in relational databases.

For multi-relational databases in which data instances are clearly identifiable (characterized by one-to-many relationships among the target table and other data tables), various techniques can be used for transforming a multi-relational database into a propositional single-table format [68]. After performing such a transformation, entitled *propositionalization* [69], standard propositional learners can be used, such as decision trees and classification rule learners.

Wordification is a new propositionalization approach inspired by text mining [70] and can be seen as a transformation of a relational database into a corpus of documents, where each document can be characerized by a set of properties describing the entries of a relational database. Documents are represented as Bag-Of-Words (BOW) vectors of weights and features. The words are constructed from individual attribute-values of the target table and related tables. Unlike other propositionalization techniques, which search for good (and possibly complex) relational features to describe the subsequent propositional representation, this methodology focuses on a large number of simple features with the aim of greater scalability. Since the feature construction step is efficient, it can work well for large relational databases. In fact, the newly developed methodology transforms a given relational database in time linear to the number of attributes times the number of examples for one-to-many databases. Furthermore, due to the simplicity of features, the generated features are easily interpretable by domain experts. On the other hand, this methodology suffers from the loss of information, since the generated features do not explicitly connect relations through variables. Instead, by using the Term Frequency Inverse Document Frequency weighting (TF-IDF) [71], it tries to capture the importance of a certain feature (attribute value) of a relation in an aggregate manner.

In summary, the input to the wordification approach is a relational database, and the output is a set of feature vectors, which can be viewed as a corpus of text documents represented in the Bag-Of-Words vector format. Each text document represents an individual entry of the main data table. A document is described by a set of words, where a word is constructed as a combination of the table name, name of the attribute and its discrete value. After the documents are transformed into Bag-Of-Words representation, the class attribute column is appended to the transformed table.

The aim of the use case is to show that the developed wordification approach is simple, more efficient and at least as accurate as the comparable state-of-the-art propositionalization methods such as the RSD algorithm [72] and RelF [73], and comparable to methods that directly induce relational patterns or models without propositionalization, such as Aleph [74]. To achieve this, two experimental workflows were developed in ClowdFlows. One workflow was developed for learning and another for evaluating and visualizing the results. The implemented workflows allow methodology reuse and repeatability of experiments.

4.2 Experimentation and Evaluation in the ClowdFlows Platform

To evaluate and validate the newly developed propositionalisation methodology it is necessary to perform experiments on different data sets and compare different methodologies. To faciliate this, we have implemented mechanisms to perform multi-fold cross-validation and visually compare results.

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set [75]. It is an iterative process where the data are partitioned into a training and test set multiple times. The model is trained on the training data and executed on the test data and performance measures are calculated. Finally, the performance measures (classification accuracy, precision, recall) of the test are merged over several iterations (folds) to derive a more accurate estimate of model prediction performance.

As cross-validation is an iterative process, it is not intuitively translated into a general graphical workflow. A workflow in ClowdFlows is defined as a set of workflow components (widgets) and a set of connections. In order to implement a process similar to cross-validation as a graphical workflow, it is required to encapsulate workflows or parts of workflows in widgets that can be replicated and reused. To this end, we developed the sub-workflow widget, which can contain entire workflows. The inputs of the container widget are implicitly connected to the outputs of special widgets inside the sub-workflow (which can be added or removed at will, so that their number corresponds to the number of desired inputs of the container widget). The outputs of the container widget are implicitly connected to the inputs of special widgets inside the sub-workflow. In short, the simplest sub-workflow widget contains a workflow that consists of two widgets, the *input* and the *output* widget, whose output and input are implicitly connected to the input and output of the container widget.

Sub-workflows are useful for encapsulating parts of workflows for re-use, however using only sub-workflows it is not possible to implement a general cross-validation process. As cross-validation is an iterative process, we augmented the sub-workflow mechanism with an iteration. If a sub-workflow contains a special pair of widgets (the *for loop input* and the *for loop output*), the sub-workflow will be executed multiple times, once for each instance of an iterable object (e.g. a data set table) on the input. The iterable object is constructed from the result of each iteration on the output. To simplify implementation of cross-validation, we added an iterative sub-workflow that randomly splits the iterable object into two sets, a predefined number of times. To ensure reproducibility of experiments, the number of folds and the seed of the random generator are part of the workflow.

To increase the performance of cross-validation, ClowdFlows leverages it with its cloud-based nature. ClowdFlows can be installed on several machines and use them as processing units (or workers). As each iteration in a ClowdFlows loop is independant of previous iterations, it is possible to run the iterations in parallel. Iterations are uniformly distributed across workers as separate tasks. Once all the tasks are finished, the results are combined into another iterable object. In the case of cross-validation, these results are performance measurements for iterations that need to be combined.

The averaged performance measurements can then be displayed or rendered visually. A visual rendering of the performance measurements is implemented via the interactive VIPER (Visual Performance Evaluation) charts [76]. These interactive charts display the results in the precision-recall space. This visualization simplifies the comparison of performance measurements for different methodologies.

Using cross-validation workflow widgets, it is possible to utilize ClowdFlows as a test bed for newly developed methodologies that can be validated using cross-validation and visually compared with others. A general comparison workflow is often used to compare multiple implementations of methodologies and processes. The individual implementations can be simply replaced with other ones by replacing a single workflow element. This workflow was successfully employed in the validation of the newly developed *wordification* propositionalization methodology.

4.3 Related Publication

The methodology and its implementation in ClowdFlows is described in detail in the following journal publication:

M. Perovšek, A. Vavpetič, J. Kranjc, B. Cestnik, and N. Lavrač, "Wordification: Propositionalization by unfolding relational data into bags of words," *Expert Systems with Applications*, vol. 42, no. 17, pp. 6442–6456, 2015.

In this publication we achieve the following:

- We present and improve the wordification methodology and provide a formal framework for it and its pseudo code.
- We statistically evaluate the compared algorithms on multiple relational databases.
- We perform experiments which show favorable results of wordification in terms of accuracy and efficiency.
- We prove that feature simplicity can be compensated by n-gram construction and feature weighting.
- We implement the full experimental and validation workflow in the data mining platform ClowdFlows.

The authors' contributions are as follows. Matic Perovšek implemented the wordification widgets in ClowdFlows. Anže Vavpetič contributed widgets for connection to relational databases and widgets for other methodologies used to validate wordification. The testing environment and the cross-validation techniques used for validating the experiments in ClowdFlows were implemented by Janez Kranjc. Nada Lavrač provided the main idea for the wordification methodology and together with Bojan Cestnik served as supervisor. All authors contributed to the text of the publication.

Expert Systems with Applications 42 (2015) 6442–6456

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Wordification: Propositionalization by unfolding relational data into bags of words

Matic Perovšek [a,b,*], Anže Vavpetič [a,b], Janez Kranjc [a,b], Bojan Cestnik [a,c], Nada Lavrač [a,b,d]

[a] Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[c] Temida d.o.o, Ljubljana, Slovenia
[d] University of Nova Gorica, Nova Gorica, Slovenia

A R T I C L E I N F O

Article history:
Available online 24 April 2015

Keywords:
Wordification
Inductive Logic Programming
Relational Data Mining
Propositionalization
Text mining
Classification

A B S T R A C T

Inductive Logic Programming (ILP) and Relational Data Mining (RDM) address the task of inducing models or patterns from multi-relational data. One of the established approaches to RDM is propositionalization, characterized by transforming a relational database into a single-table representation. This paper presents a propositionalization technique called *wordification* which can be seen as a transformation of a relational database into a corpus of text documents. Wordification constructs simple, easy to understand features, acting as words in the transformed Bag-Of-Words representation. This paper presents the wordification methodology, together with an experimental comparison of several propositionalization approaches on seven relational datasets. The main advantages of the approach are: simple implementation, accuracy comparable to competitive methods, and greater scalability, as it performs several times faster on all experimental databases. Furthermore, the wordification methodology and the evaluation procedure are implemented as executable workflows in the web-based data mining platform ClowdFlows. The implemented workflows include also several other ILP and RDM algorithms, as well as the utility components that were added to the platform to enable access to these techniques to a wider research audience.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

Standard propositional data mining algorithms, included in established data mining tools like Weka (Witten, Frank, & Hall, 2011), induce models or patterns learned from a single data table. On the other hand, the aim of Inductive Logic Programming (ILP) and Relational Data Mining (RDM) is to induce models or patterns from multi-relational data (De Raedt, 2008; Džeroski & Lavrač, 2001; Lavrač & Džeroski, 1994; Muggleton, 1992). Most types of propositional models and patterns have corresponding relational counterparts, such as relational classification rules, relational regression trees or relational association rules.

For multi-relational databases in which data instances are clearly identifiable (the so-called individual-centered representation (Flach & Lachiche, 1999), characterized by one-to-many relationships among the target table and other data tables), various techniques can be used for transforming a multi-relational database into a propositional single-table format (Krogel et al., 2003). After performing such a transformation (Lavrač, Džeroski, & Grobelnik, 1991), named *propositionalization* (Kramer, Pfahringer, & Helma, 1998), standard propositional learners can be used, including decision tree and classification rule learners.

Inspired by text mining, this paper presents a propositionalization approach to Relational Data Mining, called *wordification*. Unlike other propositionalization techniques (Kramer et al., 1998; Kuželka & Železný, 2011; Lavrač et al., 1991; Železný & Lavrač, 2006), which first construct complex relational features (constructed as a chain of joins of one or more tables related to the target table), used as attributes in the resulting tabular data representation, wordification generates much simpler features with the aim of achieving greater scalability.

Wordification can be viewed as a transformation of a relational database into a set of feature vectors, where each original instance is transformed into a-kind-of 'document' represented as a Bag-Of-Words (BOW) vector of weights of simple features, which can be interpreted as 'words' in the transformed BOW space. The 'words' constructed by wordification correspond to individual

* Corresponding author at: Jožef Stefan Institute, Ljubljana, Slovenia.
E-mail addresses: matic.perovsek@ijs.si (M. Perovšek), anze.vavpetic@ijs.si (A. Vavpetič), janez.kranjc@ijs.si (J. Kranjc), bojan.cestnik@temida.si (B. Cestnik), nada.lavrac@ijs.si (N. Lavrač).

http://dx.doi.org/10.1016/j.eswa.2015.04.017
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

attribute–values of the target table and of the related tables, subsequently weighted by their Term Frequency-Inverse Document Frequency (TF-IDF) value (Jones, 1972; Salton & Buckley, 1988) (requiring real-valued attributes to be discretized first). Alternatively, instead of TF-IDF, simpler schemes can be used such as term frequency (TF) 'word' count, or the binary scheme indicating just the presence/absence of a 'word' in the 'document'.

To intuitively phrase the main idea of wordification, take two simple examples illustrating the wordification data preprocessing step in class-labeled data, where each structured data instance is transformed into a tuple of simple features, which are counts/weights of individual attribute–value pairs. Take the well-known relational domain of East–West Trains (Michie, Muggleton, Page, & Srinivasan, 1994) with cars containing different loads: one of the train's features in the BOW representation is the count/weight of rectangular loads it carries, no matter in which cars these loads are stored. Or in the standard Mutagenesis domain (Srinivasan, Muggleton, King, & Sternberg, 1994), a molecule may prove to be toxic if it contains a lot of atoms characterized by the property *atom_type* = *lead*, no matter how these atoms are bonded in the molecule. The main hypothesis of the wordification approach is that the use of this simple representation bias is suitable for achieving good results in classification tasks. Moreover, when using a binary scheme, this representation bias allows for simple and very intuitive interpretation in descriptive induction tasks, such as association rule learning from unlabeled multi-relational data.

Wordification suffers from some loss of information, compared to propositionalization methods which construct complex first-order features (which get values *true* or *false* for a given individual) as a chain of joins of one or more tables related to the target table. Nevertheless, despite some information loss, wordification has numerous advantages. Due to the simplicity of features, the generated hypotheses are easily interpretable by domain experts. The feature construction step in wordification is very efficient, therefore it can scale well for large relational databases. As wordification constructs each 'document' independently from the other 'documents', a large main table can be divided into smaller batches of examples, which can be propositionalized in parallel. Next, wordification can use TF or TF-IDF word weighting to capture the importance of a given feature (attribute value) of a relation in an aggregate manner, while feature dependence is modeled by constructing a-kind-of word '*n*-grams' as conjuncts of a predefined number of simple features. Finally, the wordification approach has the advantage of using techniques developed in the text mining community, such as efficient document clustering or word cloud visualization, which can now be effectively exploited in multi-relational data mining.

This paper shows that the developed wordification technique is simple, considerably more efficient and at least as accurate as the comparable state-of-the-art propositionalization methods. This paper extends our previous research (Perovšek, Vavpetič, & Lavrač, 2012, 2013) in many ways. The related work is more extensively covered. The improvements to the methodology include feature filtering by frequency, performance optimization (indexing by value), new options regarding feature weighting (next to TF-IDF, we added TF and binary), and a parallel version of the algorithm. The methodology description is now more detailed, including the formal wordification framework, the wordification algorithm pseudo code as well as time and space complexity analysis. The experimental evaluation has been substantially extended to include a comparison of three different term weighting schemes, additional propositionalization algorithms RelF (Kuželka & Železný, 2011) and Aleph (Srinivasan, 2007), as well as an additional classifier (SVM), which were applied to an extended set of experimental relational datasets. Such exhaustive experimentation has enabled us to statistically validate the experimental results by using the Friedman test and the Nemenyi post hoc test on the seven benchmark problems from the five relational domains (two of which have two database variants): IMDB,[1] Carcinogenesis (Srinivasan, King, Muggleton, & Sternberg, 1997), Financial[2] and two variants of Trains (Michie et al., 1994) and Mutagenesis (Srinivasan et al., 1994). Further experiments were done to analyze the effects of feature weighting, pruning and *n*-gram construction. In addition to the two experimental workflows developed in the web-based data mining platform ClowdFlows (Kranjc, Podpečan, & Lavrač, 2012), one workflow developed for learning and another for results evaluation and visualization, this paper introduces another wordification workflow applicable in association rule learning tasks from binarized features. The implemented workflows, which are available online through ClowdFlows, allow for methodology reuse and experiment repeatability. As a side-product of workflow development, the competing propositionalization algorithms used in experimental comparisons are also made available through ClowdFlows and can therefore be easily reused in combination with numerous pre-existing ClowdFlows components for data discretization, learning, visualization and evaluation, including a large number of Weka (Witten et al., 2011) and Orange (Demšar, Zupan, Leban, & Curk, 2004) components. Making selected RDM algorithms handy to use in real-life data analytics may therefore contribute to improved accessibility and popularity of Relational Data Mining.

The paper is organized as follows. Section 2 describes the background and the related work. Section 3 gives an informal overview of the wordification methodology, while Section 4 presents the formalism and the details of the developed wordification algorithm. The implementation of the methodology as a workflow in the ClowdFlows platform is described in Section 5. Section 6 presents the evaluation methodology implementation and the experimental results. Section 7 illustrates the utility of wordification in a descriptive induction setting of learning association rules from two real-life domains, using data from a subset of the IMDB movies database and from a database of traffic accidents. Section 8 concludes the paper by presenting the plans for further work.

2. Background and related work

Inductive Logic Programming (ILP) and Relational Data Mining (RDM) algorithms are characterized by the ability to use background knowledge in learning relational models or patterns (Džeroski & Lavrač, 2001; De Raedt, 2008; Lavrač & Džeroski, 1994; Muggleton, 1992), as by taking into account additional relations among the data objects the performance of data mining algorithms can be significantly improved.

Propositionalization (Kramer et al., 1998; Lavrač et al., 1991) is an approach to ILP and RDM, which offers a way to transform a relational database into a propositional single-table format. In contrast to methods that directly induce relational patterns or models, such as Aleph (Srinivasan, 2007) and Progol (Muggleton, 1995), propositionalization algorithms transform a relational problem into a form which can be solved by standard machine learning or data mining algorithms. Consequently, learning with propositionalization techniques is divided into two self-contained phases: (1) relational data transformation into a single-table data format and (2) selecting and applying a propositional learner on the transformed data table. As an advantage, propositionalization is not limited to specific data mining tasks such as classification, which is usually the case with ILP and RDM methods that directly induce models from relational data.

[1] http://www.webstepbook.com/supplements/databases/imdb.sql.
[2] http://lisp.vse.cz/pkdd99/Challenge/berka.htm.

The transformation to a single-table format can be achieved for the so-called individual-centered relational databases (Flach & Lachiche, 1999), i.e., databases that have a clear notion of an individual. The East–West Trains challenge (Michie et al., 1994), where the task is to classify the Trains as East-bound or West-bound, is a well-known domain in which individuals are clearly identified: each train is a single individual related with one or more cars that have different characteristics.

Most of the related work involves propositionalization through first-order feature construction (Kramer et al., 1998; Kuželka & Železný, 2011; Lavrač et al., 1991; Železný & Lavrač, 2006), where the algorithms construct complex first-order features, which then act as binary attributes in the new propositional representation of examples. One of the first propositionalization algorithms, LINUS (Lavrač et al., 1991), generates features that do not allow recursion and newly introduced variables. An improvement of LINUS, named SINUS (Lavrač & Flach, 2001), incorporates more advanced feature construction techniques inspired by feature construction implemented in 1BC (Flach & Lachiche, 1999). RSD (Železný & Lavrač, 2006) is a relational subgroup discovery algorithm composed of two main steps: the propositionalization step and the subgroup discovery step, where 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 constructed feature can be decomposed into a conjunction of two or more features.

RELAGGS (Krogel & Wrobel, 2001), which stands for *rel*ational *agg*regation, is a propositionalization approach, which uses the input relational database schema as a basis for a declarative bias and aims to use optimization techniques usually used in relational databases (e.g., indexes). Furthermore, the approach employs aggregation functions in order to summarize non-target relations with respect to the individuals in the target table.

An early experimental comparison of propositionalization techniques was reported in Krogel et al. (2003), where RSD, SINUS and RELAGGS algorithms were compared.

Other means of propositionalization include stochastic propositionalization (Kramer et al., 1998), which employs a search strategy similar to random mutation hill-climbing: the algorithm iterates over generations of individuals, which are added and removed with a probability proportional to the fitness of individuals, where the fitness function used is based on the Minimum Description Length (MDL) principle.

Safarii[3] is a commercial multi-relation data mining tool. Safarii, extensively described in Knobbe (2005), offers a unique pattern language that merges ILP-style structural descriptions as well as aggregations. Furthermore, Safarii comes with a tool called ProSafarii, which offers several preprocessing utilities, including propositionalization via aggregation.

Ceci and Appice (2006) investigate spatial classification using two techniques: a propositionalization approach which constructs features using spatial association rules to produce an attribute–value representation. They compare the approach to a structural approach using an extended Naive Bayes classifier. They report an advantage of the structural alternative in terms of accuracy, while the propositional approach performs faster. Ceci, Appice, and Malerba (2008) present two emerging patterns based classifiers that work in the multi-relational setting: one uses a heuristic evaluation function to classify objects, while the other is based on a probabilistic evaluation. The main result of the study is that both approaches perform better than associative classification to which they were compared.

Kuželka and Železný (2011) developed RelF, which constructs a set of tree-like relational features by combining smaller conjunctive blocks. The novelty is that 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.

An approach that is related to propositionalization is presented by Guo and Viktor (2008). The authors propose a strategy of multi-relational learning where they neither upgrade a propositional learner to work with multiple relations or propositionalize the relations. Instead, their approach learns from multiple views (feature sets) of a RDB and then integrates the individual view learners to construct a final model. Their approach exhibits comparable classification accuracies compared to related approaches, and a faster runtime.

Recently, a propositionalization technique called Bottom Clause Propositionalization (BCP) was introduced by França, Zaverucha, and d'Avila Garcez (2014). It was integrated with C-IL^2P (Garcez et al., 1999); the combined system, named CILP++, achieves accuracy comparable to Aleph, while being faster. Compared to RSD, BCP is better in terms of accuracy when using a neural network and similar when using C4.5.

3. Informal description of the wordification approach

This section provides an informal description of the proposed approach, where wordification is illustrated on a simplified variant of the well-known East–West Trains problem (Michie et al., 1994).

The transformation from a relational database representation into a Bag-Of-Words feature vector representation is illustrated in Fig. 1, where the input to wordification is a relational database, and the output is a set of feature vectors, which can be viewed as a corpus of text documents represented in the Bag-Of-Words (BOW) vector format. Each text document represents an individual entry of the main data table. A document is described by a set of words (or features), where a word is constructed as a combination of the table name, name of the attribute and its discrete (or discretized) value[4]:

$$[\mathit{table\ name}]\_[\mathit{attribute\ name}]\_[\mathit{value}]. \qquad (1)$$

Such constructs are called *word-items* or *witems* or simply *words* in the rest of the paper. Note that values of every non-discrete attribute need to be discretized beforehand in order to be able to represent them as word-items. For each individual, the word-items are first generated for the main table and then for each entry from the related tables, and finally joined together according to the relational schema of the database.[5] In the described transformation there is some loss of information as a consequence of building the document for each instance (each individual row in the main table) by concatenating all word-items from multiple instances (rows) of the connected tables into a single document. To overcome this loss, we extended the document construction step of the initial wordification methodology by concatenating to the document also n-grams of word-items, constructed as conjunctions of several word-items. These concatenations of elementary word-items represent conjunctions of features occurring together in individual instances (rows of joined tables). Technically, n-gram construction is performed by taking every combination of length k of word-items from the set of all

[3] http://www.kiminkii.com/safarii.html.

[4] See Line 4 in Algorithm 2 presented in Section 4.3.

[5] See Line 10 in Algorithm 2 presented in Section 4.3.

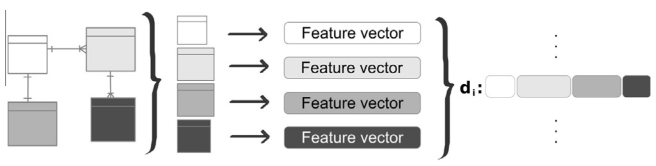


Fig. 1. The transformation from a relational database representation into a Bag-Of-Words feature vector representation. For each individual entry of the main table one Bag-Of-Words (BOW) vector d_i of weights of 'words' is constructed, where 'words' correspond to the features (attribute values) of the main table and the related tables.

word-items corresponding to the given individual, and concatenating them as follows:

$$[witem_1]\_\_[witem_2]\_\_\ldots\_\_[witem_k], \qquad (2)$$

where $1 \leqslant k \leqslant n$ and—as mentioned earlier—each word-item is a combination of the table name, name of the attribute and its discrete value. The witems are concatenated in a predetermined order, each using the "__" concatenation symbol.

In the rest of this section, for simplicity, we refer to individuals as documents, to features as words, and to the resulting representation as the Bag-Of-Words (BOW) representation. For a given word w in document d from corpus D, the TF-IDF measure is defined as follows:

$$\mathrm{tfidf}(w,d) = \mathrm{tf}(w,d) \times \log \frac{|D|}{|\{d \in D : w \in d\}|}, \qquad (3)$$

where $\mathrm{tf}(\cdot)$ represents the number of times word w appears in document d. In other words, a word with a high TF-IDF value will be considered important for the given individual provided that it is frequent within this document and not frequent in the entire corpus. Consequently, the weight of a word provides a strong indication of how relevant is the feature for the given individual. The TF-IDF weights can then be used either for filtering out words with low importance or using them directly by a propositional learner.

In addition to the TF-IDF weigthing scheme, the implementation of wordification (described in detail in Section 5) includes also the term frequency (TF) and the binary (0/1) weighting schemes. A comparison of the three schemes can be found in the Appendix A (see Table A.6). Given that different weighting schemes do not perform significantly differently on the classification tasks used in our experiments, in the rest of the paper we use the TF-IDF scheme since this form of weighting is prevalent in text mining applications.

The wordification approach is illustrated on a modified and substantially simplified version of the well-known East–West Trains domain (Michie et al., 1994), where the input database consists of just two tables shown in Fig. 2, where we have only one East-bound and one West-bound train, each with just two cars with certain properties. Note that in the experimental section we use the standard version of the East–West Trains domain.

The TRAIN table is the main table and the Trains are the individuals. We want to learn a classifier to determine the direction of an unseen train. For this purpose the direction attribute is not preprocessed and is only appended to the resulting feature vector (list of words).

First, the corresponding two documents (one for each train t1 and t5) are generated, as shown in Fig. 3. After this, the documents are transformed into the Bag-Of-Words representation by calculating the TF-IDF values for each word of each document (using Eq. (3)) with the class attribute column appended to the transformed Bag-Of-Words table, as shown in Fig. 4. For simplicity, only unigrams and bigrams are shown in this example.

TRAIN

trainID	eastbound
t1	east
…	…
t5	west
…	…

CAR

carID	shape	roof	wheels	train
c11	rectangle	none	2	t1
c12	rectangle	peaked	3	t1
…	…	…	…	…
c51	rectangle	none	2	t5
c52	hexagon	flat	2	t5
…	…	…	…	…

Fig. 2. Example input for wordification in the East–West Trains domain.

4. Wordification methodology

This section formally describes the wordification methodology by presenting the input data model and input language bias, the relational database representation, followed by the presentation of the pseudo-code and the worst-case complexity analysis of the wordification algorithm.

4.1. Data model

A data model describes the structure of the data. It can be expressed as an entity-relationship (ER) diagram. The ER diagram, illustrated in Fig. 5, shows three relations appearing in the original East–West Trains problem (in addition to the TRAIN and CAR relationship, it includes also the LOAD relationship, which was skipped for simplicity in Fig. 2, which contains just the TRAIN and CAR relational tables). The boxes in the ER diagram indicate *entities*, which are individuals or parts of individuals. Here, the Train entity is the individual, each Car is part of a train, and each Load is part of a car. The ovals denote attributes of entities. The diamonds indicate *relationships* between entities. There is a *one-to-many relationship* from Train to Car, indicating that each train can have an arbitrary number of cars but each car is contained in exactly one train; and a *one-to-one relationship* between Car and Load, indicating that each car has exactly one load and each load is part of exactly one car.

```
t1: [car_roof_none, car_shape_rectangle, car_wheels_2,
    car_roof_none__car_shape_rectangle, car_roof_none__car_wheels_2,
    car_shape_rectangle__car_wheels_2, car_roof_peaked, car_shape_rectangle,
    car_wheels_3, car_roof_peaked__car_shape_rectangle,
    car_roof_peaked__car_wheels_3, car_shape_rectangle__car_wheels_3], east
...
t5: [car_roof_none, car_shape_rectangle, car_wheels_2,
    car_roof_none__car_shape_rectangle, car_roof_none__car_wheels_2,
    car_shape_rectangle__car_wheels_2, car_roof_flat, car_shape_hexagon,
    car_wheels_2, car_roof_flat__car_shape_hexagon,
    car_roof_flat__car_wheels_2, car_shape_hexagon__car_wheels_2], west
...
```

Fig. 3. The database from Fig. 2 in the Bag-Of-Words document representation.

	car_shape _rectangle	car_roof _peaked	car_wheels_3	car_roof_peaked__ car_shape_rectangle	car_shape_rectangle __car_wheels_3	...	class
t1	0.000	0.693	0.693	0.693	0.693	...	east
...	...	...	...	...	...	...	...
t5	0.000	0.000	0.000	0.000	0.000	...	west
...	...	...	...	...	...	...	...

Fig. 4. The transformed database (consisting of TF-IDF values, which are zero if the term appears in all documents) from Fig. 2 using the wordification approach. This final output can be given as an input to a propositional classifier.

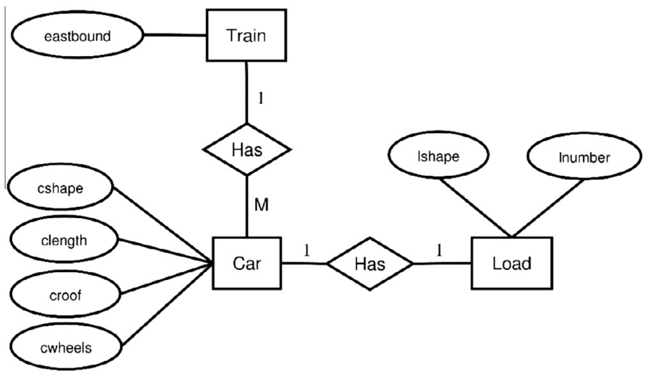


Fig. 5. Entity-relationship diagram for the East–West challenge.

Entity-relationship diagrams can be used to choose a proper logical representation for the data. If we store the data in a relational database the most obvious representation is to have a separate table for each entity in the domain, with relationships being expressed by *foreign keys*.[6] This is not the only possibility: for instance, since the relationship between Car and Load is one-to-one, both entities could be combined in a single table, while entities linked by a one-to-many relationship cannot be combined without either introducing significant redundancy or significant loss of information, e.g., introduced through aggregate attributes. Note that one-to-many relationships distinguish relational learning and Inductive Logic Programming from propositional learning.

In wordification, we use the entity-relationship diagram to define types of objects in the domain, where each entity will correspond to a distinct type. The data model constitutes a *language bias* that can be used to restrict the hypothesis space and guide the search. In most problems, only individuals and their parts exist as entities, which means that the entity-relationship model has a tree-structure with the individual entity at the root and only *one-to-one* or *one-to-many* relations in the downward direction. Representations with this restriction are called *individual-centered representations*. This restriction determines the language bias, constraining the relational database input to wordification.

4.2. Formal setting

The framework, established in this section, defines a learning setting which is very similar to the standard propositionalization problem setting. As in every propositionalization approach to Relational Data Mining, a two-step approach is implemented: (1) in the first propositionalization step the data is transformed from a relational database format to a tabular format, and (2) the tabular data is used as input for learning models or patterns by a selected propositional learner, having its own hypothesis language bias (e.g., decision trees or propositional classification rules). The formal framework described below focuses only on step (1) of the two-step wordification methodology. For simplicity, the formalization describes the setting using only unigram features.

Input. The input to wordification is a *relational database* (RDB), given as a set of relations $\{R_1, \dots, R_n\}$ and a set of foreign-key connections between the relations denoted by $R_i \rightarrow R_j$, where R_i has a foreign-key pointing to relation R_j. The foreign-key connections correspond to the relationships in the entity-relationship diagram. For example, the `train` attribute in the CAR relation is a foreign-key referring to `trainID` in TRAIN. It defines the `CAR` $\rightarrow$ `TRAIN` connection; as expected, it is a many-to-one connection from CAR to TRAIN.

A n-ary relation R_i is formally defined as a set of *tuples*: a subset of the Cartesian product of m_i domains: $R_i \subset \prod_{j=1}^{m_i} D_{i_j} = D_{i_1} \times D_{i_2} \times \dots \times D_{i_{m_i}}$, where a *domain* (or a *type*) is a specification of the valid set of values for the corresponding argument.

$$D_{i_j} = \{v_{i_{j_1}}, v_{i_{j_2}}, \dots, v_{i_{j_{k_{ij}}}}\}$$

Note that for wordification we require that each domain D_{i_j} must have a finite number of unique values k_{ij}, thus discretization of continuous domains is needed.

A further requirement is that the RDB must be individual-centered. This means that a target relation $R_T \in$ RDB must exist, such that it does not have any foreign keys:

$$\nexists i : R_T \rightarrow R_i; \quad R_i \in \text{RDB}$$

Output. Having established the data model, the individual-centered data representation language bias and the relational database representation of input data, the formal output (a transformed single-relation representation $R_{T'}$) of the wordification methodology can be defined as follows:

$$R_{T'} \subset \prod_{i,j,k} D_{T'_{ijk}} = \prod_{i,j,k} \text{domain}(R_i, D_{i_j}, v_{i_{j_k}}); R_i \xrightarrow{*} R_T$$

or in other words, one domain in the resulting relation $R_{T'}$ is defined for each relation R_i (that is connected by following the foreign-key path, denoted by $\xrightarrow{*}$ to R_T), and each of its domains D_{i_j} as well as domain values $v_{i_{j_k}}$. These domains have the property

$$D_{T'_{ijk}} = \mathbb{R}_0^+$$

since they are determined by the TF-IDF formula. This final output relation (table) can be given as an input to any propositional learner.

4.3. Wordification algorithm

This section presents the wordification methodology by describing in detail the individual transformation steps in Algorithms 1 and 2.

[6] In the context of relational databases, a foreign key is a field in a relational table that matches a candidate key of another table. The foreign key can be used to cross-reference tables.

Algorithm 1. Wordification(T, p, k)

```
Input  : target table $T$, pruning percentage $p$, maximal number of witems per word $k$
Output: Propositionalized table $R$ with TF-IDF values, corpus of documents $D$
1 $D \leftarrow []$;
2 $W \leftarrow \emptyset$ ;                                  // vocabulary set
3 for $ex \in T$ do
4     $d \leftarrow$wordify($T, ex, k$) ;                    // construct the document
5     $D \leftarrow D + [d]$ ;                          // append document to the corpus
6     $W \leftarrow W \cup keys(d)$ ;
7 end
8 $W \leftarrow$prune($W, p$) ;                              // optional step
9 return [ calculateTFIDFs($D, W$), $D$];
```

Algorithm 2. Wordify(T, ex, k)

```
Input  : table $T$, example $ex$ from table $T$, maximal number of witems per word $k$
Output: document word count $d$
1 $d \leftarrow \{\}$;                                // hash with a default value 0
2 for $i \leftarrow 1$ to $k$ do                         // for every word witem length
3     for $comb \in$ attrCombs($ex, k$) do        // attr.  combinations of length k
4         $d$[word($comb$)] $\leftarrow d$[word($comb$)] + 1;
5     end
6 end
7 // for every connected table through an example
8 for $secTable \in$ connectedTables($T$) do
9     for $secEx \in secTable$ do
10        if primaryKeyValue($ex$)=foreignKeyValue($secEx$) then
11            for $(word, count) \in$ wordify($secTable, secEx, k$) do
12                $d[word] \leftarrow d[word] + count$;
13            end
14        end
15    end
16 end
17 return $d$;
```

The algorithm starts recursive document construction on the instances of the main table (Lines 3–7 in Algorithm 1). First it creates word-items for the attributes of the target table (Lines 2–6 in Algorithm 2), which is followed by concatenations of the word-items and results of the recursive search through examples of the connecting tables (Lines 8–16 in Algorithm 2). As this document construction step is done independently for each example of the main table, this allows simultaneous search along the tree of connected tables. In order to perform concurrent propositionalization, Lines 3–7 in Algorithm 1 need to be run in parallel. A common obstacle in parallel computing is memory synchronization between the different subtasks, which is not the case here as concurrent processes in our implementation of wordification only need to share a cached list of subtrees. This list stores the results of subtree word concatenations in order to visit every subtree only once.

As wordification can produce a large number of features (words), especially when the maximal number of n-grams per word-items is large, we perform pruning of words that occur in less than a predefined percentage (5% on default) of documents. This reduces the size of trees by removing sections of the tree that is expected to provide little power for instance classification.

The constructed features are simple, and as we do not explicitly use existential variables in the new features (words), we instead rely on the Term Frequency-Inverse Document Frequency (TF-IDF) measure to implicitly capture the importance of a word for a given individual. In the context of text mining, TF-IDF value reflects how representative is a certain feature (word) for a given individual (document).

4.4. Time and space complexity

This section covers the worst-case complexity analysis of the wordification algorithm. Let t be the number of tables in a database. To simplify the analysis, we assume that each table is connected with exactly one other table in a one-to-many relation. Let m_i and n_i be the number of rows and the number of attributes

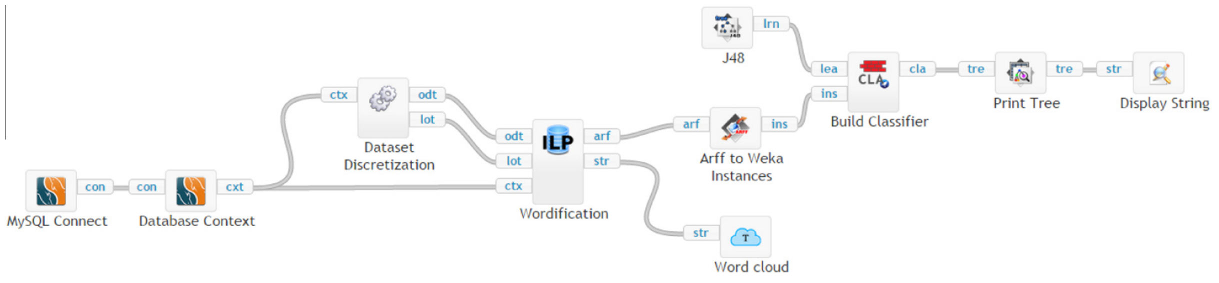


Fig. 6. Clowdflows wordification workflow with additional analyses after the wordification process. This workflow is publicly available at http://clowdflows.org/workflow/1455/. The abbreviations on the input and output stubs (which are not important for understanding the workflow) are as follows: *con* connection, *ctx* context, *odt* Orange data table, *lot* list of Orange data tables, *str* string, *arf* ARFF file, *ins* instances, *lrn* learner, *cla* classifier.

in table i, respectively. Further, let $m = \max(m_1, m_2, \ldots, m_t)$ and $n = \max(n_1, n_2, \ldots, n_t)$. The maximal number of rows is generally much higher than the number of attributes and the number of tables in a relational database: $t \ll n \ll m$. Let k be the maximal number of n-grams per word.

Time complexity. The upper-bound time complexity for the propositionalization step of the wordification methodology when each word is constructed from one witem $(k = 1)$ is $\mathcal{O}(m \cdot n \cdot m^{t-1}) = \mathcal{O}(n \cdot m^t)$. When we use words that are combinations of up to $1 \leqslant k \leqslant n$ witems (Lines 3–5 in Algorithm 2) the complexity of the algorithm is $\mathcal{O}\left(m^t \cdot \sum_{i=1}^{k} \binom{n}{i}\right)$. As $\lim_{k\to n} \sum_{i=1}^{k} \binom{n}{i} = 2^n - 1$, the worst-case time complexity is therefore $\mathcal{O}(2^n \cdot m^t)$.

Space complexity. The space complexity for the wordification algorithm using unigram features $(k = 1)$ is $\mathcal{O}(m \cdot t \cdot n)$. When we increase the maximal word length (number of witems per word) the feature space of the algorithm also increases exponentially. When the maximal word length is equal to the maximal number of attributes, the space complexity is $\mathcal{O}\left(m \cdot t \cdot \sum_{i=1}^{k} \binom{n}{i}\right)$. Following a similar reasoning as in the time complexity analysis, the worst case space complexity of the algorithm is therefore $\mathcal{O}(m \cdot t \cdot 2^n)$.

When $k \to n$ both time and space complexity are in its worst cases exponential in the number of attributes, but as evidenced from the experiments in Section 6, good performance can be achieved with $k = 1$ or $k = 2$, in which case the space complexity is linear $\mathcal{O}(m \cdot t \cdot n)$ and the time complexity is polynomial $\mathcal{O}(n \cdot m^t)$. Since t is usually small, the approach can perform orders of magnitude faster than its competitors, as demonstrated in Section 6.

5. Implementation

This section describes the implementation of the wordification methodology in the ClowdFlows platform. We briefly present the platform with its distinguishing features including the ILP module, followed by a description of the wordification workflow and its components.

5.1. The ClowdFlows platform

The ClowdFlows platform (Kranjc et al., 2012) is an open-source, web-based data mining platform that supports the construction and execution of scientific workflows. ClowdFlows differs from comparable platforms by its web based architecture. During run-time the ClowdFlows platform resides on a server (or on a cluster of machines) while its graphical user interface that allows workflow construction is served as a web application accessible from any modern web browser. ClowdFlows is essentially a cloud-based web application that can be accessed and controlled from anywhere while the processing is performed in a cloud of computing nodes.[7]

The ClowdFlows platform is written in Python using the Django framework. Its graphical user interface is implemented in JavaScript and features simple operations that allow workflow construction: adding workflow components (widgets) on a canvas and creating connections between the components to form executable workflows.

The platform has the following distinguishing features: an implementation of a visual programming paradigm, a web-service consumption module, a real-time processing module for mining data streams, the ability to easily share and publicize workflows constructed in ClowdFlows, and an ever growing roster of reusable workflow components and entire workflows.

Following a modular design, workflow components in ClowdFlows are organized into packages which allows for easier distributed development. ClowdFlows is easily extensible by adding new packages and workflow components by writing simple or complex Python functions. Workflow components of several types also allow graphical user interaction during runtime, visualization of results by implementing views in JavaScript, HTML or any other format that can be displayed in a web browser (e.g., Flash, Java Applet). The ClowdFlows packages include Weka algorithms (Witten et al., 2011), Orange algorithms (Demšar et al., 2004), text mining, as well as different ILP and RDM algorithms.

The current ILP module includes components, such as the popular ILP system Aleph (Srinivasan, 2007), as well as RSD (Železný & Lavrač, 2006), SDM (Vavpetič & Lavrač, 2013) and RelF (Kuželka & Železný, 2011). Aleph is an ILP toolkit on its own, with a wide range of functionalities: from decision tree learning to feature generation and first-order rule induction. We have extended the ClowdFlows ILP module with the implementation of the *wordification* component, which together with the existing components from different modules of the ClowdFlows platform forms the entire workflow of the wordification methodology, as can be seen in Fig. 6. Along with other components in the ILP module, wordification components can be used to construct diverse RDM workflows. As completed workflows, data, and results can also be made public by the author of the workflow, the platform can serve as an easy-to-access integration platform for various RDM workflows. Each public workflow is assigned a unique URL that can be accessed by anyone to either repeat the experiment, or use the workflow as a template to design another workflow.

5.2. Wordification workflow

This section describes the main components of the wordification workflow, which is shown in Fig. 6. The implementation

[7] A public installation of ClowdFlows is accessible at http://clowdflows.org.

Table 1
Table properties of the experimental data.

	# Rows	# Attributes
Trains		
Cars	63	9 (10)
Trains	20	2
Carcinogenesis		
Atom	9064	5
canc	329	2
sbond_1	13,562	4
sbond_2	926	4
sbond_3	12	4
sbond_7	4134	4
Mutagenesis 42		
Atoms	1001	5
Bonds	1066	5
Drugs	42	7
Ring_atom	1785	3
Ring_strucs	279	3
Rings	259	2
Mutagenesis 188		
Atoms	4893	5
Bonds	5243	5
Drugs	188	7
Ring_atom	9330	3
Ring_strucs	1433	3
Rings	1317	2
IMDB		
Actors	7118	4
Directors	130	3
Directors_genres	1123	4
Movies	166	4
Movies_directors	180	3
Movies_genres	408	3
Roles	7738	4
Financial		
Accounts	4500	4
Cards	892	4
Clients	5369	4
Disps	5369	4
Districts	77	16
Loans	682	7
Orders	6471	6
tkeys	234	2
Trans	1,056,320	10

allows the user to provide as input a relational database by connecting to a MySQL database server. First, the user is required to select the target table from the initial relational database, which will later represent the main table in the wordification component of the workflow. Second, the user is able to discretize each table using one of the various discretization techniques provided. These discretized tables are used by the wordification widget, where the transformation from the relational tables to a corpus of documents is performed. The workflow components are described in more detail below.

MySQL Connect. Since relational data is often stored in SQL databases, we use the MySQL package to access the training data by connecting to a MySQL database server. The MySQL Connect widget is used for entering information required to connect to a database (e.g., user credentials, database address, database name, etc.) in order to retrieve the training data from a MySQL database server and automatically construct the background knowledge and the training examples.

Database Context. This widget enables a selection of tables and columns that will be used in the next steps of the methodology. The information is carried to the connected widgets through the so-called database context objects. These objects also contain the detected table relationships. In case that the input relational database does not have predefined primary and foreign keys between the tables, the user is given an option for simple table connection search through the names of the attributes.

Table 2
Majority classifier performance for each dataset.

Domain	CA [%]
Trains	50.00
Mutagenesis 42	69.05
Mutagenesis 188	66.50
IMDB	73.49
Carcinogenesis	55.32
Financial	86.75

Dataset Discretization. The sole task of this widget is to convert continuous attributes to categorical, by discretizing the continuous attributes. Dataset Discretization widget supports three discretization methods: using equal-width intervals, using equal-frequency intervals, and class-aware discretization proposed by Fayyad and Irani (1993) that uses MDL and entropy to find the best cut-off points. Dataset Discretization widget can take as input either a single data set or a list of multiple datasets. In the latter case, discretization of all continuous attributes of every dataset is performed.

Wordification. The wordification widget transforms the relational database to a corpus of documents for the main table. As an input it takes three arguments: the target (main) table, a list of additional tables and a database context, which contains the relations between the tables. The widget first indexes the examples of every table by their primary and foreign keys' values. This step is required for performance optimization of data retrieval operations when searching for connecting instances from different (connected) tables in the word-item concatenation step. Next, recursive document construction for every individual is performed. The algorithm starts on every example of the main table: it creates word-items for its attributes, followed by concatenations of the word-items and results of the recursive search through the connecting tables. When searching along the tree of connected tables, the algorithm stores the results of subtree word concatenations for every instance. Consequently, the algorithm iterates over every subtree only once. The user can run the widget with different parameters: maximal number of *n*-grams per word, as well as the pruning threshold parameter. The wordification widget produces two outputs: a set of generated word documents and an *arff* table with calculated TF-IDF values for every example of the main table.

Word Cloud. A set of generated word documents can be displayed using word cloud visualization, enabling improved domain understanding.

Data Mining. After the wordification step the user can perform various types of analysis, depending on the task at hand (classification, clustering, etc). In the example workflow shown in Fig. 6, the *arff* output is used as input to build and display a J48 decision tree.

6. Experiments

This section presents the evaluation of the wordification methodology. After describing the relational databases used in this study, we describe the experiments performed on these datasets and provide a comparison of wordification to other propositionalization techniques. In comparison with the experimental setting described in Perovšek, Vavpetič, Cestnik, and Lavrač (2013), a larger number of datasets is used, and very favorable results are obtained by using decision tree learner J48, compared to relatively poor results reported in previous work, where the Naive Bayesian classifier assuming feature independence was used. In addition to the J48 tree learner, we also tested the LibSVM learner.

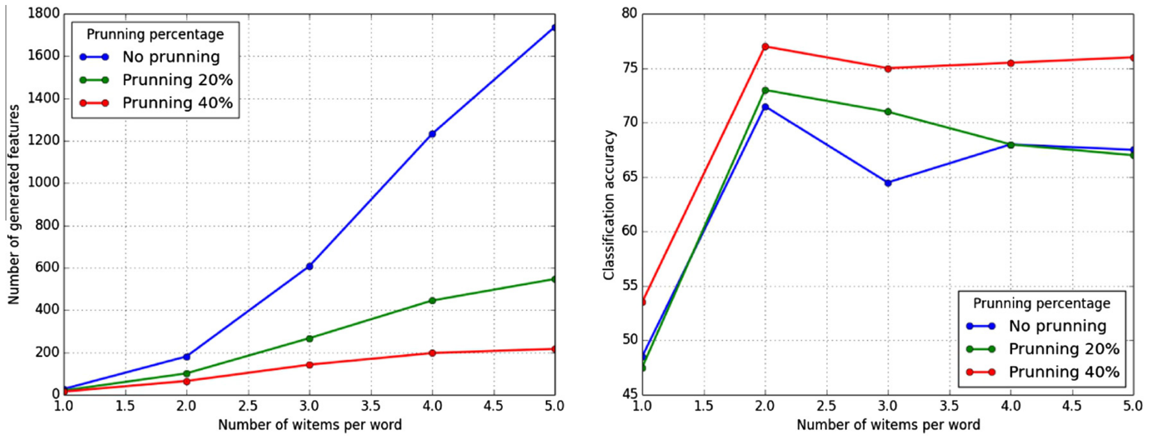


Fig. 7. Experiments on the wordification propositionalization step. Left: Size of the feature space in correlation with the number witems per word. Right: Classification accuracies in leave-one-out cross-validation (using the J48 decision tree learner with default parameter setting) as a function of the maximum number of *n*-grams per word.

Let us first present the five relational databases used in the experiments: Trains (in two variants), Carcinogenesis, Mutagenensis with 42 and 188 examples, IMDB, and Financial. Table 1 lists the characteristics of the datasets. All the datasets can be downloaded from a web page,[8] making them easily reusable in other experiments.

Trains. The well-known East–West Trains challenge is an ILP problem of predicting whether a train is East-bound or West-bound. A train contains a variable number of cars that have different shapes and carry different loads. We have considered two versions of the data for our experiments: the original dataset from the East–West Trains challenge and a modified dataset where every car also has its position as an additional attribute. In both datasets we have considered East-bound Trains as positive examples.

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

Mutagenesis In this task the goal is to predict mutagenicity of aromatic and heteroaromatic nitro compounds (Debnath, Lopez de Compadre, Debnath, Shusterman, & Hansch, 1991). 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 dataset 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 datasets of the original Debnath paper (Debnath et al., 1991), where the data was split into two subsets: a 188 compound dataset and a smaller dataset with 42 compounds.

IMDB The complete IMDB database is publicly available in the SQL format. This database contains tables of movies, actors, movie genres, directors, and director genres. The database used in our experiments consists only of the movies whose titles and years of production exist on the IMDB's top 250 and bottom 100 chart.[9] The database therefore consists of 166 movies, along with all of their actors, genres and directors. Movies present in the IMDB's top 250 chart were considered as positive examples, while those in the bottom 100 were regarded as negative.

[8] http://kt.ijs.si/janez_kranjc/ilp_datasets/.

[9] As of July 2, 2012.

Financial. This is a publicly available dataset, which was artificially constructed as part of the PKDD'99 Discovery Challenge. The classification task addressed is the prediction of successful loans. The dataset consists of 8 tables describing clients of a bank, their accounts, transactions, permanent orders, granted loans and issued credit cards.

Table 2 presents the performance of the majority classifier for each of the described datasets. This should serve as a baseline for the classification results reported in the following subsections.

6.1. Evaluation of feature construction and filtering

The experiments, enabling the analysis of the feature generation step of wordification were performed on the original East–West Trains challenge dataset using different parameter settings: using elementary word-items and complex word-items constructed from up to 5-grams of witems.

The left plot in Fig. 7 shows that the size of the feature space in the non-pruned version of wordification increases exponentially as the maximal number of witems per word increases. Note that wordification also implements a pruning technique where words that occur in less than a predefined percentage of documents are pruned. As shown in Fig. 7, using higher thresholds for feature filtering drastically reduces the dimensionality of the data, resulting in more efficient learning.

We have also applied different wordification settings in the classification task on the Trains dataset. The classification accuracies using the J48 decision tree of leave-one-out cross-validation for different parameters are shown on the right side of Fig. 7 (the reason for using leave-one-out instead of the standard 10-fold cross-validation setting is a very small number of instances in the Trains dataset). The results show that using larger *n*-grams of witems only marginally improves the classification accuracies, but results in longer run-times of the propositionalization step because of a larger feature space. In this specific domain, pruning performs favorably in terms of classification accuracy, though as the experiments in the Appendix A show (Table A.7), this observation is only applicable to small domains, while for larger domains with more potential witem combinations this observation does not hold.

6.2. Comparative evaluation of propositionalization techniques

This section describes the experiments performed in the evaluation of different propositionalization approaches on binary classification tasks, using the datasets from the five relational domains.

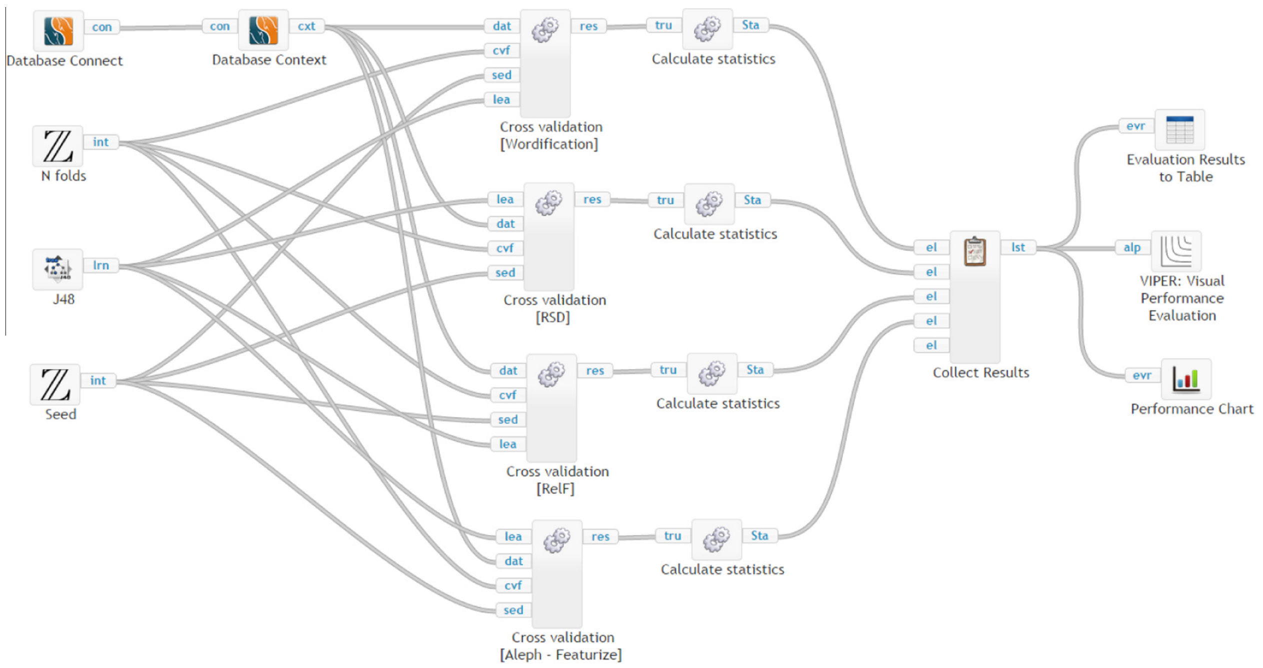


Fig. 8. Evaluation workflow for evaluating and comparing Wordification, Aleph, RSD, and RelF, implemented in the ClowdFlows data mining platform. The abbreviations on the input and output stubs are as follows: *con* connection, *ctx* context, *dat* full dataset for cross-validation, *cvf* number of cross-validation folds, *sed* random seed, *lrn*/*lea* learner instance, *res* cross-validation results, *sta* classification statistics, *evr* evaluation results.

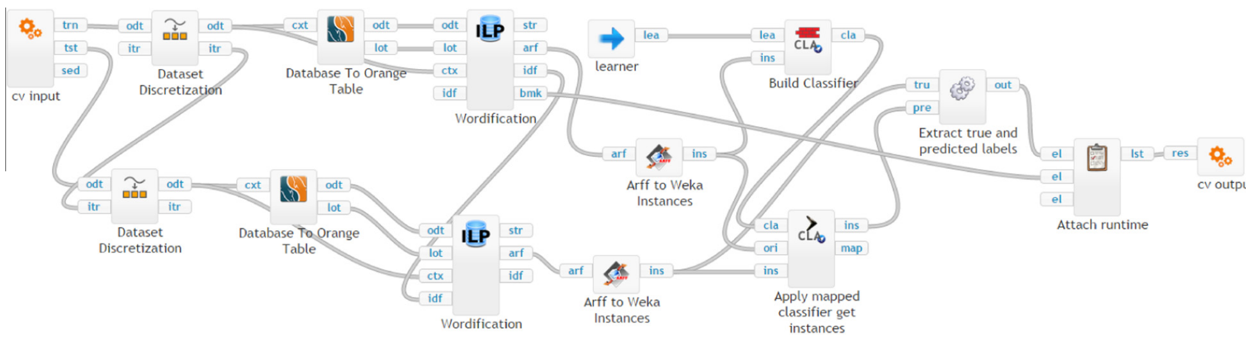


Fig. 9. The cross-validation subprocess from Fig. 8. This workflow gets a training set (input *trn*) and a test set (input *tst*) as input and is executed for each fold. There are two Wordification widgets in the workflows: one responsible for constructing the features on the Trains set and the other on the test set. The connection between the two widgets is needed for transferring the IDF weights learned on the training set, which are used for feature construction on the test set. The results of each step are collected by the *cv output* widget. Other widget input and output abbreviations are not important for understanding the workflow.

Fig. 8 shows the full experimental workflow (from connecting to a relational database management server to visualizing the experimental results and evaluation). This evaluation workflow is available online[10] in the ClowdFlows platform, which enables ILP researchers to reuse the developed workflow and its components in future experimentation.

The first step of the evaluation methodology is to read the relational data, stored in an SQL database, using the MySQL package widgets. Data then enters the cross-validation subprocess (Fig. 9), where the following steps are repeated for each fold (we used 10-fold cross-validation). First, discretization of the training fold of the relational database is performed. We have arbitrarily selected equi-distance discretization with 3 intervals of values to discretize the continuous attributes of the experimental relational datasets, such that none of the techniques was given an advantage. Then a propositionalization technique is applied to the training data and the results are formatted in a way to be used by the Weka algorithms. The J48 decision tree and the LibSVM learners were selected with their default parameter settings to perform binary classification.

The test set is handled as follows. First, the data is discretized to the intervals determined on the training set. Second, the features produced by the given propositionalization approach on the training set are evaluated on the test set to produce a propositional representation of the test data. Note that this process is slightly different for wordification, since the features do not have to be evaluated. We do however need the IDF values calculated on the training set. Finally, these test examples are classified by the classifiers trained on the training data. The results of each step are then collected to be returned at the end by the cross-validation subprocess.

Every propositionalization algorithm was run with its default settings. A non-parallel version of wordification was run using only the elementary words (maximal number of witems per word was set to 1) and without pruning, as none of our datasets required this.

[10] http://clowdflows.org/workflow/4018/.

Table 3
Classifier evaluation on different databases. The bolded items indicate the best results.

Domain	Algorithm	J48		LibSVM		Time [s]
		CA [%]	AUC	CA [%]	AUC	
Trains without position	Wordification	50.00	0.50	50.00	0.50	**0.11**
	RelF	**65.00**	**0.65**	**80.00**	**0.80**	1.04
	RSD	**65.00**	**0.65**	75.00	0.75	0.53
	Aleph – Featurize	60.00	0.60	65.00	0.65	0.40
Trains	Wordification	**95.00**	**0.95**	50.00	0.50	**0.12**
	RelF	75.00	0.75	75.00	0.75	1.06
	RSD	60.00	0.60	**80.00**	**0.80**	0.47
	Aleph – Featurize	55.00	0.55	70.00	0.70	0.38
Mutagenesis 42	Wordification	**97.62**	**0.96**	**78.57**	**0.65**	**0.39**
	RelF	76.19	0.68	76.19	0.62	2.11
	RSD	**97.62**	**0.96**	69.05	0.50	2.63
	Aleph – Featurize	69.05	0.50	69.05	0.50	2.07
Mutagenesis 188	Wordification	68.62	0.55	**81.91**	**0.78**	**1.65**
	RelF	**75.00**	**0.68**	68.62	0.54	7.76
	RSD	68.09	0.54	71.28	0.58	10.10
	Aleph – Featurize	60.11	**0.68**	60.11	0.68	19.27
IMDB	Wordification	**81.93**	**0.75**	73.49	0.50	**1.23**
	RelF	69.88	0.66	73.49	0.50	32.49
	RSD	74.70	0.59	73.49	0.50	4.33
	Aleph – Featurize	73.49	0.50	73.49	0.50	4.96
Carcinogenesis	Wordification	**62.31**	**0.61**	**60.79**	**0.58**	**1.79**
	RelF	60.18	0.59	56.23	0.52	16.44
	RSD	60.49	0.59	56.23	0.52	9.29
	Aleph – Featurize	55.32	0.50	55.32	0.50	104.70
Financial	Wordification	86.75	0.50	**86.75**	0.50	**4.65**
	RelF	**97.85**	**0.92**	86.70	0.50	260.93
	RSD	86.75	0.50	79,06	0.50	533.68
	Aleph - Featurize	86.75	0.50	**86.75**	0.50	525.86

RSD was specified to construct features with a maximum length of a feature body of 8. None of the constructed features were discarded as the minimum example coverage of the algorithm was set to 1.

Aleph was run in the feature construction mode (named AlephFeaturize in the evaluation workflow) with coverage as the evaluation function and maximal clause length of 4. The minimal number of positive examples was set to 1 and the maximal number of false positives to 0.

RelF, the most relevant of the algorithms in the TreeLiker software (Kuželka & Železný, 2011), was run in the default setting as well, but it is not clear from the documentation what exactly are the default parameter values. RelF expects a feature *template* from the user. In this case, we constructed relatively simple templates (enabling features with depth 1), since constructing and selecting more complex templates is out of scope for the analysis in this paper. It should be noted that templates more finely tuned to a particular domain could yield significantly better results. RelF also supports continuous attributes, but since in our experiments all approaches were given a discretized dataset, this feature could not be exploited.

6.3. Results comparison

The results of the experiments on multiple datasets, presented in Table 3, show the classification accuracy and the ROC AUC obtained by the J48 and LibSVM learners (when applied on the data obtained as a result of propositionalization approaches), as well as the run-times needed for propositionalization. The run-time performance for each algorithm was done by measuring the time an algorithm took to propositionalize the full database in each domain. The results show that the wordification methodology achieves scores comparable to the state-of-the-art propositionalization algorithms RSD and RelF, as well as compared to propositionalization performed by using features constructed by Aleph, while the run-time required for transforming the database into its propositional form is much faster.

In terms of classification accuracy obtained by the J48 classifier, wordification performs favorably compared to other propositionalization techniques, except on the Trains dataset (without the car's position attribute) and the Financial dataset. Poor performance on the Trains dataset can be explained by examining the J48 tree in the wordified Trains dataset with the position attribute, where the J48 classifier puts the cars_position_3 attribute into the root of the decision tree. Because of the absence of this attribute in the first dataset and the usage of only unigram words, the decision tree failed to find a clear distinction between the positive and negative examples (this problem can be solved by using bigrams of witems). Similar results were obtained using the LibSVM classifier where wordification achieved the best results on every dataset except for the two variants of the Trains data.

From the point of view of run-times, wordification is clearly the most efficient system, as it outperforms other techniques on every dataset. The true value of the wordification methodology, its low time-complexity, shows even more drastically on larger datasets, such as Carcinogenesis and Financial datasets, where it achieves comparable classification results in up to 100-times faster manner (compared to RSD or Aleph feature construction).

In order to statistically compare classification accuracies of multiple propositionalization approaches (separately for each of the classifiers) on multiple datasets, we applied the Friedman test (Friedman, 1937) using significance level $\alpha = 0.05$ and the corresponding Nemenyi post hoc test (Nemenyi, 1963). This approach is used as an alternative to the *t*-test, which is proven to be

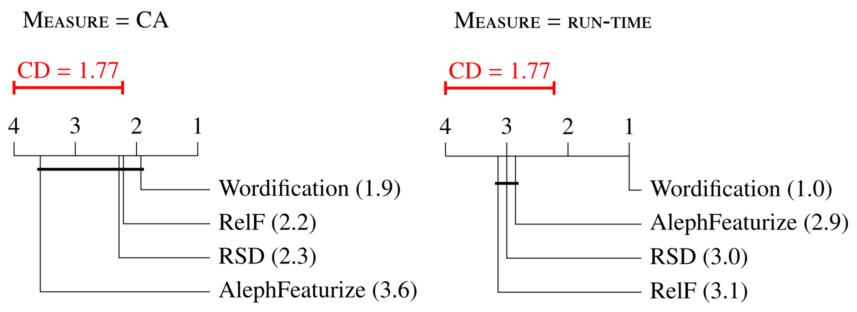


Fig. 10. Critical distance diagram for the reported classification accuracy (left; not enough evidence to prove that any algorithm performs better) and run-time (right; significant differences for $\alpha = 0.05$) results. The numbers in parentheses are the average ranks.

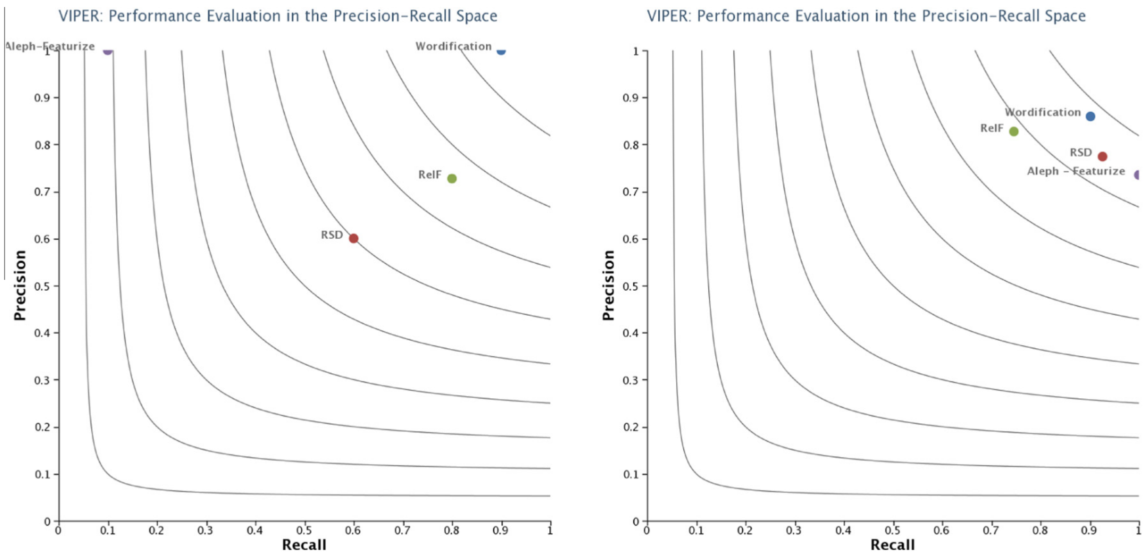


Fig. 11. The VIPER visualization showing evaluations of the standard J48 algorithm after applying propositionalizaton techniques. In the Trains dataset (left), 'East' was selected as the target class, while in the IMDB dataset (right) positive class was selected as the target.

Table 4
Table properties of the experimental data.

	# Rows	# Attributes
IMDB		
Movies	166	4
Roles	7738	2
Actors	7118	4
Movie_genres	408	2
Movie_directors	180	2
Directors	130	3
Director_genres	243	3
Accidents		
Accident	102,756	10
Person	201,534	10

Table 5
Document properties after applying the wordification methodology.

Domain	Individual	# Examples	# Words	# Words after filtering
IMDB	Movie	166	7453	3234
Accidents	Accident	102,759	186	79

inappropriate for testing multiple algorithms on multiple datasets (Demšar, 2006).

The Friedman test ranks the algorithms for each dataset, the best performing algorithm getting the rank of 1, the second best rank 2, etc. In the case of ties, average ranks are assigned. The Friedman test then compares the average ranks of the algorithms. The null-hypothesis states that all the algorithms are equivalent and so their ranks should be equal. If the null-hypothesis is rejected, we can proceed with a post hoc test, in our case the Nemenyi test. The Nemenyi test is used when we want to compare multiple algorithms to each other. The performance of the algorithms is significantly different if the average ranks differ by at least the *critical distance* (CD), as defined by Demšar (2006). This test can be visualized compactly with a critical distance diagram; see Fig. 10 for classification accuracy (CA) and run-time, when using J48 as the selected classifier (omitting AUC due to similar results obtained as for CA).

The described statistical test was performed using J48 for the three reported measures: classification accuracy, AUC and run-time. The validation yielded the following. For classification accuracy and AUC, there is not enough evidence to prove that any propositionalization algorithm on average performs better than the others (Fig. 10 left, for significance level $\alpha = 0.05$), even though wordification achieves the best results on 5 out of 7 benchmarks. This is due to the fact that the test takes into account the order of *all* algorithms, not only one versus the others.

We repeated the same statistical analysis for the LibSVM results, where the conclusion ended up the same. For classification accuracy and AUC, there is not enough evidence to prove that any propositionalization algorithm on average performs better than the others, even though wordification also achieves the best results on 5 out of 7 benchmarks.

For run-time, the results are statistically significant in favor of wordification; see the critical distance diagram in the right part of Fig. 10. The diagram tells us that the wordification approach performs statistically significantly faster than other approaches, under the significance level $\alpha = 0.05$. Other approaches fall within the same critical distance and no statistically significant difference was detected.

As shown in Fig. 8, the results of the Cross Validation widget (precision, recall, F-score) are connected to the input of the VIPER (Visual Performance Evaluation) widget. VIPER is an alternative evaluation visualization (Sluban, Gamberger, & Lavrač, 2014), implemented in the ClowdFlows data mining platform, which displays the results as points in the two dimensional precision-recall space (for the selected target class). Fig. 11 presents the VIPER performance visualization, evaluating J48 and LibSVM results after applying wordification, RSD, RelF and Aleph feature construction as propositionalizaton techniques. The results are presented in the so-called precision-recall space, where each point represents the result of an algorithm. Points closer to the upper-right corner have higher precision and recall values. F-measure values are presented as isolines (contour lines) in the precision-recall space, which allows a simple comparison of algorithm performances.

From the results shown in Fig. 11 we can conclude that in terms of precision and recall J48 achieves best results using the wordification propositionalization. Using the wordification methodology, not only a higher percentage of positive examples was retrieved (higher recall score), but also a slightly higher percentage of correctly classified examples of the target class (higher precision score) compared to other propositionalization techniques.

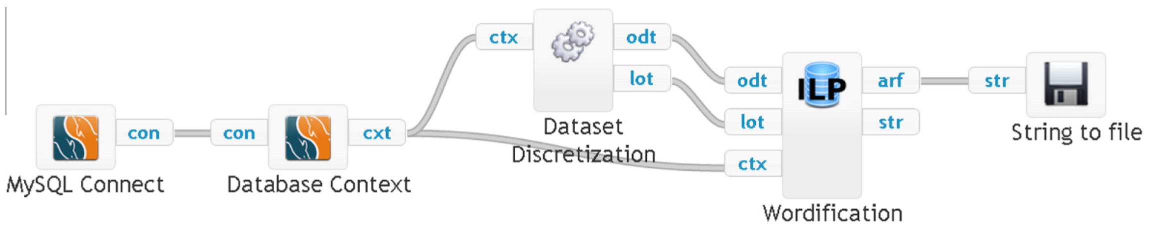


Fig. 12. Clowdflows wordification workflow used for feature construction before applying association rule learning. This workflow is publicly available at http://clowdflows.org/workflow/3969/. The abbreviations (not important for understanding the workflow) on the input and output stubs are as follows: *con* connection, *ctx* context, *odt* Orange data table, *lot* list of Orange data tables, *str* string, *arf* ARFF file, *ins* instances, *lrn* learner, *cla* classifier.

$$\begin{aligned}\texttt{goodMovie} \leftarrow\ & \texttt{director\_genre\_drama}, \texttt{movie\_genre\_thriller},\\ & \texttt{director\_name\_AlfredHitchcock}.\end{aligned}$$
(Support: 5.38% Confidence: 100.00%)

$$\texttt{movie\_genre\_drama} \leftarrow \texttt{goodMovie}, \texttt{actor\_name\_RobertDeNiro}.$$
(Support: 3.59% Confidence: 100.00%)

$$\texttt{director\_name\_AlfredHitchcock} \leftarrow \texttt{actor\_name\_AlfredHitchcock}.$$
(Support: 4.79% Confidence: 100.00%)

$$\begin{aligned}\texttt{director\_name\_StevenSpielberg} \leftarrow\ & \texttt{goodMovie}, \texttt{movie\_genre\_adventure},\\ & \texttt{actor\_name\_TedGrossman}.\end{aligned}$$
(Support: 1.79% Confidence: 100.00%)

Fig. 13. Examples of interesting association rules discovered in the IMDB database.

$$\begin{aligned}\texttt{noInjuries} \leftarrow\ & \texttt{accident\_trafficDensity\_rare},\\ & \texttt{accident\_location\_parkingLot}.\end{aligned}$$
(Support: 0.73% Confidence: 97.66%)

$$\texttt{person\_gender\_male} \leftarrow \texttt{person\_vehicleType\_motorcycle}.$$
(Support: 0.11% Confidence: 99.12%)

Fig. 14. Examples of interesting association rules discovered in the accidents database.

7. Applications

This section presents results of association rule learning experiments on two real-life relational databases: a collection of best and worst movies from the Internet Movie DataBase (IMDB) and a database of Slovenian traffic accidents. Tables 4 and 5 list the characteristics of both databases.

The preprocessing procedure was performed on the two databases as follows. First, the wordification step was applied. As shown in Fig. 12, we used ClowdFlows to read the relational data from the MySQL database, discretize continuous attributes and apply the propositionalization step. Due to lack of support for association rule learning in the ClowdFlows platform, the results of the wordification feature construction step were saved as an ARFF file and imported into RapidMiner (Mierswa, Wurst, Klinkenberg, Scholz, & Euler, 2006). Using RapidMiner we first removed irrelevant features (which have the same value across all the examples), which resulted in the reduction of the features to less than half of the original (see Table 5). In order to prepare the data for association rule mining, we also binarized the data: after experimenting with different TF-IDF thresholds, features with a higher TF-IDF weight than 0.06 were assigned the value *true* and *false* otherwise.

7.1. IMDB database

The complete IMDB database is publicly available in the SQL format.[11] This database contains tables of movies, actors, movie genres, directors, director genres.

The evaluation database used in our experiments consists only of the movies whose titles and years of production exist on IMDB's top 250 and bottom 100 chart. The database therefore consisted of 166 movies, along with all of their actors, genres and directors. Movies present in the IMDB's top 250 chart were added an additional label *goodMovie*, while those in the bottom 100 were marked as *badMovie*. Additionally, attribute age was discretized; a movie was marked as *old* if it was made before 1950, *fairlyNew* if it was produced between 1950 and 2000 and *new* otherwise.

After preprocessing the dataset using the wordification methodology, we performed association rule learning. Frequent itemsets were generated using RapidMiner's FP-growth implementation (Mierswa et al., 2006). Next, association rules for the resulting frequent itemsets were produced. Among all the discovered rules, several interesting rules were found. Fig. 13 presents some of the interesting rules selected by the expert. The first rule states that if the movie's genre is thriller and is directed by Alfred Hitchcock, who is also known for drama movies, then the movie is considered to be good. The second rule we have selected concludes that if the movie is good and Robert De Niro acts in it, than it must be a drama. The third interesting rule shows that Alfred Hitchcock acts only in the movies he also directs. The last rule concludes that if Ted Grossman acts in a good adventure movie, then the director is Steven Spielberg. Note that Ted Grossman usually plays the role of a stunt coordinator or performer.

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

7.2. Traffic accident database

The second dataset consists of all accidents that happened in Slovenia's capital Ljubljana between years 1995 and 2005. The data is publicly accessible from the national police department website.[12] The database contains the information about accidents along with all the accident's participants.

The data already contained discretized attributes, so further discretization was not needed. Similarly to the IMDB database, preprocessing using wordification methodology, FP-growth itemset mining and association rule mining were performed. Fig. 14 presents some of the interesting rules found in the Slovenian traffic accidents dataset.

The first rule indicates that if the traffic is rare and the accident happened in a parking lot, then no injuries occurred. The second rule implies that whenever a motorcycle is involved in an accident, a male person is involved.

8. Conclusions

This paper presents the propositionalization technique called wordification, which aims at constructing a propositional table using simple and easy to understand features. This methodology is inspired by text mining and can be seen as a transformation of a relational database into a corpus of documents, where document 'words' are constructed from attribute values by concatenating each table name, attribute name and value (called word-item or witem in this paper) into a single named-entity. As is typical for propositionalization methods, after the wordification step any propositional data mining algorithm can be applied.

As shown in the experiments on seven standard ILP datasets, the proposed wordification approach using the J48 and LibSVM classifiers performs favorably (in terms of accuracy and efficiency), compared to state-of-the-art propositionalization algorithms (RSD, RelF) as well as compared to propositionalization performed by using features constructed by Aleph. In addition, the proposed approach has the advantage of producing easy to understand hypotheses, using much simpler features than RSD and other systems, which construct complex logical features as conjunctions of first-order literals. It is interesting to observe that in wordification feature simplicity is compensated by the mechanism of feature weighting, inherited from text mining, which successfully compensates for the loss of information compared to complex relational features constructed by other propositionalization algorithms. In our experiments we also considered feature construction using *n*-grams. However, our preliminary experiments indicate that in larger domains this technique should be coupled with feature selection algorithms, which we plan to address in our further work.

Other advantages of wordification, to be explored in further work, include the capacity to perform clustering on relational databases; while this can be achieved also with other propositionalization approaches, wordification may successfully exploit document similarity measures and word clouds as easily understandable means of cluster visualization.

The implementation of the entire experimental workflow (from connecting to a relational database management server to visualizing the experimental results and evaluation) in the web-based data mining platform ClowdFlows is another major contribution, which will enable ILP researchers to reuse the developed software in future experimentation. To the best of our knowledge, this is the only workflow-based implementation of ILP algorithms in a platform accessible through a web browser, enabling simple workflow adaptation to the user's needs. Adding of new ILP algorithms to the platform is also possible by exposing the algorithm as a web service. This may significantly contribute to the accessibility and popularity of ILP and RDM methods in the future.

In terms of reusability of the workflows, accessible by a single click on a web page where the workflow is exposed, the ClowdFlows implementation of propositionalization algorithms is a significant step towards making the ILP legacy accessible to the research community in a systematic and user-friendly way. An additional building block in this vision is the incorporation of the VIPER visual performance evaluation engine, which enables algorithm comparison in terms of precision and recall, simplifying the experimental comparisons and results interpretation.

In future work, we will address other problem settings (such as clustering) and use the approach for solving real-life relational problems. Moreover, we plan to use the approach in a more elaborate scenario of mining heterogeneous data sources, involving a mixture of information from databases and text corpora. We will also further investigate the strength of *n*-gram construction and feature weighting, as used in the text mining community, in propositional and Relational Data Mining, as our results indicate that these mechanisms may successfully be used to compensate for the loss of information compared to constructing complex logical features.

Acknowledgments

We are grateful to Marko Grobelnik and Peter Ljubič for their initial work on this topic. This work was supported by the Slovenian Research Agency grants, and the FP7 European Commission project "Machine understanding for interactive storytelling" (MUSE, Grant agreement no: 296703).

Appendix A

Tables A.6 and A.7.

Table A.6
Evaluation of different feature weighting techniques. The bolded items indicate the best results.

Domain	Weighting	J48-accuracy [%]	J48-AUC
Trains without position	TF-IDF	50.00	0.50
	TF	**85.00**	**0.85**
	Binary	35.00	0.35
Trains	TF-IDF	**95.00**	**0.95**
	TF	80.00	0.80
	Binary	70.00	0.70
Mutagenesis 42	TF-IDF	97.62	0.96
	TF	97.62	0.96
	Binary	97.62	0.96
Mutagenesis 188	TF-IDF	**68.62**	**0.55**
	TF	68.09	0.54
	Binary	**68.62**	**0.55**
IMDB	TF-IDF	81.93	0.75
	TF	81.93	0.75
	Binary	81.93	0.75
Carcinogenesis	TF-IDF	62.31	0.61
	TF	62.61	0.61
	Binary	**62.92**	**0.62**
Financial	TF-IDF	86.75	0.50
	TF	86.75	0.50
	Binary	86.75	0.50

[12] http://www.policija.si/index.php/statistika/prometna-varnost.

Table A.7
Evaluation of different number of witems. The bolded items indicate the best results.

Domain	*k*	J48-Accuracy [%]	J48-AUC	Time [s]
Trains without position	1	50.00	0.50	**0.12**
	2	**75.00**	**0.75**	0.15
	3	**75.00**	**0.75**	0.20
Trains	1	**95.00**	**0.95**	**0.12**
	2	75.00	0.75	0.16
	3	70.00	0.70	0.22
Mutagenesis 42	1	**97.62**	**0.96**	**0.65**
	2	**97.62**	**0.96**	0.83
	3	92.86	0.88	0.88
Mutagenesis 188	1	**68.62**	**0.55**	**1.25**
	2	**68.62**	**0.55**	2.26
	3	66.49	0.50	2.68
IMDB	1	73.49	0.50	**0.16**
	2	73.49	0.50	0.20
	3	73.49	0.50	0.25
Carcinogenesis	1	**56.84**	**0.56**	**5.31**
	2	51.67	0.51	6.65
	3	52.58	0.51	7.04
Financial	1	86.75	0.50	**4.11**
	2	86.75	0.50	4.24
	3	86.75	0.50	4.38

References

Avila Garcez, A., & Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. *Applied Intelligence, 11*(1), 59–77.

Ceci, M., & Appice, A. (2006). Spatial associative classification: Propositional vs structural approach. *Journal of Intelligent Information Systems, 27*(3), 191–213.

Ceci, M., Appice, A., & Malerba, D. (2008). Emerging pattern based classification in relational data mining. In S. Bhowmick, J. Kng, & R. Wagner (Eds.), *Database and expert systems applications. Lecture notes in computer science* (Vol. 5181, pp. 283–296). Berlin Heidelberg: Springer.

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

Demšar, J., Zupan, B., Leban, G., & Curk, T. (2004). Orange: From experimental machine learning to interactive data mining. In J.-F. Boulicaut, F. Esposito, F. Giannotti, & D. Pedreschi (Eds.), *PKDD 2004. Proceedings of 8th european conference on principles and practice of knowledge discovery in databases September 20–24, 2004. Lecture notes in computer science* (Vol. 3202, pp. 537–539). Pisa, Italy: Springer.

Demšar, J. (2006). Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

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

Fayyad, U., & Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In R. Bajcsy (Ed.), *IJCAI-93. Proceedings of the 13th international joint conference on artificial intelligence, August 28–September 3, 1993* (pp. 1022–1029). Chambéry, France: Morgan Kaufman.

Flach, P. A., Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. In: *Proceedings of the 9th international workshop on inductive logic programming* (pp. 92–103).

França, M., Zaverucha, G., & d'Avila Garcez, A. (2014). Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning, 94*(1), 81–104.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association, 32*, 675–701.

Guo, H., & Viktor, H. (2008). Multirelational classification: A multiple view approach. *Knowledge and Information Systems, 17*(3), 287–312.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation, 28*, 11–21.

Knobbe, A. J. (Ed.). (2005). *Multi-relational data mining. Frontiers in artificial intelligence and applications* (Vol. 145). IOS Press.

Kramer, S., Pfahringer, B., & Helma, C. (1998). Stochastic propositionalization of non-determinate background knowledge. In D. Page (Ed.), *ILP-98. Proceedings of the 8th international workshop on inductive logic programming, July 22–24, 1998. Lecture notes in computer science* (Vol. 1446, pp. 80–94). Madison, Wisconsin, USA: Springer.

Kranjc, J., Podpečan, V., & Lavrač, N. (2012). Clowdflows: A cloud based scientific workflow platform. In P. A. Flach, T. D. Bie, & N. Cristianini (Eds.), *ECML PKDD 2012. Proceedings (part II) of the machine learning and knowledge discovery in databases – European conference, September 24–28, 2012. Lecture notes in computer science* (Vol. 7524, pp. 816–819). Bristol, UK: Springer.

Krogel, M. A., Rawles, S., Železný, F., Flach, P. A., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In T. Horváth (Ed.), *ILP 2003. Proceedings of the 13th international conference on inductive logic programming, September 29–October 1, 2003. Lecture notes in computer science* (Vol. 2835, pp. 197–214). Szeged, Hungary: Springer.

Krogel, M. A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In C. Rouveirol & M. Sebag (Eds.), *ILP 2001. Proceedings of the 11th international conference on inductive logic programming, September 9–11, 2001. Lecture notes in computer science* (Vol. 2157, pp. 142–155). Strasbourg, France: Springer.

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

Lavrač, N., & Džeroski, S. (1994). Inductive logic programming. In *WLP* (pp. 146–160). Springer.

Lavrač, N., Džeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff (Ed.), *EWSL-91. Proceedings of the european working session on learning, March 6–8, 1991. Lecture notes in computer science* (Vol. 482, pp. 265–281). Porto, Portugal: Springer.

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

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

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). YALE rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06)* (pp. 935–940). NY, USA: ACM Press.

Muggleton, S. (Ed.). (1992). *Inductive logic programming*. London: Academic Press.

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing, Special issue on Inductive Logic Programming, 13*(3–4), 245–286.

Nemenyi, P. B. (1963). *Distribution-free multiple comparisons* (Ph.D. thesis).

Perovšek, M., Vavpetič, A., Cestnik, B., & Lavrač, N. (2013). A wordification approach to relational data mining. In J. Fürnkranz, E. Hüllermeier, & T. Higuchi (Eds.), *DS 2013. Proceedings of the 16th international conference on discovery science, October 6–9, 2013. Lecture notes in computer science* (Vol. 8140, pp. 141–154). Singapore: Springer.

Perovšek, M., Vavpetič, A., Lavrač, N. (2012). A wordification approach to relational data mining: Early results. In: F. Riguzzi, F. Železný (Eds.), *ILP 2012. Proceedings of late breaking papers of the 22nd international conference on inductive logic programming* (pp. 56–61). Dubrovnik, Croatia, September 17–19, 2012. Vol. 975 of CEUR Workshop Proceedings. CEUR-WS.org.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management, 24*(5), 513–523.

Sluban, B., Gamberger, D., & Lavrač, N. (2014). Ensemble-based noise detection: Noise ranking and visual performance evaluation. *Data Mining and Knowledge Discovery*, 1–39.

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 N. Lavrač & S. Džeroski (Eds.), *ILP-97. Proceedings of the 7th international workshop on inductive logic programming, September 17–20, 1997. Lecture notes in computer science* (Vol. 1297, pp. 273–287). Prague, Czech Republic: Springer.

Srinivasan, A., Muggleton, S., King, R., Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In: S. Wrobel (Ed.), *Proceedings of the 4th international workshop on inductive logic programming*. Vol. 237. GMD-Studien. Gesellschaft für Mathematik und Datenverarbeitung MBH (pp. 217–232).

Vavpetič, A., & Lavrač, N. (2013). Semantic subgroup discovery systems and workflows in the SDM-toolkit. *The Computer Journal, 56*(3), 304–320.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Železný, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning, 62*(1–2), 33–63.

Chapter 5

Use Case: Active Learning for Sentiment Analysis on Data Streams

This chapter presents a novel use case scenario—active learning for sentiment analysis on data streams. The clear advantage of ClowdFlows being available as a web application is that it is always online because the data processing takes place on reliable remote servers. This allows for user intervention at any time, providing a suitable environment for an active learning use case. Coupled with the ability of ClowdFlows to process streams of data makes it ideal to process a popular source of textual data such as the microblogging platform Twitter.

The first section presents the field of sentiment analysis and its challenges. This is followed by a description of active learning for sentiment analysis on data streams. The next section presents a general overview of data stream processing, lists the most common approaches therein and provides a brief description on how data stream processing was implemented in ClowdFlows. The chapter concludes with a journal publication that describes publicly available on-line workflows for dynamic adaptive sentiment analyses, which are able to handle changes in data streams and adapt their behavior over time via human-computer interaction.

5.1 Sentiment Analysis

Sentiment analysis refers to the use of natural language processing, text mining, and machine learning for detection of attitude, emotion or opinion about a given topic expressed in the text by the author [77]. A basic task in sentiment analysis is to classify text as being positive, negative, or neutral. While some approaches perform more complex analyses, in this use case we address basic sentiment analysis of short textual messages, classifying them into two (positive or negative) categories.

Sentiment analysis can be performed using several different approaches. *Lexicon-based* methods employ sentiment lexicons for determining the sentiment in text [78]. These are fast methods that are usually unable to adapt to context changes which may occur in data streams. *Linguistic* methods analyze the grammatical structure of the text to determine its sentiment [79]. These approaches tend to be more computationally demanding and are not suitable for a streaming setting. *Machine learning* methods require a labeled data collection for training classifiers [80]. Experiments have shown that the latter is the most suitable approach for the streaming setting [81].

As the use case deals with data obtained from the web, the task of sentiment analysis is especially challenging since such data often contains informal language [82], as well as grammatical and spelling mistakes [83]. Therefore, it is important to perform appropriate preprocessing to adapt the input to sentiment analysis algorithms.

5.2 Active Learning

Active learning is a special case of supervised machine learning where a learning algorithm has a collection of unlabeled instances at its disposal, but it can select only a limited number of them for hand-labeling [84]. The goal of active learning is to select instances in the best possible way so that the performance of the algorithm is maximized and the hand-labeling effort is minimized. The labeled instances are then used for learning. Active learning is useful in situations when labeling is difficult, time-consuming, or expensive [85].

Active learning is applied to various domains such as spam filtering [86], part-of-speech tagging [87], text classification [88], and others. We apply it to the sentiment prediction on data streams of short documents from microblogging platforms (such as Twitter).

Algorithms that analyze constantly incoming data have to be up-to-date with changes in the data stream, provide fast data processing, and deal with large data volumes in real-time. Trained models can become outdated due to the changes in the data stream and stream-based active learning is a reasonable solution to this problem, since its main task is to continuously select the most suitable examples from the data stream to update the model.

Sentiment analysis has previously been performed on Twitter data streams. The MOA-TweetReader [89] system performs real-time analysis of Twitter messages. The system detects changes in word frequencies and performs sentiment analysis, but does not use active learning to adapt the model. Another system was developed for Twitter sentiment analysis in real-time for the US elections [90]. There users can observe aggregated sentiment, volume, statistics, trending words, tag clouds, or individual tweets. The system allows users to label arbitrarily chosen tweets, but the system does not suggest tweets for labeling, or use labeled tweets for updating the sentiment model, so no active learning takes place.

5.3 Data Stream Mining

Data stream mining is a process of extracting knowledge structures from continuous data records. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities.

In this section we present the challenges that data stream mining presents and some common approaches employed in processing streams of data. This is followed by a description how data stream processing is implemented in a graphical workflow environment in the ClowdFlows platform.

5.3.1 Challenges and approaches

Applications that generate data at very high rates in the form of transient *data streams* require intelligent data processing and online analysis. Examples of such applications include financial applications, network monitoring, security, sensor networks, web applications, social media web sites, and others. Rapid generation of individual data items in forms of continuous streams of information has challenged storage, computation and communication capabilities in computing systems.

Among the many challenges of data stream mining that need to be addressed are the following: memory management, data pre-processing, visualization of streaming data, and resource-aware mining. *Memory management* [91] is the first fundamental issue that needs to be considered. A stream mining algorithm with high memory requirement will have difficulty being applied in many situations. *Data pre-processing* is an important and time-consuming phase in the knowledge discovery process and must be taken into consideration when mining data streams. Designing light-weight preprocessing techniques that can guarantee the quality of the mining results is an important aspect. The challenge is to automate such a process and integrate it with the mining techniques [37]. *Visualization of results* is a powerful way to facilitate data analysis. Data should be visualized in an efficient, responsive way so that it reflects the current or overall stream of data. Stream mining processes need to be *resource aware*. Stream data mining algorithms must not ignore the problem of consuming the available resources or the risk of losing data when the memory is used up [92].

Common approaches to data stream mining are mostly based on summarization techniques for producing approximate answers from large databases [93]. This can be done by either summarizing the whole dataset or by choosing a subset of the incoming stream to be analyzed. The techniques include sampling, sketching, aggregation, and sliding window [93]. *Sampling* is the process of statistically selecting the elements of the incoming stream to be analyzed [94]. Sampling is often used as a data reduction technique for producing approximate answers for queries over data streams [95]. *Sketching* is a technique where a summary of data is built using a small amount of memory [96]. It is the process of vertically sampling the incoming stream. *Aggregation* is a representation of a number of elements in one aggregated element using an arbitrary statistical measure. It is considered a data rate adaptation technique in resource-aware mining [97]. Finally, one of the techniques for producing approximate answers to a data stream query is called a *sliding window*. The idea is to perform a detailed analysis over the most recent data items and over summarized versions of the old ones.

5.3.2 Data stream processing in ClowdFlows

Standard workflows in ClowdFlows receive data at inputs and return results at the output. With the exception of iterating subworkflows, each workflow component is traditionally only executed once. In order to process a data stream, workflow components need to be executed whenever a new data instance arrives at the input.

In order to process potentially infinite data streams we have augmented the ClowdFlows workflow execution engine to execute workflows multiple times at arbitrarily small temporal intervals in parallel. In an ideal scenario the workflow should be executed whenever there is a new data instance on the input. Due to differences in connections to data streams (e.g., some need to be continuously polled) setting the frequency and parallelism needs to be optimized for each stream mining process separately. The amount of parallelism is usually dependent on the amount of processing units in the ClowdFlows installation, while the frequency is dependent on the incoming rate of data instances.

The execution of a stream mining workflow is delegated to a process referred to as the *stream processing daemon* which issues tasks to workers (processing units) that execute workflows. Special workflow components and mechanisms were implemented to ensure that all data is processed and each data instance is processed only once. Most notably, two important mechanisms were implemented to enable processing of data streams. Workflow components in stream mining have an *internal memory* which stores data about the processed stream, such as the time stamp of the last processed data instance, or an instance of the data itself. The *halting mechanism* which stops executions of workflows is used to optimize the run-time. This mechanism is activated by the workflow components. These two mechanisms enable implementation of several general purpose stream mining workflow components.

To process data streams, *streaming data inputs* were implemented. These are components that consume the stream. In short, a streaming input workflow component connects to an external data source, collects data instances and uses its internal memory to remember data from previous instances (so as to eliminate repetition). Several workflow components were developed that conform to the approaches of stream mining described in Section 5.3.1. The *aggregation* component collects a fixed number of data instances before passing the data to the next component. The internal memory of the widget is used to save the data instances until an arbitrary threshold is reached. As long as the number of instances is below the threshold, the component invokes the halting mechanism which stops the execution of the workflow for this data instance. The *sliding window* component is similar to aggregation, with the exception that it does not empty its entire internal memory upon reaching the threshold. The oldest few instances are removed from the internal memory and the instances in the sliding window are released to other widgets in the workflow. *Sampling* workflow components either pass the instance to the next component or halt the execution based on an arbitrary condition. Several *streaming visualization* components were developed for the purpose of examining results of real-time analyses. Each instance of a stream visualization component automatically creates a Web page with a unique URL that displays the results.

The stream processing daemon coupled with the stream mining workflow components allow for reliable processing of data streams in ClowdFlows and enable execution of various use cases.

5.4 Related Publication

The main idea of the use case is to create a workflow for processing arbitrary data streams and to apply active learning to create dynamic adaptive models for sentiment analysis. The web-based nature of ClowdFlows also allows for crowdsourcing of labeling examples.

A derivation of this use case was also applied to monitor the Twitter sentiment during the Bulgarian parliamentary elections [98].

The implementation details and description of the use case scenario are included in the following journal publication:

J. Kranjc, J. Smailović, V. Podpečan, M. Grčar, M. Žnidaršič, and N. Lavrač, "Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform," *Information Processing & Management*, vol. 51, no. 2, pp. 187–203, 2015.

In this publication we achieve the following:

- We present the real time capabilities of ClowdFlows and show the limitations of its concurrent use.
- We implement an active learning scenario for sentiment analysis on data streams.
- We show that machine learning methods are suitable for sentiment analysis.
- We show that active learning improves the accuracy of sentiment classification.
- We provide useful visualizations of textual data streams.

The authors' contributions are as follows. The ClowdFlows platform was augmented by Janez Kranjc with helpful insights from Vid Podpečan to facilitate mining of real time data and provide an interface for active learning. Jasmina Smailovič implemented the sentiment anlaysis algorithms and conducted active learning experiments to determine the best strategies for active learning. Miha Grčar provided the tools and algorithms used in determining the sentiment and building the models. Martin Žnidaršič supervised the sentiment analysis and active learning on the conceptional level while Nada Lavrač supervised the implementation of stream mining in ClowdFlows. All authors contributed to the text of the publication.

Information Processing and Management 51 (2015) 187–203

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform

Janez Kranjc [a,b,*], Jasmina Smailović [a,b], Vid Podpečan [a,c], Miha Grčar [a], Martin Žnidaršič [a], Nada Lavrač [a,b,d]

[a] Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia
[c] University of Ljubljana, Faculty of Mathematics and Physics, Jadranska 19, 1000 Ljubljana, Slovenia
[d] University of Nova Gorica, Vipavska 13, 5000 Nova Gorica, Slovenia

ARTICLE INFO

Article history:
Received 30 October 2013
Received in revised form 27 March 2014
Accepted 7 April 2014
Available online 1 May 2014

Keywords:
Active learning
Stream mining
Sentiment analysis
Stream-based active learning
Workflows
Data mining platform

ABSTRACT

Sentiment analysis from data streams is aimed at detecting authors' attitude, emotions and opinions from texts in real-time. To reduce the labeling effort needed in the data collection phase, active learning is often applied in streaming scenarios, where a learning algorithm is allowed to select new examples to be manually labeled in order to improve the learner's performance. Even though there are many on-line platforms which perform sentiment analysis, there is no publicly available interactive on-line platform for dynamic adaptive sentiment analysis, which would be able to handle changes in data streams and adapt its behavior over time. This paper describes ClowdFlows, a cloud-based scientific workflow platform, and its extensions enabling the analysis of data streams and active learning. Moreover, by utilizing the data and workflow sharing in ClowdFlows, the labeling of examples can be distributed through crowdsourcing. The advanced features of ClowdFlows are demonstrated on a sentiment analysis use case, using active learning with a linear Support Vector Machine for learning sentiment classification models to be applied to microblogging data streams.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

This paper addresses a data mining scenario at the intersection of active learning, sentiment analysis, stream mining and service-oriented knowledge discovery architectures effectively solved by on-line workflow implementation of the developed active learning methodology for sentiment analysis from streams of Twitter data.

Active learning is a well-studied research area (Sculley, 2007; Settles, 2011; Settles & Craven, 2008), addressing data mining scenarios where a learning algorithm can periodically select new examples to be labeled by a human annotator and add them to the training dataset to improve the learner's performance on new data. Its aim is to maximize the performance of the algorithm and minimize the human labeling effort. *Sentiment analysis* (Liu, 2012; Pang & Lee, 2008; Turney, 2002) is concerned with the detection of the author's attitude, emotion or opinion about a given topic expressed

* Corresponding author at: Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia. Tel.: +386 1 477 3655.
E-mail address: janez.kranjc@ijs.si (J. Kranjc).

http://dx.doi.org/10.1016/j.ipm.2014.04.001
0306-4573/© 2014 Elsevier Ltd. All rights reserved.

in the text. The task of sentiment analysis is especially challenging in the context of analyzing user generated content from the Internet (Petz et al., 2012, 2013). *Stream mining* (Gama, Rodrigues, Spinosa, & de Carvalho, 2010) is an online learning paradigm, aiming to incorporate the information from the evolving data stream into the model, without having to re-learn the model from scratch; while batch learning is a finite process that starts with a data collection phase and ends with a stationary model, the online learning process starts with the arrival of some training instances and lasts as long as there is new data available for learning. As such, it is a dynamic process that has to encapsulate the collection of data, the learning and the validation phase in a single continuous cycle.

This paper introduces a cloud-based scientific workflow platform, which is able to perform on-line dynamic adaptive sentiment analysis of microblogging posts. Even though there are many on-line platforms which apply sentiment analysis on microblogging texts, there is still no such pltaform that could be used for on-line dynamic adaptive sentiment analysis and would thus be able to handle changes in data streams and adapt its components over time. In order to provide continuous updating of the sentiment classifier with time we used an active learning approach. In this paper, we address this issue by presenting an approach to interactive stream-based sentiment analysis of microblogging messages in a cloud-based scientific workflow platform ClowdFlows.[1] With the aim to minimize the effort required to apply labels to tweets, this browser-based platform provides an easy way to share the results and a Web interface for labeling tweets.

ClowdFlows is a new open-sourced data mining platform designed as a cloud-based Web application in order to overcome several deficiencies of similar data mining platforms, providing a handful of novel features that benefit the data mining community. ClowdFlows was first developed as a data mining tool for processing static data (Kranjc, Podpecian, & Lavraci, 2012a, 2012b), which successfully bridges different operating systems and platforms, and is able to fully utilize available server resources in order to relieve the client from heavy-duty processing and data transfer as the platform is entirely Web based and can be accessed from any modern browser. ClowdFlows also benefits from a service-oriented architecture which allows users to utilize arbitrary Web services as workflow components. In this paper we present the adaptation of the ClowdFlows platform, enabling it to work on real time data streams. As a result, workflows in ClowdFlows are no longer limited to static data on the server but can connect to multiple data sources and can process the data continuously. One such data source is the Twitter API which provides a potentially infinite stream of tweets which are the subject of sentiment analysis in this paper.

The paper is structured as follows. Section 2 presents the related work. Comparable data mining and stream mining platforms are presented and their differences and similarities with ClowdFlows are discussed. Related work concerning active learning in data streams is also presented. Section 3 presents the technical background and implementation details of the ClowdFlows platform. The architecture of the system is presented along with specific methods that allow stream mining in a workflow environment. The proposed sentiment analysis and active learning methods are presented in Section 4. The implementation details and the workflow enabling active learning for sentiment analysis are presented in Section 5. In Section 6 we conclude the paper by presenting the directions for further work.

2. Related work

This section presents an overview of data mining platforms and their key features: visual programming and execution of scientific workflows, diversity of workflow components, service-oriented architectures, remote workflow execution, big data processing, stream mining, and data sharing. The overview is followed presenting current research in the field of active learning on data streams.

2.1. Data mining platforms

Visual construction and execution of scientific workflows is one of the key features of the majority of current data mining software platforms. It enables the users to construct complex data analysis scenarios without programming and allows to easily compare different options. All major data mining platforms, such as Weka (Witten, Frank, & Hall, 2011), RapidMiner (Mierswa, Wurst, Klinkenberg, Scholz, & Euler, 2006), KNIME (Berthold et al., 2007) and Orange (Demšar, Zupan, Leban, & Curk, 2004) support workflow construction. The most important common feature is the implementation of a *workflow canvas* where complex workflows can be constructed using simple drag, drop and connect operations on the available components. The range of available components typically includes database connectivity, data loading from files and preprocessing, data and pattern mining algorithms, algorithm performance evaluation, and interactive and non-interactive visualizations.

Even though such data mining software solutions are reasonably user-friendly and offer a wide range of components, some of their deficiencies severely limit their utility. Firstly, all available workflow components provided by any of these platforms are specific and can be used only in the given platform. Secondly, the described platforms are implemented as standalone applications and have specific hardware and software dependencies. Thirdly, in order to extend the range of available workflow components in any of these platforms, knowledge of a specific programming language is required. This also means that they are not capable of using existing software components, implemented as Web services, freely available on the Web.

[1] http://clowdflows.org.

As a benefit of service-oriented architecture concepts, software tools have emerged, which are able to make use of Web services, and can access large public databases (some supporting grid deployment and P2P computing). Environments such as Weka4WS (Talia, Trunfio, & Verta, 2005), Orange4WS (Podpečan, Zemenova, & Lavrač, 2012), Web Extension for Rapid-Miner, Triana (Taylor, Shields, Wang, & Harrison, 2007), Taverna (Hull et al., 2006) and Kepler (Altintas et al., 2004) allow for the integration of Web services as workflow components. However, with the exception of Orange4WS and Web Extension for RapidMiner, these environments are mostly specialized to domains like systems biology, chemistry, medical imaging, ecology and geology. Lastly, all mentioned platforms are still based on technologies that do not benefit from modern Web technologies which enable truly independent software solutions. On the other hand, Web-based workflow construction environments exist, which are however too general and not coupled to any data mining library. For example, Oryx Editor (Decker, Overdick, & Weske, 2008) can be used for modeling business processes and workflows while the Galaxy (Blankenberg et al., 2001, chap. 19) genome analysis tool (implemented as a Web application) is limited exclusively to the workflow components provided by the project itself.

Remote workflow execution (on different machines than the one used for workflow construction) is employed by KNIME Cluster execution and RapidMiner using the RapidAnalytics server. This allows the execution of workflows on more powerful machines and data sharing with other users, with the requirement that the client software is installed on the user's machine. The client software is still used for designing workflows which are executed on remote machines, while only the results can be viewed using a Web interface.

In support of the ever increasing amount of data several truly distributed software platforms have emerged. Such platforms can be categorized into two groups: batch data processing and data stream processing. A well known example of a distributed batch processing framework is Apache Hadoop,[2] an open-source implementation of the MapReduce programming model (Dean & Ghemawat, 2008) and a distributed file system called Hadoop Distributed Filesystem (HDFS). It is used in many real life environments and several modifications and extensions exist, also for online (stream) processing (Condie et al., 2010) (parallelization of a variety of learning algoriths using an adaptation of MapReduce is discussed by Chu et al. (2006)). Apache Hadoop is also the base framework of Apache Mahout,[3] a machine learning library for large data sets, which currently supports recommendation mining, clustering, classification and frequent itemset mining. Radoop,[4] a commercial big data analytics solution, is based on RapidMiner and Mahout, and uses RapidMiner's data flow interface.

For data stream processing, two of the most known platforms were released by Yahoo! (the S4 platform[5]) and Twitter (Storm[6]). SAMOA (Morales, 2013) is an example of a new generation platform which is targeted at processing big data streams. In contrast with distributed data mining tools for batch processing using MapReduce (e.g., Apache Mahout), SAMOA features a pluggable architecture on top of S4 and Storm for performing the most common tasks such as classification and clustering. However, the platform is under development, no software has been released yet and it is not known whether the platform will support visual programming with workflows. MOA (Massive On-line Analysis) is a non-distributed framework for mining data streams (Bifet, Holmes, Kirkby, & Pfahringer, 2010). It is related to the WEKA project and bi-directional interaction of the two is possible. MOA itself does not support visual programming of workflows but the ADAMS project (Reutemann & Vanschoren, 2012) provides a workflow engine for MOA which uses a tree-like structure instead of an interactive canvas.

Sharing data and experiments has been implemented in the OpenML Experiment Database (Vanschoren & Blockeel, 2009), which is a database of standardized machine learning experimentation results. Instead of a workflow engine it features a visual query engine for querying the database, and an API for submitting experiments and data.

2.2. Active learning for data streams

There exist three different scenarios for active learning: (i) membership query synthesis, (ii) pool-based sampling, and (iii) stream-based selective sampling (Settles, 2010). In the membership query synthesis scenario, the learning algorithm can select examples for labeling from the input space or it can produce new examples itself. In the pool-based scenario, the learner has access to a collection of previously seen examples and may request labeling for any of them. In this study, we are interested in the third scenario: the stream-based active learning approach. More specific, we are interested in the active learning on stream data for sentiment analysis of Twitter posts. In this scenario, examples are constantly arriving from a data stream and the learning algorithm has to decide in real time whether to select an arriving example for labeling or not. Therefore, the approach which would handle this scenario has to:

- have constant access to a source of data,
- have the ability to quickly and in real time process each incoming instance and decide whether to request a label for it,
- periodically update the model and apply it to new instances.

[2] http://hadoop.apache.org/.
[3] http://mahout.apache.org/.
[4] http://www.radoop.eu.
[5] http://incubator.apache.org/s4/.
[6] http://storm-project.net/.

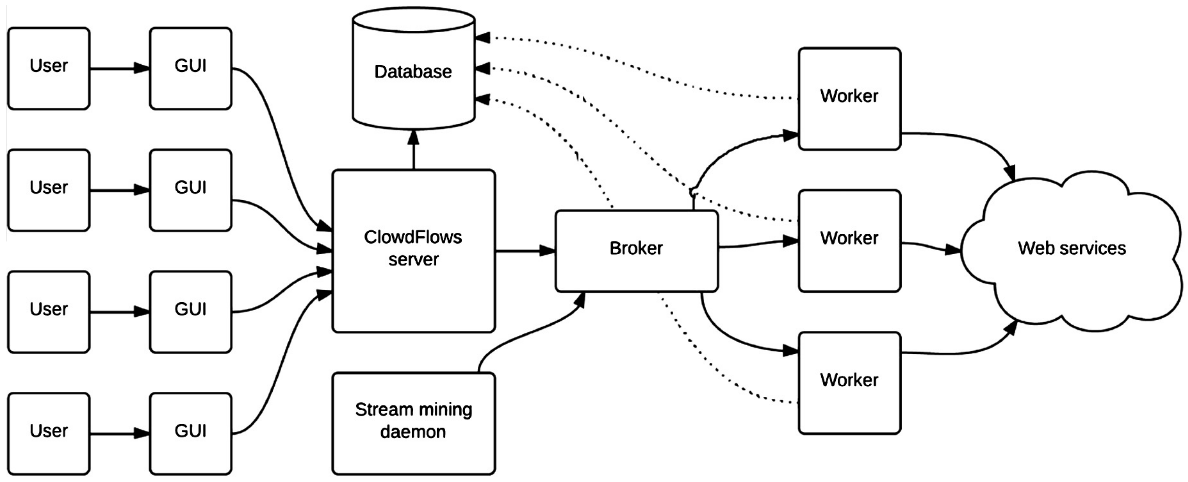


Fig. 1. An overview of the ClowdFlows platform design.

In the stream-based active learning setting, there exist several approaches to deciding whether or not to request hand labels for examples which come from a data stream. One of the simplest strategies is to use some informativeness measure and request labeling for the examples which are the most informative. For instance, the examples for which the learner has the highest uncertainty can be considered the most informative and be selected for labeling. Zhu, Zhang, Lin, and Shi (2007) used uncertainty sampling to label examples within a batch of data from the data stream. Žliobaitė, Bifet, Pfahringer, and Holmes (2011) propose strategies that are based on uncertainty, dynamic allocation of labeling efforts over time and randomization of the search space. Our active learning approach also employs randomization of the search space, but in contrast to the work of Žliobaitė et al. (2011), we organize the examples from the data stream into batches. The decision which examples are best for labeling can be made by a single evolving classifier (Žliobaitė et al., 2011) or by a classifier ensemble (Wang, Zhang, & Guo, 2012; Zhu et al., 2007, Zhu, Zhang, Lin, & Shi, 2010). In our study, we use a single evolving sentiment classifier for Twitter posts.

Our preliminary work on active learning on stream data for sentiment analysis of tweets is presented in (Saveski & Grčar, 2011). The closely related contribution was made in (Settles, 2011), where the author demonstrated the application of DUALIST, an active learning annotation tool, to Twitter sentiment analysis. The author intended to show the generality of the annotation tool, since it is not adjusted specifically to tweets. On the other hand, our approach is particularly adjusted to Twitter data. Regarding the on-line platform which would handle active learning on stream data for sentiment analysis of Twitter posts, to best of our knowledge, we are the first addressing this issue.

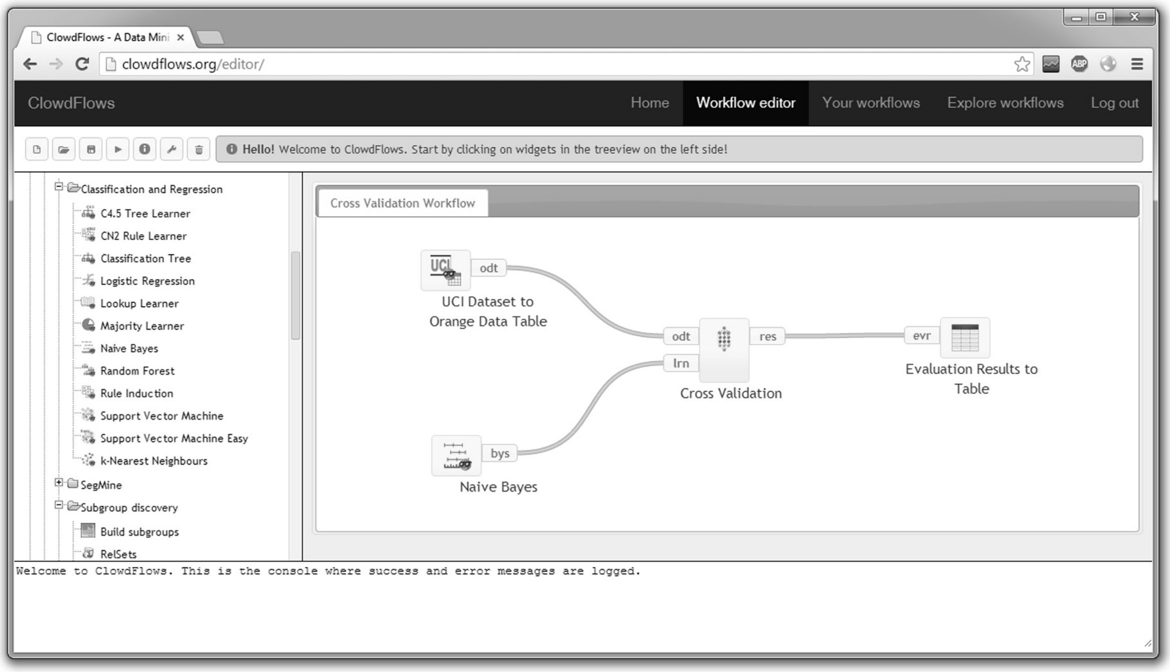
3. The ClowdFlows platform

In this section the ClowdFlows platform is presented. The enabling technologies are presented briefly and displayed in the architecture of the system. To validate the design of the platform we present a stress test with many simultaneous users executing their workflows. The graphical user interface and the workflow model are presented. Finally the real-time analysis features of ClowdFlows are described.

3.1. Platform design

As a new generation data mining platform, ClowdFlows (Kranjc et al., 2012a, Kranjc, Podpečan, & Lavrač, 2012b) is designed and implemented using modern technologies and computing paradigms. It is essentially a cloud-based Web application that can be accessed and controlled from anywhere while the processing is performed in a cloud of computing nodes. To achieve the goal of developing a platform that can be accessed and controlled from anywhere and executed on a cloud, we have designed it as a cloud-based Web application. As such it can be, based on the technologies used, logically separated on two sides – the client side, and the server side. The architecture of the platform accessed by multiple users is shown in Fig. 1. A similar architecture figure was previously published in (Kranjc et al., 2012a), with some major differences. In contrast to the previously published architecture, the platform now features a relational database for storing workflows, a broker for delegating tasks to worker nodes and a stream mining daemon for processing data streams.

The client side of the platform consists of operations that involve user interaction. The user interacts with the platform primarily through the graphical user interface in a contemporary Web browser. We have implemented the graphical user



Hello! Welcome to ClowdFlows. Start by clicking on widgets in the treeview on the left side!

```
Welcome to ClowdFlows. This is the console where success and error messages are logged.
```

Fig. 2. A screenshot of the ClowdFlows graphical user interface loaded in the Google Chrome Web browser.

interface in HTML and JavaScript, with an extensive use of the jQuery library.[7] The jQuery library was designed to simplify client-side scripting, and is the most popular JavaScript library in use today.[8]

The server side is written in Python and uses the Django Web framework.[9] Django is a high level Python Web framework that encourages rapid development and provides an object-relational mapper and a powerful template system. The object-relational mapper provides an API that links objects to a database, which means that the ClowdFlows platform is database agnostic. PostgreSQL, MySQL, SQLite and Oracle databases are all supported. MySQL is used in the public installation of ClowdFlows.

In order to allow consumption of Web services and importing them as workflow components, the PySimpleSoap library[10] is used. PySimpleSoap is a light-weight library written in Python and provides an interface for client and server Web service communication, which allows importing WSDL Web services as workflow components, and exposing entire workflows as WSDL Web services.

ClowdFlows may also be installed on multiple computers, which is enabled by using the RabbitMQ[11] messaging server and a Django implementation of Celery,[12] a distributed task queue, which allows passing asynchronous tasks between servers. With this tools it is possible to install ClowdFlows on worker nodes which execute workflows. To demonstrate the scalability of the platform with these tools we have performed a stress test which we describe in Section 3.2.

ClowdFlows is publically available for use at http://clowdflows.org. The source code is open sourced under the General Public Licence and can be downloaded at http://github.com/janezkranjc/clowdflows. Detailed installation instructions are provided with the source code.

3.2. Scalability of the platform

In order to test the scalability of the ClowdFlows platform and validate the design decisions described in Section 3.1 enabling big data analytics for data streams we have performed a stress test in which we simulated several users executing their workflows simultaneously and measured the average execution time.

The test was conducted on a simplified workflow that performs 10-fold cross validation with the Naive Bayes algorithm. The workflow is shown in Fig. 2. In order to simulate concurrent users we have implemented a simulation of a user that

[7] http://jquery.com.
[8] http://w3techs.com/technologies/overview/javascript_library/all.
[9] https://www.djangoproject.com.
[10] https://code.google.com/p/pysimplesoap/.
[11] http://www.rabbitmq.com/.
[12] http://celeryproject.org/.

Table 1
Average response time for the execution of the workflow based on different numbers of concurrent users and different setups of worker nodes. The cells display the average execution time in seconds (plus the standard deviation) and the number of workflow executions done by all the worker nodes in the time of the test in parantheses.

Users	Worker nodes			
	1×8	2×8	3×8	$3 \times 8 + 1 \times 16$
1	3.466 ± 0.603 (18)	3.252 ± 0.010 (19)	3.255 ± 0.011 (19)	3.248 ± 0.011 (19)
10	5.665 ± 2.927 (109)	3.913 ± 1.436 (157)	4.012 ± 1.399 (154)	3.476 ± 0.779 (179)
20	9.369 ± 5.449 (130)	5.530 ± 2.786 (222)	5.090 ± 4.189 (236)	4.268 ± 1.742 (290)
50	17.176 ± 8.105 (182)	9.856 ± 6.615 (303)	7.433 ± 4.207 (407)	6.154 ± 3.799 (495)
100	27.534 ± 13.071 (238)	18.088 ± 9.132 (351)	12.745 ± 6.732 (499)	9.882 ± 5.990 (617)

executes her workflow continuously for 60 s without a pause. This settings is also equivalent to executing a streaming process for 1 min. After 60 s have passed the user waits until the final workflow execution results are returned. We tested the platform against different sets of concurrent users: 1 user, 10 users, 20 users, 50 users, and 100 users for different setups of the worker nodes.

A worker node is a headless installation of the ClowdFlows platform that executes workflows. We tested the platform with a single worker node, two worker nodes, and three worker nodes. Each of these worker nodes was installed on equivalent computers with 8 cores. The workers were setup to work on 8 concurrent threads. For the final test we have setup an additional worker on a computer with 16 cores to run 16 threads. We have measured the execution times for each workflow and calculated the average execution time from the beginning of the request until the result was received. The results are shown in Table 1.

The results show that a single user continuously executing her cross validation workflow will be able to execute it 18 or 19 times on any setup if she is the only user executing workflows. In order for ten concurrent users to execute their workflows at a comparable speed at least two worker nodes are needed. The most efficient current setup is three workers with 8 threads each and one worker with 16 threads which still allows a hundred users to issue workflow execution requests to the platform at a reasonable response time.

The platform has succesfully passed the stress test. The results show that the ClowdFlows platform can serve many concurrent users that continuously execute workflows. The average execution times can be controlled by adding or removing worker nodes. The worker nodes can be added and removed during runtime, which means that heavy loads can be resolved simply by adding more computing power to the ClowdFlows worker cluster. As adding worker nodes at times with lower loads does not improve the average processing time, we would like to implement a mechanism for automatically spawning and removing worker nodes on services such as the Amazon Elastic Compute Cloud in future work.

3.3. The workflow model

The integral part of the ClowdFlows platform is the workflow model which consists of an abstract representation of workflows and workflow components. Workflows are executable graphical representations of complex procedures. A workflow in ClowdFlows is a set of processing components and connections. A processing component is a single workflow processing unit with inputs, outputs and parameters. Each component performs a task considering its inputs and parameters, and then stores the results of the task on its outputs. Connections are used to transfer data between two components and may exist only between an output of a widget and an input of another widget. Data is transffered between connections, so each input can only receive data from a connected output. Parameters are similar to inputs, but need to be entered manually by users. Inputs can be transformed into parameters and vice versa, depending on the users' needs.

3.4. The graphical user interface

The graphical user interface used for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The building blocks (workflow components) in ClowdFlows are referred to as widgets. The graphical user interface implements an easy to use way to arrange widgets on a canvas to form a graphical representation of a procedure. The ClowdFlows graphical user interface rendered in a Web browser is shown in Fig. 2.

The graphical user interface of the ClowdFlows system consists of a workflow canvas and a widget repository. The widget repository is a set of widgets ordered in a hierarchy of categories. Upon clicking on a widget in the repository, that widget appears on the canvas. The workflow canvas implements moving, connecting, issuing commands to execute and delete widgets. Widgets can be arbitrarily arranged on the canvas by dragging and dropping. Connections between widgets can be added by selecting an output of a widget and an input of another widget.

Information on each operation the user performs on the workflow canvas is sent to the server using an asynchronous HTTP POST request. The operation is validated on the server and a success or error message with additional information is passed to the user interface (the client's browser) formatted in JavaScript Object Notation (JSON) or HTML.

On the top of the graphical user interface is a toolbar where entire workflows can be saved, deleted, and executed.

3.5. The widget repository

Widgets in ClowdFlows are separated into four groups based on their purpose: regular widgets, visualization widgets, interactive widgets and workflow control widgets.

Regular widgets perform specific tasks that transform the data from the inputs and the parameters to data on the outputs, and provide success or error messages to the system. The task of a widget is written as a Python function that takes a Python dictionary of inputs and parameters as its arguments and returns a dictionary of outputs. The function is called each time the widget is executed. Widgets that implement complex procedures can also implement a progress bar, that displays progress to the user in real time.

Visualization widgets are extended versions of regular widgets as they also provide the ability to render an HTML template with JavaScript to the client's browser. These are useful for data visualizations and presentation of more detailed feedback to the user. Visualization widgets are regular widgets with the addition of a second Python function which controls the rendering of the template. This function is only invoked when the workflow is executed from the user interface.

An interactive widget is a widget that requires data before execution in order to prompt the user for the correct parameters. These widgets are extensions of regular widgets as they perform three functions. The data preparation function executes first and takes the inputs and parameters as the arguments. The second function is a rendering function where a modal window is prepared by using an HTML template which prompts the user to manipulate the data. The final function's arguments are the user's input and the inputs and parameters of the widget. A widget can also be a combination of an interactive and a visualization widget, where it executes a fourth rendering function to display the results.

Three special widgets provide additional workflow controls. These are the *Sub-workflow*, *Input*, and *Output* widget. Whenever a *Sub-workflow* widget is added to a workflow, an empty workflow is created that will be executed when the sub-workflow widget is executed. The *Sub-workflow* widget has no inputs and outputs by default, so they have to be added to the workflow by the user using the *Input* and the *Output* widget. Whenever an *Input* or *Output* widget is put on a workflow that is a sub-workflow of another workflow, an actual input or output is added to the widget representing the sub-workflow. Workflows can be indefinitely nested this way.

Two variations of the input and output widget provide ways to loop through sub-workflows. The input and output widgets can be replaced by the *For Input* and *For Output* widgets. Whenever a workflow contains these two widgets, the workflow execution engine will attempt to break down the object on the input and execute the workflow once for each piece of data that is on the input. With these controls a workflow can be executed on a list or array of data.

3.6. The workflow execution engine

The job of the workflow execution engine is to execute all executable widgets in the workflow in the correct order. The engine is implemented twice, both in Python and JavaScript due to performance issues when the user wishes to see the order of the executed widgets in real time.

The two implementations of the workflow execution engine are similar with two differences. The JavaScript engine is enabled by default due to the requests for executing separate widgets being asynchronous HTTP requests. Each request is handled by the server separately and executes a single widget, saves the changed and returns the results to the client where the execution continues. The server side Python implementation only receives one HTTP request for the entire workflow and multiprocessing had to be implemented manually. For performance issues, sub-workflows and loops are executed by the Python implementation, while top-level workflows executed from the user interface are processed by the JavaScript implementation. The JavaScript implementation shows the results of the execution of each widget in real time, while the user can only see the results of the Python implemented workflow execution after it has finished in full.

When a workflow is running, the execution engine perpetually checks for widgets that are executable and executes them. Executable widgets are widgets which either have no predecessors, or their predecessors have already been successfully executed. Whenever two or more widgets are executable at the same time they are asynchronously executed in parallel, since they are independent. The implemented widget state mechanism ensures that no two widgets where the inputs of a widget are dependent on an output of another widget will be executable at the same time. The execution of a workflow is complete when there are no executable or running widgets.

3.7. Public workflows

Since workflows in ClowdFlows are processed and stored on remote servers they can be accessed from anywhere with an internet connection. By default, each workflow can only be accessed by its author. We have implemented an option that allows users to create public versions of their workflows.

The ClowdFlows platform generates a specific URL for each workflow that has been saved as public. Users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by a user, a copy of the workflow is created on the fly and added to the user's private workflow repository. The workflow is copied with all the data to ensure the repeatability of experiments. Each such copied public workflow can also be edited, augmented or used as a template to create a new workflow, which can be made public as well.

3.8. Real-time data analysis in ClowdFlows

In comparison with the early implementations of the ClowdFlows platform described in (Kranjc et al., 2012a, 2012b) the novelty of this work is the ability of ClowdFlows to process real-time data streams. Its workflow engine has been augmented with continuous parallel execution and the halting mechanism and several specialized widgets for stream data processing were developed. In the following we describe the new data stream processing capabilities of the ClowdFlows platform.

3.8.1. Continuous workflow execution with the halting mechanism

Regular workflows and stream mining workflows are primarily distinguished by their execution times. A widget in a static workflow is executed a finite amount of times and the workflow has a finite execution time. Widgets in a stream mining workflow are executed a potentially infinite amount of times and the workflows are executed until manually terminated by users. Another major difference between regular workflows and stream mining workflows is the data on the input. The data that is processed by regular workflows is available in whole during the entire processing time, while data entering the stream mining workflows is potentially infinite and is only exposed as a small instance at any given time.

In order to handle potentially infinite data streams we have modified the workflow execution engine to execute the workflow multiple times at arbitrarily small temporal intervals in parallel. The amount of parallelism and the frequency of the execution are parameters that can be (providing the hardware availability) modified for each stream to maximize the throughput.

The execution of the workflows is delegated by a special *stream mining daemon* that issues tasks to the messaging queue. The stream mining daemon's task is to issue commands to execute streaming workflows. The daemon can also prioritize execution of some streams over others based on the users' preferences. Tasks are picked up from the messaging queue by workers that execute the workflow. To ensure that each execution of a workflow processes a different instance of the data, special widgets and mechanisms were developed, which can halt the execution of streaming workflows. This *halting* mechanism can be activated by widgets in a streaming workflow to halt the current execution.

Workflows that are executed as a stream mining process need to be saved as streaming workflows and executed separately. The user cannot inspect the execution of the workflow in real time, as many processes are running in parallel. The user can, however, see the results from special *stream visualization* widgets.

3.8.2. Specialized workflow widgets for real-time processing

Widgets in stream mining workflows have, in contrast to widgets in regular workflows, the internal memory and the ability to halt the execution of the current workflow. The internal memory is used to store information about the data stream, such as the timestamp of the last processed data instance, or an instance of the data itself. These two mechanisms were used to develop several specialized stream mining widgets.

In order to process data streams, *streaming data inputs* had to be implemented. Each type of stream requires its own widget to consume the stream. In principle, a streaming input widget connects to an external data stream source, collects instances of the data that it had not yet seen, and uses its internal memory to remember the current data instances. This can be done by saving small hashes of the data, to preserve space or just the timestamp of the latest instance if they are available in the stream itself. If the input widget encounters no new data instances at the stream source it halts the execution of the stream. No other widgets that are directly connected to it via its outputs will be executed until the workflow is executed again.

Several other popular stream mining approaches (Ikonomovska, Loskovska, & Gjorgjevik, 2007) were also implemented as workflow components. The *aggregation* widget was implemented to collect a fixed number of data instances before passing the data to the next widget. The internal memory of the widget is used to save the data instances until the threshold is reached. While the number of instances is below the threshold, the widget halts the execution. The internal memory is emptied and the data instances are passed to the next widget once the threshold has been reached.

The *sliding window* widget is similar to the aggregation widget, except that it does not empty its entire internal memory upon reaching the threshold. Only the oldest few instances are *forgotten* and the instances inside the sliding window are released to other widgets in the workflow for processing. By using the sliding window, each data instance can be processed more than once.

Sampling widgets are fairly simple. They either pass the instance to the next widget or halt the execution, based on an arbitrary condition. This condition can be dependent on the data or not (e.g. drop every second instance). The internal memory can be used to store counters, which are used to decide which data is left in the sample.

Special *stream visualization* widgets were also developed for the purpose of examining results of real-time analyses. Each instance of a stream visualization widget creates a special Web page with a unique URL that displays the results in various formats. This is useful because the results can be shared without having to share the actual workflows.

4. Active learning for sentiment analysis

In this section we first describe the dataset we use for the default tweet sentiment classifier, preprocessing techniques and the algorithm for sentiment analysis. The approach to tweet preprocessing and classifier training is implemented using the LATINO[13] software library of text processing and data mining algorithms. The section continues with a description of the active learning algorithm and the strategy used to select data instances for labeling.

4.1. The data used for the default sentiment classifier

The default tweet sentiment classifier is trained on a collection of 1,600,000 (800,000 positive and 800,000 negative) tweets collected and prepared by Stanford University (Go, Bhayani, & Huang, 2009), where the tweets were labeled based on positive and negative emoticons in them. Therefore, the emoticons approximate the actual positive and negative sentiment labels. This approach was introduced by Read (Read, 2005). If a tweet contains ":)", ":-)", ":)", ":D" or "=)" emoticon it was labeled as positive, and if it contains ":(", ":-(" or ": (" emoticon it was labeled as negative. In the training data, the tweets containing both positive and negative emoticons, retweets and duplicate tweets were removed (Go et al., 2009). The emoticons, which approximate sentiment labels, were also already removed from the tweets in order not to put too much weight on them in the training phase, and therefore the classifier learns from the other features of tweets. The tweets in this collection do not belong to any particular domain.

4.2. Data preprocessing

Preprocessing of data is an important step when using supervised machine learning techniques. On the Twitter data, we apply both standard and Twitter-specific text preprocessing to better define the feature space. The specific text preprocessing is especially important for Twitter messages, since user generated content on the Internet often contains slang (Petz et al., 2012) and messages from social media are considered noisy, containing many grammatical and spelling mistakes (Petz et al., 2013). Therefore, with our Twitter-preprocessing, we try to overcome these problems and improve the quality of features.

As a part of the Twitter preprocessing step (Agarwal, Xie, Vovsha, Rambow, & Passonneau, 2011; Go et al., 2009; Smailović, Grčar, Lavrač, & Žnidaršič, 2013; Smailović, Grčar, & Žnidaršič, 2012) we replace mentioning of other Twitter users in a tweet of the form @TwitterUser by a single token named *USERNAME* and writing different Web links by a single token named *URL*. Moreover, letters which repeat for more than two times are replaced by one occurrence of such letter; for example, the word *looooooooove* is transformed to *love*. We replace negation words (*not*, *isn't*, *aren't*, *wasn't*, *weren't*, *hasn't*, *haven't*, *hadn't*, *doesn't*, *don't*, *didn't*) with a single token named *NEGATION*. Finally, exclamation marks are replaced by a single token *EXCLAMATION* and question marks by a single token *QUESTION*.

Besides the Twitter-specific text preprocessing, we also apply standard preprocessing techniques (Feldman & Sanger, 2007) in order to better define and reduce the feature space. These involve text tokenization (text splitting into individual words/terms), stopwords removal (removing words which do not contain relevant information, e.g., *a*, *an*, *the*, *and*, *but*, *if*, *or*, etc.), stemming (converting words into their base or root form) and N-gram construction (concatenating 1 to N stemmed words appearing consecutively in a tweet). The resulting terms are used as features in the construction of feature vectors representing the tweets, where the feature vector construction is based on term frequency feature weighting scheme. We do not apply a part of speech (POS) tagger, since it was indicated by Go et al. (2009) and Pang and Lee (2002) that POS tags are not useful when using SVMs for sentiment analysis. Also, Kouloumpis, Wilson, and Moore (2011) showed that POS features may not be useful for sentiment analysis in the microblogging domain.

4.3. The algorithm used for sentiment classification

Sentiment analysis methods (Liu, 2012; Pang & Lee, 2008; Turney, 2002) aim at detecting the authors attitude, emotions or opinion about a given topic expressed in text. There are three generally known approaches to sentiment analysis (Pang & Lee, 2008; Thelwall, Buckley, & Paltoglou, 2011): (i) machine learning, (ii) lexicon-based methods and (iii) linguistic analysis.

We use a machine learning approach, applying the linear Support Vector Machine (SVM) (Cortes & Vapnik, 1995; Vapnik, 1995, 1998), which is a typical algorithm used in document classification. The SVM training algorithm represents the labeled training examples as points in the space and separates them with a hyperplane. A hyperplane is placed in such a way that the examples of the separate classes are divided from each other as much as possible. New examples are then mapped into the same space and classified based on the side of the hyperplane they are. For training the tweet sentiment classifier, we use the SVM^{perf} (Joachims, 2005, 2006; Joachims & Yu, 2009) implementation of the SVM algorithm. In order to test its classification accuracy, we trained the SVM classiffier on the collection of 1,600,000 smiley labeled tweets (Go et al., 2009) and tested it on 177 negative and 182 positive manually labeled tweets, prepared and labeled by Stanford University

[13] LATINO (Link Analysis and Text Mining Toolbox) is open-source—mostly under the LGPL license—and is available at http://latino.sourceforge.net/.

Table 2
Average accuracy, precision and recall in the setting without active learning and with active learning while experimenting with different proportions of random tweets (#rnd) and tweets which are closest to the SVM hyperplane (#hyp) from every batch, containing 1000 tweets, for hand labeling.

Setting	#rnd	#hyp	Accuracy	Precision positive class	Recall positive class	Precision negative class	Recall negative class
No active learning	0	0	0.349	0.463	0.569	0.223	0.643
Active learning	0	100	0.406	0.456	0.829	0.272	0.351
Active learning	25	75	0.413	0.454	0.858	0.275	0.296
Active learning	50	50	0.410	0.459	0.837	0.256	0.335
Active learning	75	25	0.418	0.451	0.881	0.279	0.265
Active learning	100	0	0.416	0.448	0.886	0.288	0.251

(Go et al., 2009). We applied both standard and Twitter specific preprocessing. In this experiment we achieved the accuracy of 83.01% (which is a comparable result with (Go et al., 2009)).

The reason for using a machine learning approach and not lexicon-based or linguistic methods is the following. In the context of active learning for sentiment analysis on data streams, the linguistic methods pose several challenges, as they tend to be too computationally demanding for the use in a streaming near real time setting. Also, there is the lack of readily available tools for parsing tweets. On the other hand, lexicon-based methods are faster, but they usually rely on explicit notion of sentiment and dismiss the terminology that bears sentiment more implicitly. For example, the word 'Greece' bears negative sentiment in the light of the financial crisis, but in general it is neutrally or even positively connoted word.

Nevertheless, in order to compare lexicon and machine learning methods, we have tested a lexicon method classification accuracy on the same collection of 177 negative and 182 positive manually labeled tweets (Go et al., 2009), as for the machine learning approach. In the lexicon-based method, we used an opinion lexicon containing 2006 positive and 4783 negative words[14] (Hu & Liu, 2004; Liu, Hu, & Cheng, 2005). The lexicon is adjusted to social media content, as it also contains many misspelled words which are frequently used in social media language. We applied Twitter specific preprocessing on the test tweets and calculated positive and negative score for each tweet, based on the occurrences of positive and negative lexicon words in them. For example, if a tweet contains a word 'love' from the positive lexicon list, the positive score will increase by one. The score will not increase if the currently observed lexicon word contains or it is contained in some of the previously seen lexicon words for that specific class in the observed tweet. For example, a tweet could contain a word 'nicely'. On the other hand, the positive word lexicon list contains both 'nice' and 'nicely' words. The algorithm will detect that the word 'nice' is presented in the tweet and it will increase the positive score by one. Next, it will check the presence of the word 'nicely' and it will find out that the tweet contains this word, but this word contains a word ('nice'), which already increased the positive score for this tweet, and therefore it will not increase the positive score. If the resulting positive score for a tweet is the same or higher than the negative score, the tweet is labeled as positive. If it is lower, it is labeled as negative. The tweets with equal positive and negative score are labeled as positive, since the positive lexicon list contains less words. In this experiment we achieved the accuracy of 76.04% on the test set.

Since the accuracy on the test set obtained with the machine learning approach was higher than the accuracy obtained with the lexicon-based approach, we decided to focus on the machine learning approach in our study.

4.4. Active learning

In active learning, the learning algorithm periodically asks an oracle (e.g., a human annotator) to manually label the examples which he finds most suitable for labeling. Using this approach and an appropriate query strategy, the number of examples that need to be manually labeled is largely decreased. Typically, the active learning algorithm first learns from an initially labeled collection of examples. Based on the initial model and the characteristics of the newly observed unlabeled examples, the algorithm selects new examples for manual labeling. After the labeling is finished, the model is updated and the process is repeated for the new incoming examples. This procedure is repeated until some threshold (for example, time limit, labeling quota or target performance) is reached or, in the case of data streams, it continues as long as the application is active and new examples are arriving.

In our software, the active learning algorithm first learns from the Stanford smiley labeled data set as an initial labeled data set. According to this initial model, the algorithm classifies new incoming tweets from the data stream as positive or negative. Tweets, which come from the data stream, are split into batches. The algorithm selects most suitable tweets from a first batch for hand-labeling and puts them in a pool of query tweets. The process is repeated for every following batch and every time the pool of query tweets is updated and the tweets in the pool are reordered according to how suitable they are for hand-labeling. When the user decides to conduct manual labeling, she is given a selected number of top tweets from the pool of query tweets for hand-labeling. The user can label a tweet as positive, negative or neutral. After the labeling, labeled tweets are placed in the pool of labeled tweets and removed from the pool of query tweets. Periodically, using the initial and

[14] The opinion lexicon was obtained from http://www.cs.uic.edu/liub/FBS/sentiment-analysis.html.

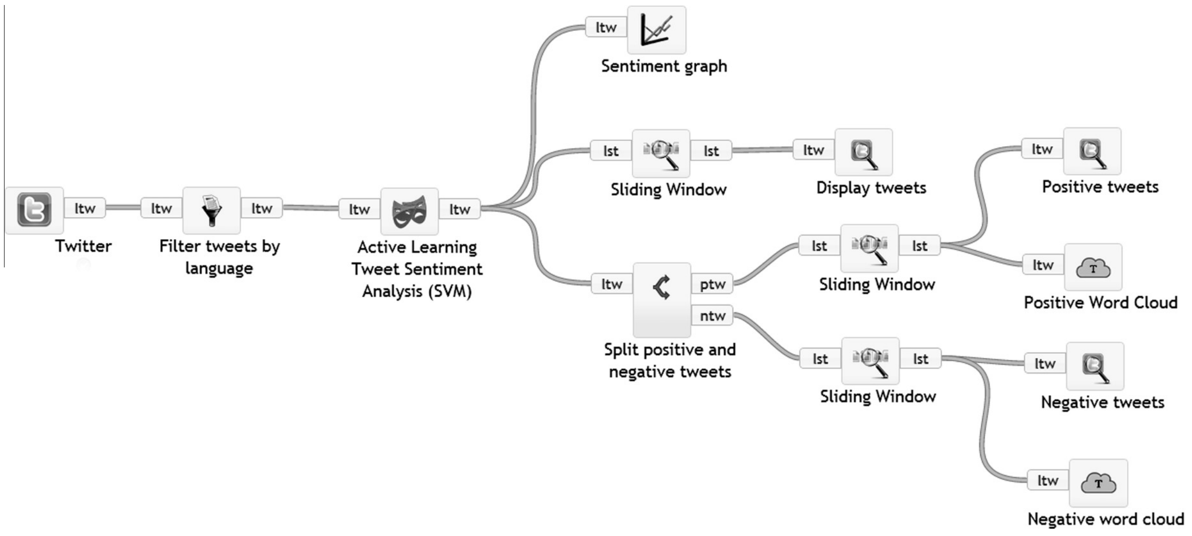


Fig. 3. The Twitter sentiment analysis workflow.

manually positively and negatively labeled tweets from the pool of labeled tweets, the model is retrained. This process is repeated until it is terminated by the user.

The selection of tweets, which are suitable for manual labeling is based on uncertainty strategy and randomization of the search space. The randomization of the search space was also used by Žliobaitė et al. (2011). We experimented with different proportions of random tweets and tweets which are closest to the SVM hyperplane in order to find the best combination of them. Additionally, we performed one experiment in which we did not apply the active learning strategy, i.e., the sentiment classifier was static and did not update over time. In order to automatically conduct these experiments, we hand-labeled a data set of 11,389 financial tweets (4861 positive, 1856 negative and 4672 neutral tweets) discussing the Web search engine provider Baidu,[15] which were collected for a period from March 11 to December 9, 2011. The evaluation method was based on a holdout evaluation approach (Bifet & Kirkby, 2009; Ikonomovska, 2012; Ikonomovska, Gama, & Džeroski, 2011) for data streams where concept drift is assumed. The classifier‘s performance is tested on a new batch of tweets which come from the data stream. After the testing is finished, the algorithm selects a predefined number of tweets from the same batch and asks an oracle to label them. The newly labeled tweets are added to the training set and used for updating the sentiment model. This procedure is repeated for every new batch of tweets from the data stream. We calculate the accuracy, precision and recall for every batch and at the end of the simulation we report the overall average measures for all the batches. In our off-line evaluation experiments, we select 100 tweets from every batch, which contains 1000 tweets, and then update the model. The results are presented in Table 2. As can be seen from the table, the accuracy of the sentiment classification is higher when the active learning approach is applied. Among the querying strategies, the highest accuracy is obtained by selecting 75 random tweets and 25 tweets which are closest to the SVM hyperplane.

In contrast to the experimental setting described above, the workflow developed for practical use (shown in Fig. 3) by default splits tweets from the data stream into batches which contain 100 tweets, and selects 10 for hand labeling. Following the best strategy from Table 2, the algorithm selects 3 tweets that are closest to the SVM hyperplane and puts them into the pool of query tweets, so that the top most are the ones which are closest to the hyperplane, i.e., the most uncertain ones for the classifier. The other 7 tweets are chosen randomly from the batch and put into a separate pool of random tweets. With time, as new tweets arrive, the pools are updated. Whenever the user decides to label some tweets, she is presented with a set of tweets to label, which contains 3 most uncertain ones from the pool of query tweets and 7 random ones from the pool of random tweets. The hand-labeled tweets are placed in the pool of labeled tweets. Periodically, using the initial and manually labeled tweets from the pool of labeled tweets, the model is retrained.

5. Active learning sentiment analysis workflow implementation in ClowdFlows

In this section we present the implementation of an active learning sentiment analysis use case on Twitter data in the form of an executable workflow. The use case description is written as a step-by-step report on how the workflow was constructed. Following the description in this section, it is possible for the reader to construct a fully functioning streaming active learning sentiment analysis process and observe its results.

[15] http://www.baidu.com/.

The aim of the use case is to monitor the Twitter sentiment on a given subject with the possibility to manually label tweets to improve the classification model. For the purpose of this use case we have selected to monitor tweets containing the keyword *Snowden*, as it is one of the trending keywords during the time of writing this article. We wish to measure the Twitter sentiment over time regarding Edward Snowden, who leaked details of several top-secret documents to the press.

5.1. Rationale

We have decided to implement this stream-mining workflow in the ClowdFlows platform for several reasons.

The execution of the stream-mining workflow is bottlenecked by the rate of incoming Tweets, which is imposed by the Twitter API. Therefore any stream mining platform capable of processing tweets at a higher rate than the API's incoming rate would be as efficient as ClowdFlows for this use case. However, the benefit of using ClowdFlows for this task is its extensible user interface which allows for human–computer interaction during the course of the stream mining process. In this use case the user interface is used during runtime for labeling Tweets. The user interface can also be used to modify the workflow by using the intuitive visual programming paradigm interface. Moreover, the ability to share workflows allows us to publish this workflow on the Web and allow single click deployment of it to the users. The users can also augment, extend or modify the workflow to suit their needs without any coding knowledge just by rearranging the workflow components on the canvas.

5.2. Development of necessary components

To construct the workflow we required a *stream input* widget that can collect tweets based on a query, a *sampling* widget that should discard any non-English tweets, a widget to perform sentiment analysis on tweets, a *stream splitter* to split the stream of tweets into a stream of positive and a stream of negative tweets, and three types of visualization widgets to display the line chart of the sentiment over time, a word cloud of positive or negative tweets, and the latest tweets.

5.2.1. Streaming input, filtering, and visualizations

To consume the incoming stream we implemented a widget that connects to Twitter via the Twitter API.[16] The widget accepts several parameters: the search query, by which it filters the incoming tweets, the geographical location (optional), which filters tweets based on location, and the credentials for the Twitter API. The widget works both in a streaming and non-streaming environment. Whenever the widget is executed it will fetch the latest results of the search query. For streaming workflows, the internal memory of the widget holds the ID of the latest tweet, which is passed to the Twitter API, so that only the tweets that have not yet been seen are fetched.

Since tweets returned by the Twitter API are annotated with their language, we constructed a widget for filtering tweets based on their language. This widget discards all tweets that are not in English.

A simple widget was implemented that splits the stream of tweets into two streams, based on their sentiment. This was done so that positive and negative tweets could be separately inspected.

To visualize the sentiment we implemented a line chart that displays the volume of all tweets, the volume of positive tweets, the volume of negative tweets, and the difference of positive and negative tweets. The visualization was implemented with the HighCharts JavaScript visualization library.[17]

To inspect separate tweets a simple table was implemented where each tweet is colored red or green based on its sentiment (red for negative and green for positive).

The word cloud visualization was implemented to show most popular words in recent tweets. This visualization is dynamic and changes with the stream. Looking at the word cloud and seeing popular words appearing and unpopular words disappearing is a novel way to inspect data streams in real-time. The visualization was developed with the D3.js JavaScript library (Bostock, Ogievetsky, & Heer, 2011).

5.2.2. Sentiment classification and active learning

To implement sentiment classification and active learning discussed in Section 4, which was developed in the.NET framework, we exposed it as a Web service that provided several operations:

- classify a set of tweets for a specific workflow,
- return a set of tweets for manual labeling for a specific workflow,
- update a model for a specific workflow.

The service keeps track of multiple workflows and builds a model for workflows separately (in order to better conform the models for specific topics and to avoid malicious labeling affecting the models for legitimate users). Whenever the service is queried a unique identifier of the processing component is also passed along to determine which model to use.

[16] https://dev.twitter.com/.
[17] http://www.highcharts.com/.

The *Classify a set of tweets* operation accepts a set of tweets and an identifier of the processing component at the input. Upon execution it loads the appropriate model and applies it to the tweets. The loading times of the models were reduced to become shorter than the waiting time required to conform to the rate limit of the Twitter API in order to guarantee the processing of all the tweets. The operation returns a set of labels for the tweets.

Return a set of tweets for manual labeling is an operation that accepts the unique identifier of the processing component and returns ten tweets for manual labeling for that specific model. The tweets are then deleted from its pool of query tweets.

The *Update model* operation accepts a set of labeled tweets and a unique identifier of the processing component to update the model. The updating of a model takes several minutes so special care was taken in order to only update models when really necessary.

The functions were implemented into a workflow processing component in the following way: we have developed a streaming workflow component that receives a list of tweets at the input. These tweets are provided by the Twitter API usually in a batch of a hundred or less tweets. The tweets are sent to the *Classify a set of tweets* operation of the Web service. The sentiment labels that are returned from the Web service are appended to the tweets which are sent to the visualization widgets in the workflow. The active learning workflow component also has a special view that functions as an interactive visualization. This view is accessible the same way as other visualizations of the workflow (by special URLs). Whenever this view is accessed the Web service is polled for tweets that require manual labeling. If there are no query tweets in the pool, a friendly message is displayed to the user, prompting her to come back later. If there are query tweets in the pool they are displayed to the user along with a simple form that can be used to manually label the tweets either as positive, negative, or neutral. When the user labels the tweets and clicks the *Submit labels* button, the labeled tweets are saved into the internal memory of the active learning sentiment analysis component. The *Update model* operation is invoked once a day for every streaming workflow that has an active learning widget with new labeled tweets.

5.3. Constructing the workflow

The workflow was constructed using the ClowdFlows graphical user interface. Widgets were selected from the widget repository and added to the canvas and connected as shown in Fig. 3.

Parameters were set after the workflow was constructed. Parameters of a widget are set by double clicking the widget. Twitter API credentials and the search query were entered as parameters for the *Twitter* widget. The language code *en* was entered as a parameter of the *Filter tweets by language* widget. We have also added three *sliding window* widgets with the size 500 (entered as parameter) to the workflow. This is done because the visualization widgets that display tweets and word clouds only display the last data that was received as an input for these widgets. By setting the size of the window to 500 the word cloud will always consist of the words of most recent 500 tweets.

The workflow was saved by clicking the save button in the toolbar. We have also marked the workflow as public so that the workflow can be viewed and copied by other people. The URL of the workflow is http://clowdflows.org/workflow/1041/. We have then navigated to the workflows page (http://clowdflows.org/your-workflows/) and clicked the button "Start stream mining" next to our saved workflow. By doing this we have instructed the platform to start executing the workflow with the stream mining daemon. A special Web page was created where detailed information about the stream mining process is displayed. This page also contains links to visualization pages that were generated by the widgets. The stream mining process was left running from the 14th of June until the 10th of July 2013.

5.4. Monitoring the results

We have put several stream visualization widgets in the workflow which allowed us to inspect the results during the process of stream mining. ClowdFlows has generated a Web page for each stream visualization widget, which can be viewed by anybody since the workflow is public.

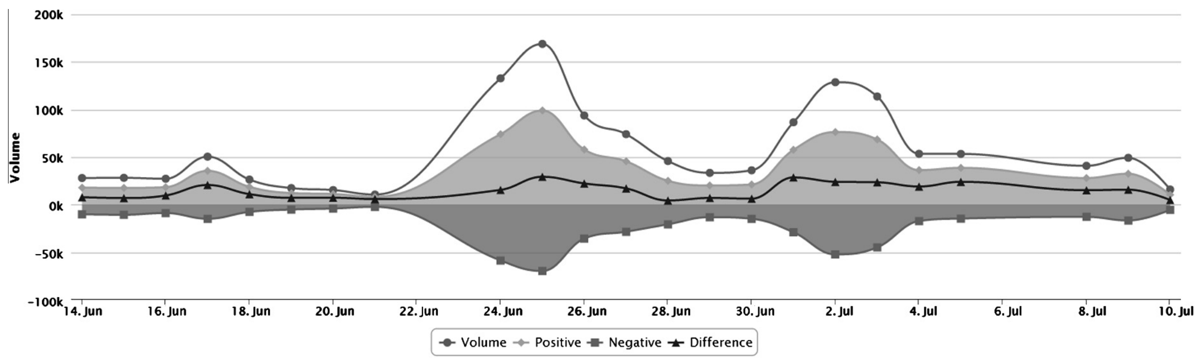


Fig. 4. A line chart of sentiment, volume, and sentiment difference over time.

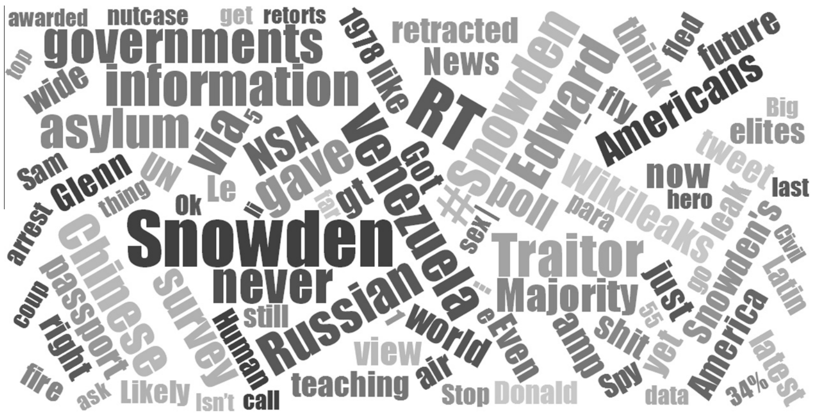
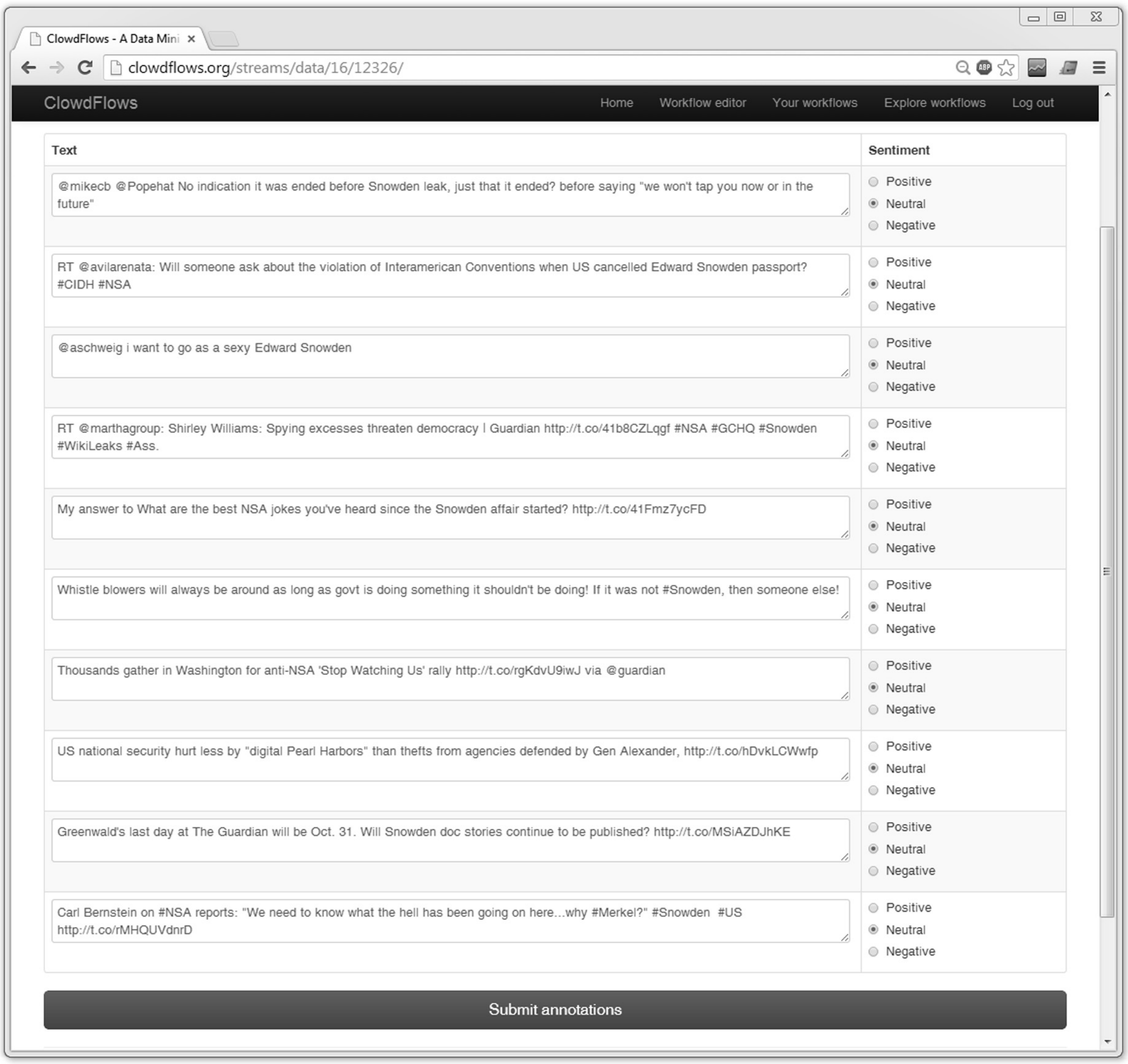


Fig. 5. A word cloud constructed from tweets with a negative sentiment.



Text	Sentiment
@mikecb @Popehat No indication it was ended before Snowden leak, just that it ended? before saying "we won't tap you now or in the future"	Positive [✓ Neutral] Negative
RT @avilarenata: Will someone ask about the violation of Interamerican Conventions when US cancelled Edward Snowden passport? #CIDH #NSA	Positive [✓ Neutral] Negative
@aschweig i want to go as a sexy Edward Snowden	Positive [✓ Neutral] Negative
RT @marthagroup: Shirley Williams: Spying excesses threaten democracy | Guardian http://t.co/41b8CZLqgf #NSA #GCHQ #Snowden #WikiLeaks #Ass.	Positive [✓ Neutral] Negative
My answer to What are the best NSA jokes you've heard since the Snowden affair started? http://t.co/41Fmz7ycFD	Positive [✓ Neutral] Negative
Whistle blowers will always be around as long as govt is doing something it shouldn't be doing! If it was not #Snowden, then someone else!	Positive [✓ Neutral] Negative
Thousands gather in Washington for anti-NSA 'Stop Watching Us' rally http://t.co/rgKdvU9iwJ via @guardian	Positive [✓ Neutral] Negative
US national security hurt less by "digital Pearl Harbors" than thefts from agencies defended by Gen Alexander, http://t.co/hDvkLCWwfp	Positive [✓ Neutral] Negative
Greenwald's last day at The Guardian will be Oct. 31. Will Snowden doc stories continue to be published? http://t.co/MSiAZDJhKE	Positive [✓ Neutral] Negative
Carl Bernstein on #NSA reports: "We need to know what the hell has been going on here...why #Merkel?" #Snowden #US http://t.co/rMHQUVdnrD	Positive [✓ Neutral] Negative

Fig. 6. The tweet labeling user interface.

The *Sentiment graph* visualization displaying the line chart of volumes of tweets, volumes of positive tweets, negative volume of negative tweets and the difference of positive and negative sentiment is available at http://clowdflows.org/streams/data/4/9056/ and is shown in Fig. 4. By looking at this visualization we can see that the sentiment in the tweets mentioning Snowden is generally more positive than negative. We can observe several spikes in the volume which correspond to the times when news articles regarding this subject were published. On June 23rd, news of Edward Snowden's departure from Hong Kong and arrival in Moscow was published. On the first of July Edward Snowden released a statement on the Wikileaks website and lots of news reports focused on possible countries that could offer asylum to Edward Snowden.

The word cloud visualization of negative tweets is available at http://clowdflows.org/streams/data/4/9065/ and is shown in Fig. 5. This visualization helps put the stream into another perspective and can display changing trends in real-time. When the word cloud is opened in the browser and the stream mining process is active the words change positions and sizes corresponding to their occurrences in the tweets. Links to the visualizations of other stream visualization widgets are also present on the two provided visualization pages.

The workflow presented in this use case is general and reusable. The query chosen for monitoring was arbitrary and can be trivially changed. This type of workflow could also be used for monitoring sentiment on other subjects, such as monitoring the Twitter sentiment of political candidates during an election, or monitoring the sentiment of financial tweets with stock symbols as queries.

5.5. Labeling the tweets

Similar to stream visualization widgets the *Active learning sentiment analysis* widget provides a special URL that can be accessed by human annotators. Propagating this link is an easy way to crowdsource labeling of tweets.

The labeling interface for this use case is available at http://clowdflows.org/streams/data/16/12326/ and is shown in Fig. 6. The tweets that require labeling are displayed and users can label them as positive, negative, or neutral. Upon clicking the button *Submit annotations* the labeled tweets are saved into the widget's internal memory. These tweets are accessed and sent to the sentiment analysis Web service once a day, if there are any new labeled tweets on that particular day.

6. Conclusion and further work

We have implemented an active learning scenario for sentiment analysis on Twitter data in a cloud-based data mining platform. In order to do so we adapted the platform to work with data streams by use of two mechanisms: widget memory and the halting mechanism.

We have developed a Web service that utilizes the Support Vector Machine algorithm to build and update sentiment analysis models. The service also applies the models on unlabeled tweets and determines which tweets require manual labeling by the user. We have developed workflow components that utilize this Web service in order to provide an intuitive interface for labeling tweets and setting up new active learning sentiment analysis scenarios from scratch without the need of programming or installing complex software. For each active learning workflow a special Web page is created where tweets can be labeled. By propagating the address of this Web page, crowdsourcing and collaborative knowledge discovery can be utilized to label vast amounts of tweets.

In future work we wish to implement several different strategies for selecting the tweets suitable for labeling and to allow the user to select the most appropriate one. We also wish to allow more control over the generation of the initial models and a richer selection of initially labeled datasets. In the current version of our software, we assume sentiment analysis to be a two class classification problem and classify tweets only as positive or negative, in order to enable simple and efficient calculations in real time. But, tweets can also be neutral, and our current implementation of the software does not allow 3 class classification. In our previous study (Smailović et al., 2013) we introduced a method to classify tweets also as neutral. In future work we plan to adapt and implement this method for inclusion in ClowdFlows.

The source code of the platform is released under an open source licence (GPL) and can be obtained at http://github.com/janezkranjc/clowdflows.

Acknowledgements

This work was supported in part by the European Commission FP7 FET-Open project FOC (Forecasting financial crises, Grant No. 255987), the FP7 European Commission project MUSE (Machine understanding for interactive storytelling, grant agreement no: 296703), the project ConCreTe, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET Grant No. 611733, and by the Slovenian Research Agency through the research program Knowledge Technologies under Grant P2-0103. The research was also supported by Ad Futura Programme of the Slovenian Human Resources and Scholarship Fund.

References

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the workshop on languages in social media* (pp. 30–38). Association for Computational Linguistics.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., & Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th international conference on scientific and statistical database management* (pp. 423–424). New York: IEEE Computer Society.

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., et al (2007). KNIME: The Konstanz information miner. In C. Preisach, H. Burkhardt, L. Schmidt-Thieme, & R. Decker (Eds.), *GfKl, studies in classification, data analysis, and knowledge organization* (pp. 319–326). Springer.

Bifet, A., & Kirkby, R. (2009). Data stream mining: A practical approach, Citeseer.

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604.

Blankenberg, D., Kuster, G. V., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., et al (2001). Galaxy: A web-based genome analysis tool for experimentalists. *Current Protocols in Molecular Biology*.

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics, 17*, 2301–2309.

Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G. R., Ng, A. Y., et al (2006). Map-reduce for machine learning on multicore. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Proceedings of NIPS* (pp. 281–288). MIT Press.

Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010). Mapreduce online. In *Proceedings of the 7th USENIX conference on networked systems design and implementation, NSDI'10* (pp. 21–34). Berkeley, CA, USA: USENIX Association.

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

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*, 107–113.

Decker, G., Overdick, H., & Weske, M. (2008). Oryx – An open modeling platform for the bpm community. In M. Dumas, M. Reichert, & M.-C. Shan (Eds.), *Business process management* (pp. 382–385). Springer.

Demšar, J., Zupan, B., Leban, G., & Curk, T. (2004). Orange: From experimental machine learning to interactive data mining. In J.-F. Boulicaut, F. Esposito, F. Giannotti, & D. Pedreschi (Eds.), *Proceedings of PKDD* (pp. 537–539). Springer.

Feldman, R., & Sanger, J. (2007). *The text mining handbook: Advanced approaches in analyzing unstructured data*. Cambridge University Press.

Gama, J., Rodrigues, P. P., Spinosa, E. J., & de Carvalho, A. C. P. L. F. (2010). Knowledge discovery from data streams, Citeseer.

Go, A., Bhayani, R., & Huang, L. (2009). *Twitter sentiment classification using distant supervision*. CS224N Project Report, Stanford (pp. 1–12).

Hu, M., & Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 168–177). ACM.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C. A., Pocock, M. R., Li, P., et al (2006). Taverna: A tool for building and running workflows of services. *Nucleic Acids Research, 34*, 729–732.

Ikonomovska, E. (2012). *Algorithms for learning regression trees and ensembles on evolving data streams*. Ph.D. thesis, Jožef Stefan International Postgraduate School.

Ikonomovska, E., Gama, J., & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery, 23*, 128–168.

Ikonomovska, E., Loskovska, S., & Gjorgjevik, D. (2007). A survey of stream data mining. In *Proceedings of 8th national conference with international participation* (pp. 19–21). ETAI.

Joachims, T. (2005). A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on machine learning* (pp. 377–384). ACM.

Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 217–226). ACM.

Joachims, T., & Yu, C.-N. J. (2009). Sparse kernel svms via cutting-plane training. *Machine Learning, 76*, 179–193.

Kouloumpis, E., Wilson, T., & Moore, J. (2011). Twitter sentiment analysis: The good the bad and the omg! In *Proceedings of fifth international AAAI conference on weblogs and social media* (pp. 538–541). ICWSM.

Kranjc, J., Podpečan, V., & Lavrač, N. (2012a). Knowledge discovery using a service oriented web application. In *Proceedings of the fourth international conference on information, process, and knowledge management* (pp. 82–87). eKNOW.

Kranjc, J., Podpečan, V., & Lavrač, N. (2012b). Clowdflows: A cloud based scientific workflow platform. In P. A. Flach, T. D. Bie, & N. Cristianini (Eds.), *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD (2)* (pp. 816–819). Springer.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies, 5*, 1–167.

Liu, B., Hu, M., & Cheng, J. (2005). Opinion observer: Analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web* (pp. 342–351). ACM.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In L. Ungar, M. Craven, D. Gunopulos, & T. Eliassi-Rad (Eds.), *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 935–940). KDD.

Morales, G. D. F. (2013). SAMOA: A platform for mining big data streams. In L. Carr, A. H. F. Laender, B. F. Lóscio, I. King, M. Fontoura, & D. Vrandecic, et al. (Eds.), *WWW (companion volume)* (pp. 777–778). ACM.

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval, 2*, 1–135.

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on empirical methods in natural language processing* (pp. 79–86). Association for Computational Linguistics.

Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Stříteský, V., Holzinger, A., et al (2013). In *Proceedings of human–computer interaction and knowledge discovery in complex, unstructured, big data* (pp. 35–46). Springer.

Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Winkler, S. M., Schaller, S., et al (2012). On text preprocessing for opinion mining outside of laboratory environments. In *Active media technology* (pp. 618–629). Springer.

Podpečan, V., Zemenova, M., & Lavrač, N. (2012). Orange4WS environment for service-oriented data mining. *The Computer Journal, 55*, 89–98.

Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL student research workshop* (pp. 43–48). Association for Computational Linguistics.

Reutemann, P., & Vanschoren, J. (2012). Scientific workflow management with ADAMS. In P. A. Flach, T. D. Bie, & N. Cristianini (Eds.), *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD (2)* (pp. 833–837). Springer.

Saveski, M., & Grčar, M. (2011). Web services for stream mining: A stream-based active learning use case, machine learning and knowledge discovery in databases. *ECML PKDD, 2011*, 36.

Sculley, D. (2007). Online active learning methods for fast label-efficient spam filtering. In *Proceedings of the fourth conference on email and anti-spam*. CEAS.

Settles, B. (2010). *Active learning literature survey*. Madison: University of Wisconsin.

Settles, B. (2011). From theories to queries: Active learning in practice. *Active Learning and Experimental Design, W*, 1–18.

Settles, B. (2011). Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1467–1478). Association for Computational Linguistics.

Settles, B., & Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1070–1079). Association for Computational Linguistics.

Smailović, J., Grčar, M., & Žnidaršič, M. (2012). Sentiment analysis on tweets in a financial domain. In *Proceedings of the fourth Jožef Stefan international postgraduate school students conference* (pp. 169–175). Jožef Stefan International Postgraduate School.

Smailović, J., Grčar, M., Lavrač, N., & Žnidaršič, M. (2013). Predictive sentiment analysis of tweets: A stock market application. In *Proceedings of human–computer interaction and knowledge discovery in complex, unstructured, big data* (pp. 77–88). Springer.

Talia, D., Trunfio, P., & Verta, O. (2005). Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids. In A. Jorge, L. Torgo, P. Brazdil, R. Camacho, & J. Gama (Eds.), *Proceedings of PKDD* (pp. 309–320). Springer.

Taylor, I., Shields, M., Wang, I., & Harrison, A. (2007). The Triana workflow environment: Architecture and applications. *Workflows for e-Science, 1*, 320–339.

Thelwall, M., Buckley, K., & Paltoglou, G. (2011). Sentiment in twitter events. *Journal of the American Society for Information Science and Technology, 62*, 406–418.
Turney, P. D. (2002). Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 417–424). Association for Computational Linguistics.
Vanschoren, J., & Blockeel, H. (2009). A community-based platform for machine learning experimentation. In W. Buntine, M. Grobelnik, D. Mladenič, & J. Shawe-Taylor (Eds.), *Proceedings of machine learning and knowledge discovery in databases* (pp. 750–754). Springer.
Vapnik, V. (1995). *The nature of statistical learning theory*. Springer.
Vapnik, V. N. (1998). *Statistical learning theory*. Wiley.
Wang, P., Zhang, P., & Guo, L. (2012). Mining multi-label data streams using ensemble-based active learning. In *Proceedings of the 2012 SIAM international conference on data mining* (pp. 1131–1140). SDM.
Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). Amsterdam: Morgan Kaufman.
Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2007). Active learning from data streams. In *Proceedings of the seventh IEEE international conference on data mining, ICDM* (pp. 757–762). IEEE.
Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2010). Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 40*, 1607–1621.
Žliobaitė, I., Bifet, A., Pfahringer, B., & Holmes, G. (2011). Active learning with evolving streaming data. In *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD* (pp. 597–612). Springer.

Chapter 6

Conclusions and Further Work

The thesis presents a modern web-based platform for distributed computing, developed using the latest software technologies and computing paradigms for construction and execution of scientific workflows. The platform is, to the best of our knowledge, the only platform that provides a graphical user interface to the big data processing framework Disco and publicly available implementations of data mining algorithms in this platform. The ClowdFlows platform is adaptable and allows adding new workflow components via the inclusion of web services. Its well documented programmable interface allows for complete adaptations of the platform as evidenced in the presented ConCreTeFlows and TextFlows platforms. By unifying functionalities of different knowledge discovery platforms we have addressed novel scenarios which could not have been considered until now, such as relational data mining empowered by text mining. The platform's non-local nature is ideal for processing data streams, as evidenced by the active learning for sentiment analysis on streams of data from the microblogging platforms use case.

In the rest of this chapter the hypotheses of the thesis are summarized with claims supporting their validity along with the main scientific contributions of the thesis. We conclude by discussing the strengths and limitations of the current development, and provide directions for further work and improvements of the presented software platform, developed methodologies, and their implementations.

6.1 Scientific Contributions

In Section 1.4 we hypothesized that big data can efficiently be processed with a system that implements the visual programming paradigm and utilizes cloud computing by means of distributed hardware and software resources to improve scalability and adaptation to data of large proportions with nearly perfect linear speedup. In the publication included in Section 2.4 we empirically show that the ClowdFlows platform can handle progressively larger data sets by adding more computing power to the cluster of computing nodes. Likewise, the through-put of the real-time data stream analysis module can be increased to match the rate of incoming data by adding processing nodes. Similarly, in the publication included in Section 5.4 we analyze the performance of the platform and provide information on its limitations regarding the amount of concurrent users.

Our second hypothesis was that the platform will simplify the knowledge discovery process with big data and enable the analysis of data streams in real-time for non-experts and scientists from different domains of research. We have shown that ClowdFlows is adaptable to various fields of research other than data mining. Section 3.2 describes the adaptaion of the platform for computational creativity, while the platform described in Section 3.3.2 is focused on text mining and natural language processing. The workflow sharing mechanism of ClowdFlows and public workflows allow modification of existing public workflows and allow knowledge discovery without knowledge of particular software components. Use cases in forms of workflows can be copied, modified, and executed. Non-expert users can exploit publicly available workflows without the knowledge how to create one. In numbers, the public installation of ClowdFlows has more than 1,000 registered users that have produced more than 8,000 workflows. An independent survey has shown that the ClowdFlows platform is leading with regards to the number of features compared to related platforms [99].

Our third hypothesis was that by unifying functionalities of different knowledge discovery platforms and adding features to facilitate big data mining conjoined with real-time data analytics, we would now be able to address novel scenarios which were not possible until now. Among such novel scenarios are the active learning sentiment analysis workflow described in the publication included in Section 5.4, the use of relational data mining and inductive logic programming algorithms in a workflow environment as described in the publication included in Section 4.3, and workflows that employ the map-reduce paradigm for big data mining as described in the publication included in Section 2.4.

To summarize, the scientific contributions of the thesis are multi-fold.

1. Firstly, we have implemented a web-based knowledge discovery platform that features a browser-based workflow editor and facility for sharing the workflows on the web. The platform is capable of processing *big data* and features a real-time analysis module for continuous mining of data streams. The platform is developed using modern web development technologies and frameworks. It features numerous workflow components for various data mining tasks. Remote web services may be incorporated into the workflows by users during run-time. The platform was developed with a modular design, meaning that it can be installed on a cluster of machines, which enables scalability to ensure optimal performance regardless of the amount of data and amount of concurrent users. This contribution was covered in Chapter 2.

2. We have demonstrated that the platform is extensible to other fields of expertise by creating adaptations of the platform. The provided documentation is sufficient, as the platform has also successfully been adapted by users other than the author. This contribution was covered in Chapter 3.

3. Exploiting the innovative features of the platform to provide novel knowledge discovery scenarios in forms of scientific workflows is the second main scientific contribution of the thesis. We proposed a relational data mining scenario where a text mining inspired approach entitled wordification was evaluated in comparison with the existing propositionalization approaches on several relational datasets. The ClowdFlows platform's evaluation features were exploited to validate the wordification approach against comparable methodologies. As a side effect, other relational data mining algorithms were made available as ClowdFlows workflow components. Another novel use case that shows the real time analysis capabilities of ClowdFlows is the active learning for sentiment analysis on data streams of posts from microblogging social media web sites. We show that active learning improves the accuracy of sentiment classification and provides useful visualizations of textual data streams. These novel use cases were covered in Chapters 4 and 5.

6.2 Strengths and Limitations of the Approach

In this section we emphasize the strengths of the developed ClowdFlows platform and use case scenarios presented in this thesis, and critically evaluate their limitations.

6.2.1 Strengths of the ClowdFlows platform

The presented platform and the implemented use case scenarios which are easily accessible to end users and developers have numerous strengths, which were made evident throughout the thesis. A brief summary of strengths is re-emphasized below:

- ClowdFlows is a cloud-based platform. Workflows can be executed on powerful clusters of machines, while they can be designed on less powerful desktop or mobile devices.
- The platform is service-oriented. Web services can be imported as workflow components, which allows for easy extension of the roster of components.
- Developers can profit from the open-source nature of the platform by incorporating their own algorithms into ClowdFlows and using them in the workflow environment.
- Online access to workflows developed through the ClowdFlows platform allows for publishing research work online, and for experiment reuse as well as methodology and experimental design control.
- As ClowdFlows is a web application that is always running, it is an ideal setting for executing long-running workflows such as data stream mining workflows. Multiple workflow components are available for processing data streams.
- ClowdFlows supports processing of big data, which cannot be processed using conventional methods.
- Open access to predefined workflows in ClowdFlows makes the use of workflows easily accessible to non-experts.

6.2.2 Limitations and lessons learned

Potential users of ClowdFlows should be aware also of several limitations outlined below:

- ClowdFlows supports widgets for analyzing big data with the MapReduce paradigm, however these widgets are few, and a vast majority of widgets still process data in the classical sense and can falter on large amounts of data.
- The throughput of the stream analysis module is limited by the amount of computing nodes in the cluster. The limitations of the stream analysis module have been evaluated in the publication included in Section 2.4.
- Unless the user adds new components, the user is limited to the particular set of tools (such as classifier implementations, pre-processing tools, visualizations) that are available in ClowdFlows by default.
- Currently there are no available workflow components for performing deep learning, and the platform has no capabilities to leverage GPU processors which have been shown to speed up deep learning algorithms by orders of magnitude, reducing running times from weeks to days [100].
- Although composing new workflows from the existing ClowdFlows widgets looks easy and appealing, the user may get annoyed by the fact that there is currently a lack of professional documentation of available widgets and the lack of input and output types, making the development of new workflows difficult for a non-experienced user.

- Unless the user has ClowdFlows running on their own hardware, the public ClowdFlows instance is not intended for large experiments, due to memory and computation constraints, and a daily growing user base sharing hardware resources. The limitations regarding the number of concurrent users has been evaluated in the publication included in Section 5.4.
- The use of the public installation of ClowdFlows is not appropriate if the user is worried about privacy, since the data is copied and stored on the server, therefore such users need to run their own ClowdFlows instance.

The main lessons learned for the new user are that when developing a new workflow, it is best to start from pre-existing workflows, many of which have been published on the public installation of ClowdFlows. Another lesson learned is that due to the option of incorporating web services, the services may become unavailable without notification, which may cause a workflow to stop being executable. To mitigate this, the results of the last successful execution of the workflow are saved in the database.

6.3 Further Work

We envision several directions of further research, some of which we have already started working on. We divide further work into the development of the ClowdFlows platform and its features, the development of adapted platforms, and the development of use cases and workflows.

6.3.1 The ClowdFlows platform

Regarding the ClowdFlows platform we propose several ideas for future work. We will work on the de-monolithization of the platform into smaller parts to increase modularity. Currently, the core of the platform features a workflow execution engine, a graphical user interface, a big data module for mining in batch mode, and a stream mining module. Even though these components are called modules, they are essentially part of the same code base. We observe that this limits the portability and usability of ClowdFlows, therefore we intend to separate the core into several projects that can be interconnected and replaced at will. For example, the workflow execution engine could be invoked from a multitude of different graphical user interfaces, programmatically via an API, or via a command line interface. Likewise, the graphical user interface could be used to control a different type of workflow engine than the one provided by ClowdFlows.

ClowdFlows currently provides its own stream processing engine. We propose to utilize the ClowdFlows's graphical user interface for other popular stream processing engines such as Storm. Likewise, we plan to integrate Apache Hadoop and Apache Spark into the big data mining module to complement the Disco framework.

Scaling of ClowdFlows is achieved by adding new computing nodes to the cluster. To add a new node human intervention is required to provide configuration for the node. We plan to streamline the installation of the platform by providing one-click deployment solutions and provide an automatic mechanism of elastic scaling, which will scale the platform up when load is high, and down when the resources are free for a longer period of time.

Finally, to complement the repository of public workflows, we plan to provide a public repository of workflow components. Our intention is to allow adding user code to live installations of the platform. This poses challenges regarding compatibility and software dependencies of workflow components as well as security challenges.

6.3.2 Adaptations of ClowdFlows

There are several directions for future work regarding adaptations of ClowdFlows. With the de-monolithization of ClowdFlows we intend to expose workflow components from related platforms into external packages that can be included in any installation or adaptation of ClowdFlows. This would allow construction of workflows that feature components from ClowdFlows, TextFlows and ConCreTeFlows in a single workflow.

Regarding TextFlows, we plan to expand the widget repository with additional text preprocessing components (such as chunking, term extraction, syntactic parsing) and include various clustering algorithms. Furthermore, we will connect the platform to various external tools to better assist the user in the process of exploration and visualization of results.

6.3.3 Use cases and workflows

Currently, only a limited amount of workflow components are suited for big data processing. We wish to add new workflow components that broaden the possibility of big data mining in ClowdFlows. Ideally each workflow component should also have a big data processing counterpart.

Regarding the relational data mining use cases, we will address other problem settings (such as clustering) and use the approach for solving real-life relational problems. We plan to use the approach in a more elaborate scenario of mining heterogeneous data sources, involving a mixture of information from databases and text corpora.

Regarding the active learning sentiment analysis use case we intend to implement several strategies for selecting the data instances suitable for labeling and to allow users to select the most appropriate ones. In the current version of the use case, it is assumed that sentiment analysis is a two-class classification problem and tweets are classified only as positive or negative in order to enable efficient calculations in real time. As the sentiment can also be neutral, and our current implementation of the software does not allow 3-class classification, we plan to generalize the use case for multi-class classification problems. We wish to allow more control over the generation of the initial models and a richer selection of initially labeled data sets.

References

[1] J. W. Tukey, “The future of data analysis,” *Annals of Mathematical Statistics*, vol. 33, no. 1, pp. 1–67, Mar. 1962.

[2] J. R. Quinlan, *C4.5: Programs for machine learning.* Morgan Kaufmann, San Francisco, 1993.

[3] I. H. Witten, E. Frank, and M. A. Hall, *Data mining: Practical machine learning tools and techniques*, 3. Amsterdam: Morgan Kaufmann, 2011.

[4] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler, “Yale: Rapid prototyping for complex data mining tasks,” in *The Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, Eds., Philadelphia, PA, USA: ACM, Aug. 2006, pp. 935–940.

[5] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel, “KNIME: The Konstanz Information Miner,” in *Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation*, 2007, pp. 319–326.

[6] J. Demšar, B. Zupan, G. Leban, and T. Curk, “Orange: From experimental machine learning to interactive data mining,” in *The Proceedings of the 15th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2004, pp. 537–539.

[7] M. M. Burnett, “Visual Programming,” in *Wiley Encyclopedia of Electrical and Electronics Engineering.* John Wiley & Sons, Inc., New York, 2001, pp. 275–283.

[8] T. Erl, *Service-oriented architecture: Concepts, technology, and design.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[9] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive Online Analysis,” *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[10] G. D. F. Morales, “SAMOA: A platform for mining big data streams,” in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 777–778.

[11] Gartner, Inc. (2011). Gartner says solving ’big data’ challenge involves more than just managing volumes of data, [Online]. Available: `http://www.gartner.com/it/page.jsp?id=1731916` (visited on 02/28/2017).

[12] N. Marz. (2017). Storm, distributed and fault-tolerant realtime computation, [Online]. Available: `http://storm.apache.org` (visited on 02/28/2017).

[13] T. White, *Hadoop: The definitive guide.* O’Reilly Media, Inc., 2012.

[14] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM—50th anniversary issue: 1958–2008*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[15] P. Clark and R. Boswell, “Rule induction with cn2: Some recent improvements,” in *European Working Session on Learning*, Springer, 1991, pp. 151–163.

[16] G. Decker, H. Overdick, and M. Weske, “Oryx - an open modeling platform for the bpm community,” in *Business Process Management*, ser. Lecture Notes in Computer Science, M. Dumas, M. Reichert, and M.-C. Shan, Eds., vol. 5240, Springer Berlin / Heidelberg, 2008, pp. 382–385.

[17] D. Blankenberg, G. V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor, “Galaxy: A Web-Based Genome Analysis Tool for Experimentalists,” *Current Protocols in Molecular Biology*, vol. Chapter 19, Jan. 2001.

[18] R. Rak, A. Rowley, W. Black, and S. Ananiadou, “Argo: An integrative, interactive, text mining-based workbench supporting curation,” *Database: The Journal of Biological Databases and Curation*, vol. 2012, 2012.

[19] D. Hull, K. Wolstencroft, R. Stevens, C. A. Goble, M. R. Pocock, P. Li, and T. Oinn, “Taverna: A tool for building and running workflows of services,” *Nucleic Acids Research*, vol. 34, no. Web-Server-Issue, pp. 729–732, 2006.

[20] D. Talia, P. Trunfio, and O. Verta, “Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids,” in *The Proceedings of the 16th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2005, pp. 309–320.

[21] V. Podpečan, M. Zemenova, and N. Lavrač, “Orange4ws environment for service-oriented data mining,” *The Computer Journal*, vol. 55, no. 1, pp. 89–98, 2012.

[22] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, “Pegasus, a workflow management system for science automation,” *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[23] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, “Workflow management in condor,” English, in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., Springer London, 2007, pp. 357–375.

[24] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, “ASKALON: A Development and Grid Computing Environment for Scientific Workflows,” English, in *Workflows for e-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds., Springer London, 2007, pp. 450–471.

[25] J. Vanschoren and H. Blockeel, “A community-based platform for machine learning experimentation,” in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, W. Buntine, M. Grobelnik, D. Mladenič, and J. Shawe-Taylor, Eds., vol. 5782, Springer Berlin Heidelberg, 2009, pp. 750–754.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “TensorFlow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[27] Microsoft. (2016). Microsoft Azure Machine Learning, [Online]. Available: `https://azure.microsoft.com/en-us/services/machine-learning/` (visited on 02/28/2017).

[28] W. Fan and A. Bifet, “Mining big data: Current status, and forecast to the future,” *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 1–5, 2013.

[29] S. M. Weiss and N. Indurkhya, *Predictive data mining: A practical guide*. Morgan Kaufmann, 1998.

[30] D. Laney, "3D data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, p. 70, 2001.

[31] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10, San Jose, California: USENIX Association, 2010, pp. 21–21.

[32] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-Reduce for Machine Learning on Multicore," in *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., MIT Press, 2006, pp. 281–288.

[33] P. Mundkur, V. Tuulos, and J. Flatow, "Disco: A computing platform for large-scale data analytics," in *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*, ACM, 2011, pp. 84–89.

[34] T. A. Foundation. (2017). Apache mahout, [Online]. Available: `http://mahout.apache.org` (visited on 02/28/2017).

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[36] Z. Prekopcsák, G. Makrai, T. Henk, and C. Gáspár-Papanek, "Radoop: Analyzing big data with RapidMiner and Hadoop," in *Proceedings of the 2nd rapidminer community meeting and conference (rcomm 2011)*, 2011.

[37] E. Ikonomovska, S. Loskovska, and D. Gjorgjevik, "A survey of stream data mining," in *Proceedings of 8th National Conference with International Participation, ETAI*, 2007, pp. 19–21.

[38] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *The Proceedings of the 2010 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2010, pp. 170–177.

[39] P. Reutemann and J. Vanschoren, "Scientific workflow management with ADAMS," in *Proceedings of the 23rd European Conference on Machine Learning and Knowledge Discovery in Databases*, 2012, pp. 833–837.

[40] J. Kranjc, V. Podpečan, and N. Lavrač, "A browser-based platform for service-oriented knowledge discovery," in *Proceedings of the 23rd European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2011, pp. 30–35.

[41] J. Kranjc, V. Podpečan, and N. Lavrač, "Knowledge discovery using a service oriented web application," in *Proceedings of the 4th International Conference on Information, Process, and Knowledge Management*, IARIA, 2012, pp. 82–87.

[42] J. Kranjc, V. Podpečan, and N. Lavrač, "ClowdFlows: A cloud based scientific workflow platform," in *Planning to Learn and Service-Oriented Knowledge Discovery, PlanSoKD'11*, Springer, 2012, pp. 816–819.

[43] International Conference on Computational Science. (2016). Executable paper challenge, [Online]. Available: `http://web.archive.org/web/20160328092925/http://www.executablepapers.com/` (visited on 02/28/2017).

[44] J. Kranjc, V. Podpečan, and N. Lavrač, “Real-time data analysis in ClowdFlows,” in *Proceedings of the 2013 IEEE International Conference on Big Data*, IEEE, 2013, pp. 15–22.

[45] J. Kranjc, R. Orač, V. Podpečan, N. Lavrač, and M. Robnik-Šikonja, “ClowdFlows: Online workflows for distributed big data mining,” *Future Generation Computer Systems*, vol. 68, pp. 38–58, 2016.

[46] M. Žnidaršič, A. Cardoso, P. Gervás, P. Martins, R. Hervás, A. O. Alves, H. G. Oliveira, P. Xiao, S. Linkola, H. Toivonen, J. Kranjc, and N. Lavrač, “Computational creativity infrastructure for online software composition: A conceptual blending use case,” in *Proceedings of the 7th International Conference on Computational Creativity*, ICCC, 2016, pp. 371–378.

[47] M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, and N. Lavrač, “TextFlows: A visual programming platform for text mining and natural language processing,” *Science of Computer Programming*, vol. 121, pp. 128–152, 2016.

[48] N. Lavrač, M. Perovšek, and A. Vavpetič, “Propositionalization online,” in *Proceedings of the 25th european conference on machine learning and knowledge discovery in databases*, Springer, 2014, pp. 456–459.

[49] P. A. Flach and N. Lachiche, “Confirmation-guided discovery of first-order rules with tertius,” *Machine Learning*, vol. 42, no. 1-2, pp. 61–95, 2001.

[50] P. Flach and N. Lachiche, “1BC: A first-order Bayesian classifier,” in *International Conference on Inductive Logic Programming*, Springer, 1999, pp. 92–103.

[51] M. A. Krogel and S. Wrobel, “Transformation-based learning using multirelational aggregation,” in *International Conference on Inductive Logic Programming*, Springer, 2001, pp. 142–155.

[52] S. Colton and G. A. Wiggins, “Computational creativity: The final frontier?” In *Proceedings of the 20th European conference on artificial intelligence*, IOS Press, 2012, pp. 21–26.

[53] H. G. Oliveira, “Poetryme: A versatile platform for poetry generation,” *Computational Creativity, Concept Invention, and General Intelligence*, vol. 1, p. 21, 2012.

[54] T. Veale and G. Li, “Specifying viewpoint and information need with affective metaphors: a system demonstration of the metaphor magnet web app/service,” in *Proceedings of the ACL 2012 System Demonstrations*, Association for Computational Linguistics, 2012, pp. 7–12.

[55] M. T. Llano, S. Colton, R. Hepworth, and J. Gow, “Automated fictional ideation via knowledge base manipulation,” *Cognitive Computation*, vol. 8, no. 2, pp. 153–174, 2016.

[56] J. Charnley, S. Colton, and M. T. Llano, “The FloWr framework: Automated flowchart construction, optimisation and alteration for creative systems,” in *Proceedings of the 5th International Conference on Computational Creativity*, vol. 9, 2014.

[57] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, 2007.

[58] J. Han and M. Kamber, *Data mining: Concepts and techniques.* Morgan Kaufmann, San Francisco, USA, 2006.

[59] M. Poch, A. Toral, O. Hamon, V. Quochi, and N. Bel Rafecas, “Towards a user-friendly platform for building language resources based on web services,” in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC’12)*, ELRA (European Language Resources Association), 2012.

[60] E. Hinrichs, M. Hinrichs, and T. Zastrow, “Weblicht: Web-based lrt services for german,” in *Proceedings of the acl 2010 system demonstrations*, Association for Computational Linguistics, 2010, pp. 25–29.

[61] T. Ishida, *The language grid: Service-oriented collective intelligence for language resource interoperability.* Springer Science & Business Media, 2011.

[62] N. Ide, J. Pustejovsky, C. Cieri, E. Nyberg, D. DiPersio, C. Shi, K. Suderman, M. Verhagen, D. Wang, and J. Wright, “The language application grid,” in *International workshop on worldwide language service infrastructure*, Springer, 2015, pp. 51–70.

[63] S. Bird, “NLTK: The natural language toolkit,” in *Proceedings of the COLING/ACL on Interactive presentation sessions*, Association for Computational Linguistics, 2006, pp. 69–72.

[64] M. Grčar, D. Mladenič, M. Grobelnik, B. Fortuna, and J. Brank, “Ontology learning implementation,” Jožef Stefan Institute, Tech. Rep., 2006.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[66] S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, *Inductive logic programming.* Springer, 1992, vol. 38.

[67] S. Džeroski and N. Lavrač, *Relational data mining.* Springer, 2001.

[68] M.-A. Krogel, S. Rawles, F. Železny, P. A. Flach, N. Lavrač, and S. Wrobel, “Comparative evaluation of approaches to propositionalization,” in *International Conference on Inductive Logic Programming*, Springer, 2003, pp. 197–214.

[69] S. Kramer, B. Pfahringer, and C. Helma, “Stochastic propositionalization of non-determinate background knowledge,” in *International Conference on Inductive Logic Programming*, Springer, 1998, pp. 80–94.

[70] M. Perovšek, A. Vavpetič, B. Cestnik, and N. Lavrač, “A wordification approach to relational data mining,” in *International Conference on Discovery Science*, Springer, 2013, pp. 141–154.

[71] K. Sparck Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[72] F. Železny and N. Lavrač, “Propositionalization-based relational subgroup discovery with rsd,” *Machine Learning*, vol. 62, no. 1-2, pp. 33–63, 2006.

[73] O. Kuželka and F. Železny, “Block-wise construction of tree-like relational features with monotone reducibility and redundancy,” *Machine Learning*, vol. 83, no. 2, pp. 163–192, 2011.

[74] A. Srinivasan, *The Aleph Manual*, 2001.

[75] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation,” in *Encyclopedia of database systems*, Springer, 2009, pp. 532–538.

[76] B. Sluban, D. Gamberger, and N. Lavrač, “Ensemble-based noise detection: Noise ranking and visual performance evaluation,” *Data Mining and Knowledge Discovery*, vol. 28, no. 2, pp. 265–303, 2014.

[77] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[78] J. Smailović, M. Žnidaršič, and M. Grčar, "Web-based experimental platform for sentiment analysis," in *Proceedings of the 3rd International Conference on Information Society and Information Technologies (ISIT)*, Citeseer, 2011.

[79] T. T. Thet, J.-C. Na, C. S. Khoo, and S. Shakthikumar, "Sentiment analysis of movie reviews on discussion boards using a linguistic approach," in *Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, ACM, 2009, pp. 81–84.

[80] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: Sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, Association for Computational Linguistics, 2002, pp. 79–86.

[81] J. Smailović, "Sentiment analysis in streams of microblogging posts," PhD thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2014.

[82] G. Petz, M. Karpowicz, H. Fürschuß, A. Auinger, S. M. Winkler, S. Schaller, and A. Holzinger, "On text preprocessing for opinion mining outside of laboratory environments," in *International Conference on Active Media Technology*, Springer, 2012, pp. 618–629.

[83] G. Petz, M. Karpowicz, H. Fürschuß, A. Auinger, V. Střitesky, and A. Holzinger, "Opinion mining on the web 2.0–characteristics of user generated content and their impacts," in *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, Springer, 2013, pp. 35–46.

[84] B. Settles, M. Craven, and S. Ray, "Multiple-instance active learning," in *Advances in neural information processing systems*, 2008, pp. 1289–1296.

[85] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, vol. 52, no. 55-66, p. 11, 2010.

[86] W. Chu, M. Zinkevich, L. Li, A. Thomas, and B. Tseng, "Unbiased online active learning in data streams," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2011, pp. 195–203.

[87] S. Argamon-Engelson and I. Dagan, "Committee-based sample selection for probabilistic classifiers," *Journal of Artificial Intelligence Research*, vol. 11, pp. 335–360, 1999.

[88] P. Wang, P. Zhang, and L. Guo, "Mining multi-label data streams using ensemble-based active learning," in *Proceedings of the 2012 SIAM international conference on data mining*, SIAM, 2012, pp. 1131–1140.

[89] A. Bifet and E. Frank, "Sentiment knowledge discovery in twitter streaming data," in *International Conference on Discovery Science*, Springer, 2010, pp. 1–15.

[90] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A system for real-time twitter sentiment analysis of 2012 us presidential election cycle," in *Proceedings of the ACL 2012 System Demonstrations*, Association for Computational Linguistics, 2012, pp. 115–120.

[91] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM SIGMOD Record*, vol. 32, no. 2, pp. 5–14, 2003.

[92] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky, “Resource-aware knowledge discovery in data streams,” in *International Workshop on Knowledge Discovery in Data Streams*, Proceedings of the 15th European Conference on Machine Learning and Knowledge Discovery in Databases, 2004.

[93] D. Barbar'a, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, *et al.*, “The new jersey data reduction report,” in *IEEE Data Engineering Bulletin*, Citeseer, 1997.

[94] P. Domingos and G. Hulten, “A general method for scaling up machine learning algorithms and its application to clustering,” in *Proceedings of the Eighteenth International Conference on Machine Learning*, vol. 1, 2001, pp. 106–113.

[95] S. Acharya, P. B. Gibbons, and V. Poosala, “Congressional samples for approximate answering of group-by queries,” in *ACM SIGMOD Record*, ACM, vol. 29, 2000, pp. 487–498.

[96] G. S. Manku and R. Motwani, “Approximate frequency counts over data streams,” in *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB Endowment, 2002, pp. 346–357.

[97] M. M. Gaber, S. Krishnaswamy, and A. B. Zaslavsky, “Resource-aware mining of data streams,” *Journal of Universal Computer Science*, vol. 11, no. 8, pp. 1440–1453, 2005.

[98] J. Smailović, J. Kranjc, M. Grčar, M. Žnidaršič, and I. Mozetič, “Monitoring the Twitter sentiment during the Bulgarian elections,” in *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2015, pp. 1–10.

[99] O. Kurasova, V. Marcinkevičius, V. Medvedev, and A. Rapečka, “Data mining systems based on web services,” *Informacijos mokslai*, vol. 65, pp. 66–74, 2013.

[100] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 873–880.

Bibliography

Publications Related to the Thesis

Journal Articles

J. Kranjc, R. Orač, V. Podpečan, N. Lavrač, and M. Robnik-Šikonja, "ClowdFlows: Online workflows for distributed big data mining," *Future Generation Computer Systems*, vol. 68, pp. 38–58, 2016.

J. Kranjc, J. Smailović, V. Podpečan, M. Grčar, M. Žnidaršič, and N. Lavrač, "Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform," *Information Processing & Management*, vol. 51, no. 2, pp. 187–203, 2015.

M. Perovšek, A. Vavpetič, J. Kranjc, B. Cestnik, and N. Lavrač, "Wordification: Propositionalization by unfolding relational data into bags of words," *Expert Systems with Applications*, vol. 42, no. 17, pp. 6442–6456, 2015.

M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, and N. Lavrač, "TextFlows: A visual programming platform for text mining and natural language processing," *Science of Computer Programming*, vol. 121, pp. 128–152, 2016.

Conference Papers

J. Kranjc, V. Podpečan, and N. Lavrač, "ClowdFlows: A cloud based scientific workflow platform," in *Planning to Learn and Service-Oriented Knowledge Discovery, PlanSoKD'11*, Springer, 2012, pp. 816–819.

J. Smailović, J. Kranjc, M. Grčar, M. Žnidaršič, and I. Mozetič, "Monitoring the Twitter sentiment during the Bulgarian elections," in *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2015, pp. 1–10.

J. Kranjc, V. Podpečan, and N. Lavrač, "Real-time data analysis in ClowdFlows," in *Proceedings of the 2013 IEEE International Conference on Big Data*, IEEE, 2013, pp. 15–22.

J. Kranjc, V. Podpečan, and N. Lavrač, "Knowledge discovery using a service oriented web application," in *Proceedings of the 4th International Conference on Information, Process, and Knowledge Management*, IARIA, 2012, pp. 82–87.

J. Kranjc, V. Podpečan, and N. Lavrač, "A browser-based platform for service-oriented knowledge discovery," in *Proceedings of the 23rd European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2011, pp. 30–35.

M. Perovšek, V. Podpečan, J. Kranjc, T. Erjavec, S. Pollak, N. Q. Do Thi, X. Liu, C. Smith, M. Cavazza, and N. Lavrač, "Text mining platform for NLP workflow design, replication and reuse," in *Proceedings of the IJCAI Workshop on Replicability and Reusability in Natural Language Processing: From Data to Software Sharing*, AAAI Press, 2015, pp. 12–20.

S. Pollak, A. Vavpetič, J. Kranjc, N. Lavrač, and Š. Vintar, "NLP workflow for on-line definition extraction from English and Slovene text corpora.," in *Proceedings of the 11th Conference on Natural Language Processing*, Österreichischen Gesellschaft für Artificial Intelligende, 2012, pp. 53–60.

M. Žnidaršič, A. Cardoso, P. Gervás, P. Martins, R. Hervás, A. O. Alves, H. G. Oliveira, P. Xiao, S. Linkola, H. Toivonen, J. Kranjc, and N. Lavrač, "Computational creativity infrastructure for online software composition: A conceptual blending use case," in *Proceedings of the 7th International Conference on Computational Creativity*, ICCC, 2016, pp. 371–378.

Other Publications

Conference Papers

S. Pollak, B. Lesjak, J. Kranjc, V. Podpečan, and N. Lavrač, "Robochair: Creative assistant for question generation and ranking," in *The Proceedings of 2015 IEEE Symposium Series on Computational Intelligence*, IEEE, 2015, pp. 1468–1475.

J. Smailovič, M. Žnidaršič, N. Lavrač, S. Pollak, and J. Kranjc, "Modelling human appreciation of machine generated what-if ideas," in *The Proceedings of the 29th Advances in Neural Information Processing Systems: Constructive machine learning*, 2016, p. 5.

Biography

Janez Kranjc was born on November 16, 1985 in Ljubljana, Slovenia. He finished his primary and secondary education in Ljubljana, Slovenia. In 2004 he started his studies at the Faculty of Computer Science, University of Ljubljana. In 2010 he finished the undergraduate programme by defending his BSc thesis entitled "Development of web application for machine learning" under the supervision of Prof. Dr. Igor Kononenko and co-supervision of Prof. Dr. Nada Lavrač.

In the fall of 2010, he started his graduate studies at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. He enrolled in the PhD programme "Information and Communication Technologies" and started working as a research assistant at the Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia.

During his work as a research assistant he collaborated on the EU funded projects e-LICO (An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science), BISON (Bisociation Networks for Creative Information Discovery), FIRST (Large Scale Information Extraction and Integration Infrastructure for Supporting Financial Decision Making), MUSE (Machine Understanding for interactive StorytElling), ConCreTe (Concept Creation Technology), and WHIM (The What-if Machine), and attended several summer schools including Logic, Language and Information (ESSLLI 2011), where he also served as co-chair of the student session ESSLISTUS, and Language and Speech Technologies (SSLST 2011).

His research in the field of computer science mostly focuses on developing novel cloud-based approaches to data mining and knowledge discovery using graphical scientific workflows and developing applications and infrastructure to facilitate continuous mining of data streams and vast amounts of data dispersed on the cloud. The applications of the developed platform and workflows span several fields such as data mining, text mining, natural language processing, and sentiment analysis. He presented his work at international conferences and workshops.