COMPLEX NODES IN TREES FOR STRUCTURED OUTPUT PREDICTION

Tomaž Stepišnik

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Tomaž Stepišnik

COMPLEX NODES IN TREES FOR STRUCTURED OUTPUT PREDICTION

Doctoral Dissertation

KOMPLEKSNA VOZLIŠČA V DREVESIH ZA NAPOVEDOVANJE STRUKTURIRANIH VREDNOSTI

Doktorska disertacija

Supervisor: Dr. Dragi Kocev

Co-Supervisor: Prof. Dr. Sašo Džeroski

Ljubljana, Slovenia, December 2020

Moji družini

Acknowledgments

I am most grateful to my supervisor Dragi Kocev for his mentorship over the last 4 years. In the beginning, I was new to the field so I had to learn some lessons multiple times. I am thankful to Dragi for his patience and especially appreciate the time he spent working with me at unorthodox hours.

I also thank my co-supervisor Sašo Džeroski who gave me the opportunity to work here and provided several directions for research.

I thank Mili Bauer for always being available to help with any administrative or other issues that I have encountered during this period.

I thank Matej Petković, Blaž Škrlj, Jure Brence, and Martin Breskvar for the brainstorming sessions we have had and for making working hours more fun in general.

I thank Aljaž Osojnik, Jovan Tanevski, Panče Panov, and Nikola Simidjievski for multiple interesting discussions.

I thank Jörg Wicker and his group for hosting me in Auckland for three unforgettable months.

I thank Maša Matijašević for proofreading the text in record time.

I also thank the evaluation board – Ljupčo Todorovski, Michelangelo Ceci, and Grigorios Tsoumakas – for their time spent reading and evaluating this thesis, and for providing suggestions on how to improve it.

Finally, I thank Javna agencija za raziskovalno dejavnost Republike Slovenije for the young researcher grant that funded my research.

Abstract

In this thesis, we integrate complex nodes into predictive clustering trees (PCTs). PCTs are well-established machine learning models that are very flexible in terms of the machine learning tasks that they can address, including structured output prediction and semi-supervised learning. Like standard decision trees, they are learned with a greedy top-down induction algorithm that comes with two weaknesses. First, because of the greediness, the algorithm makes myopic decisions, which can lead to sub-optimal trees. Second, the split selection procedure scales poorly with the dimensionality of the output. This can lead to prohibitively long learning times in structured output prediction problems.

To reduce the myopia of the learning algorithm, we extend PCTs with option nodes. When there are multiple splits with heuristic scores close to the maximum one, option PCTs do not select only the best split but combine several alternatives in an option node. For each alternative split, the tree then continues to grow normally. When making predictions, each alternative split in an option node makes its own prediction. These predictions are then aggregated into the final prediction. We evaluate option PCTs on several structured output prediction tasks: multi-target regression, multi-label classification, and hierarchical multi-label classification. In the evaluation, we focus on how the number of option nodes introduced in the tree affects the trade-off between the predictive performance and tree size (which is the main factor of its interpretability). We show that if the number of option nodes is small, the option PCT remains interpretable while improving the performance of a standard PCT. On the other hand, option PCTs with many option nodes exhibit similar behavior to bagging ensembles of PCTs in terms of predictive performance, model size, and learning time.

To improve the computational complexity of learning PCTs, we propose PCTs with oblique split nodes. In contrast to standard PCTs, oblique PCTs use linear combinations of features in tests to split the examples. We propose two methods for learning the oblique splits in PCTs. The SVM variant first clusters the examples based on the outputs to get the ideal split, then learns a linear support vector machine on the features that approximates this split. The gradient variant uses a differentiable approximation of the heuristic used by PCTs to efficiently optimize oblique splits with gradient-based methods. The proposed methods improve the computational scaling of standard PCTs and provide additional computational benefits on sparse data. These advantages are first confirmed with theoretical analysis and later with empirical evaluation. The results of the experiments also show that the predictive performance of oblique PCTs is on par with that of PCTs, often even exceeding it. We also evaluate the approach in a semi-supervised setting and get similar results: oblique PCTs are learned faster and often achieve better performance than PCTs as well. Additionally, we present a method for estimating feature importances from oblique PCTs and perform experiments that confirm they are meaningful.

Povzetek

Tema te disertacije je integracija kompleksnih vozlišč v drevesa za napovedno razvrščanje (DNR). DNR-ji so uveljavljeni modeli v strojnem učenju, ki jih lahko uporabimo za vrsto različnih nalog, med drugim napovedovanje strukturiranih vrednosti in polnadzorovano učenje. Zgrajeni so s požrešnim rekurzivnim algoritmom, ki ima dve šibki točki. Prva težava je njegova kratkovidnost, ki je posledica požrešnosti in pomeni, da so naučena drevesa redko optimalna. Druga težava pa je računska zahtevnost učenja testov v delitvenih vozliščih drevesa, ki hitro narašča s številom izhodnih spremenljivk. To pride še posebej do izraza pri napovedovanju strukturiranih vrednosti.

Za zmanjšanje kratkovidnosti uporabimo opcijska vozlišča in predlagamo opcijske DNR-je. Ko naletimo na več testov, ki izgledajo podobno dobri kot najboljši med njimi, opcijski DNR-ji ne izberejo samo najboljšega, ampak združijo vse alternative v opcijsko vozlišče. Za vsako alternativo se potem drevo normalno gradi naprej. Ko z zgrajenim opcijskim DNR-jem delamo napovedi, opcijsko vozlišče primere posreduje vsem vsebujočim alternativam. Vsaka izmed njih pripravi svojo napoved, le-te pa se potem združi v končno napoved drevesa. Opcijske DNR-je smo evalvirali na več nalogah napovedi strukturiranih podatkov: večciljni regresiji, večoznačni klasifikaciji in hierarhični večoznačni klasifikaciji. Cilj empiričnih raziskav je bil raziskati vpliv števila opcijskih vozlišč na napovedno moč dreves in njihovo velikost, ki ključno vpliva na zmožnost razlage naučenih modelov. Rezultati kažejo, da imajo opcijski DNR-ji z malo opcijskimi vozlišči boljšo napovedno moč od navadnih DNR-jev, še vedno pa ostanejo razložljivi. Po drugi strani pa so napovedna moč, velikost modelov in čas učenja opcijskih DNR-jev z veliko opcijskimi vozlišči podobni tem od ansamblov DNR-jev.

Da izboljšamo računsko zahtevnost učenja DNR-jev, jih razširimo s poševnimi delitvenimi vozlišči, ki v svojih testih uporabljajo linearne kombinacije značilk. Takšna drevesa poimenujemo poševni DNR-ji in predlagamo dve metodi za njihovo konstrukcijo. MPV različica najprej razvrsti primere v dve skupini na podlagi izhodnih vrednosti in tako dobi idealno delitev, potem pa s pomočjo linearne metode podpornih vektorjev poišče približek te delitve na podlagi značilk. Gradientna različica pa definira odvedljiv približek kriterijske funkcije, ki jo za ocenjevanje delitev uporabljajo navadni DNR-ji, za učinkovito optimizacijo testa pa uporabi gradientne optimizacijske metode. Predlagani varianti izboljšata časovno zahtevnost algoritma na problemih z več izhodnimi spremenljivkami, dodatne računske prihranke pa nudita na redkih podatkih. Računsko prednost najprej potrdimo s teoretično analizo računske zahtevnosti, nato pa še z empirično študijo. Rezultati eksperimentov pokažejo tudi to, da poševni DNR-ji ohranijo ali celo izboljšajo vrhunsko napovedno moč navadnih DNR-jev. Tudi evalvacija v kontekstu polnadzorovanega učenja prikaže podobno sliko: poševni DNR-ji so hitrejši in imajo pogosto tudi boljšo napovedno moč. Predstavimo tudi način, kako lahko iz naučenih poševnih DNR-jev pridemo do ocen pomembnosti značilk, za katere z eksperimenti pokažemo, da so smiselne.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

SOP	...	structured output prediction
SSL	...	semi-supervised learning
BC	...	binary classification
MCC	...	multi-class classification
MLC	...	multi-label classification
HMLC	...	hierarchical multi-label classification
STR	...	single-target regression
MTR	...	multi-target regression
PCT	...	predictive clustering tree
OPCT	...	option predictive clustering tree
SPYCT	...	oblique predictive clustering tree
BAG	...	bagging ensemble
RF	...	random forest ensemble

Chapter 1

Introduction

In recent years, the popularity of *machine learning* has greatly increased. We are collecting more and more data and the improved hardware capabilities provide the computational power to process it. This combination has resulted in many exciting applications of machine learning methods. Computers achieved (or exceeded) human-level performance in many complex games (Mnih et al., 2013; Silver et al., 2017; Vinyals et al., 2019), are learning to drive cars (Badue et al., 2021; Bojarski et al., 2016), help in the development of drugs (Hodos et al., 2016; Sliwoski et al., 2014), analyze medical images (Anwar et al., 2018; Araújo et al., 2017), generate coherent text (Brown et al., 2020; Latitude, 2020), and much more.

The most common task in machine learning is *supervised learning* (Bratko, 2001; Langley & Morgan, 1996), where we are given a set of input-output pairs (learning examples) and need to learn a model that maps inputs to outputs. The goal is to learn a model that makes accurate output predictions for inputs that were not used for learning. In *unsupervised learning*, on the other hand, we are simply given some data (no input/output distinction) with the goal to find groups of similar examples (clustering) or learn (lower-dimensional) vector representations of examples (representation learning, embedding). *Semi-supervised* learning combines supervised and unsupervised learning. We are given a set of input-output pairs (*labeled* examples) and another set of inputs, that are not associated with any outputs (*unlabeled* examples). The goal of semi-supervised learning is to exploit both labeled and unlabeled examples to learn a model that maps inputs to outputs. It is mainly used when unlabeled examples are plentiful but labeled examples are rare, which limits the ability of supervised learning to generate a model with good predictive performance. This often happens in life sciences where labeling examples takes a lot of time or money because it requires wet-lab experiments. For example, when developing a new drug, we perform a screening on a handful of initial candidates. They are the labeled examples because we measured their effect on the disease progression. To complement them, we can collect the descriptions of any number of compounds from public online databases for which we do not know the effect on the progression of the disease in question. However, we can still make them available to a semi-supervised method as unlabeled examples.

The term *predictive modeling* is also used to refer to the tasks where the goal is to construct a model that maps inputs to outputs (e.g., supervised and semi-supervised learning tasks). An important consideration in predictive modeling problems is the type of output data that we are dealing with. The most basic example is *binary classification*, where the examples can be labeled with one of two possible labels, e.g., we predict whether students will pass an exam or not. If there are more than two (but still a finite number of) possible labels, the task is known as *multi-class classification*. To continue our previous example, we could try to differentiate between three groups of students: those who will not pass the exam, those who will pass but will not impress, and those who will excel. Of course, we could also try to predict their numeric grades directly. In this case, the output is numeric and the task is called *regression*. Because the types of the targets in these cases are primitive, we will refer to this group of tasks as *primitive output prediction* tasks. They are among the most commonly addressed tasks in predictive modeling.

However, many real-life problems are naturally described with more complex output types composed of multiple variables, which can have additional structure or dependencies between them. These tasks are grouped under the term *structured output prediction* (Bakır et al., 2007). For example, in *multi-target regression*, we are interested in jointly predicting multiple numeric values (e.g., the grades of a student on exams for different courses). We might also be interested in predicting which courses students will pass (or simply be good at) and give them recommendations based on that. There are multiple courses and each student can (hopefully) pass multiple of them, i.e., examples can have multiple labels. This task is known as *multi-label classification*. We can also imagine that some courses are more related than other, e.g., courses Algebra 1 and Algebra 2 are more similar to each other than Programming 1 and Physics 1. Based on this information, we can construct a hierarchy of topics with specific courses in the leaves. We could provide this hierarchy to the learning method, and the model would then predict the topics and courses a student would do well at. Such a task is called *hierarchical multi-label classification*.

A straightforward approach to structured output prediction tasks is to first decompose them into multiple primitive output prediction tasks and solve them individually. This is known as the *local approach* (Silla & Freitas, 2011). For example, when predicting which classes a student will pass, we can construct a model for each class that predicts whether the student passes that particular class. In contrast, the *global* approach (Silla & Freitas, 2011) constructs a single model that predicts the entire structured output simultaneously. The global approach can produce more accurate models and is generally faster than the local approach (Kocev et al., 2013).

Another important consideration for predictive modeling is the type of input data. Traditionally, machine learning methods expect the input data in *tabular* form, i.e., given as a table. Each row represents one example, and each column presents one feature describing the examples. The success of the machine learning methods strongly depends on the information provided by the features and whether it is enough to make accurate predictions. Data is often naturally presented in tabular form, but for some inputs, this is not the case, for example, text, images, sequences, and graphs. One approach to dealing with such inputs is to transform them into tabular data by describing each example with a set of handcrafted features (e.g., a bag of words for text). Recently, however, state-of-the-art performance with such inputs is mostly achieved with *deep learning* methods, which can handle them directly with the use of *convolutional layers* (He et al., 2016; Krizhevsky, 2014; Ma et al., 2018), *recurrent neural networks* (Liu et al., 2016; Winter et al., 2019), and *attention mechanism* (Vaswani et al., 2017; Veličković et al., 2017).

Regardless of the popularity of deep learning, when the examples can be described with good features in tabular format, traditional machine learning methods still perform just as well (or better) and are typically easier to use. A prominent traditional method is the induction of *decision and regression trees* (Breiman et al., 1984). The advantages of tree-based methods are that they are relatively fast (to learn and to make predictions) and that it is easy to inspect why a certain prediction was made. They are often combined into ensembles (Breiman, 1996, 2001) and the ensembles typically achieve great performance. Even though inspecting individual predictions in ensembles is less feasible, the models can still be interpreted via various feature ranking methods (Petković et al., 2019a; Petković et al., 2020). While there are novel methods such as SHAP (Lundberg & Lee, 2017) that are capable of explaining individual predictions for arbitrary models, they are still better suited to tree-based methods (Lundberg et al., 2020).

Predictive clustering trees (Blockeel et al., 2002; Blockeel et al., 1998) generalize standard decision and regression trees towards structured output prediction (Kocev et al., 2020; Kocev et al., 2013), semi-supervised learning (Levatić et al., 2017; Levatić et al., 2020) and unsupervised learning (Dimitrovski et al., 2016). They retain the performance and interpretability of standard trees and ensembles thereof. The main topic of this thesis is to extend and improve predictive clustering trees by introducing complex nodes into the trees.

1.1 Motivation

The motivation for this thesis is twofold. First, predictive clustering trees, like standard decision trees, are induced following the traditional top-down approach. It is a greedy algorithm that finds the best split at each step and proceeds recursively on the resulting subsets of examples. The advantage of such an approach is that it is computationally efficient. But selecting a split that appears to be the best at the moment without considering future options can lead to sub-optimal trees. To alleviate this issue, classification trees have been extended with *option nodes* (Buntine, 1992; Kohavi & Kunz, 1997). When multiple splits appear to be of similar quality, instead of selecting only the best one, an option node is created, which stores several alternative splits. From that point on, the tree construction continues recursively for each alternative split. This way, a larger subspace of trees is investigated, which can result in a better model. We wish to extend predictive clustering trees with option nodes and investigate their potential for structured output prediction problems.

The other motivation was the improvement of the computational efficiency of learning predictive clustering trees on structured output prediction tasks. While it is efficient for primitive tasks, it scales poorly with the number of output variables. This becomes especially noticeable in (hierarchical) multi-label classification problems with thousands of possible labels. It also affects semi-supervised learning of predictive clustering trees. Furthermore, label sets in multi-label problems are typically presented as binary vectors. The length of the vector equals the number of possible labels, and the value of each component denotes the presence (value 1) or absence (value 0) of a specific label. Even though there are often thousands of possible labels, most examples are typically labeled with only a handful of them. This leads to binary vectors with a very high dimensionality and mostly zero-valued components, i.e., *sparse* vectors. This sparsity could potentially be exploited to make learning faster, but standard predictive clustering trees cannot profit from it. To improve the computational efficiency, we consider *oblique split nodes* (Breiman et al., 1984; Murthy et al., 1994). Whereas the split nodes in standard trees perform tests on a single feature, oblique split nodes consider linear combinations of features in the tests. We believe this can bring computational advantages on three fronts:

- By carefully designing the optimization of the oblique splits, we can improve the scalability of learning a split with respect to the number of outputs.
- By storing sparse data in a sparse matrix format and performing matrix operations in this representation, we can achieve an additional computational advantage.
- By using oblique splits, the trees can fit the data better with fewer split nodes. This should result in smaller models and lower learning times.

1.2 Goals and Hypotheses

Following the motivation presented above, the main goal of the thesis is to address the presented limitations of predictive clustering trees (PCTs), focusing on the context of structured output prediction. Specifically, we will develop PCTs with different types of complex tree nodes. To achieve this goal, we will do the following.

- Develop option nodes for the predictive clustering trees to obtain option predictive clustering trees (OPCTs).
- Evaluate the OPCTs on benchmark datasets for structured output prediction tasks with emphasis on the influence of the number of option nodes.
- Design and implement efficient oblique predictive clustering trees and ensembles thereof.
- Evaluate oblique PCTs on benchmark datasets for standard classification and regression tasks as well as for structured output prediction tasks. Emphasis will be on the efficiency on high dimensional and sparse data.
- Evaluate oblique PCTs on semi-supervised learning tasks.
- Demonstrate that feature importance scores obtained from oblique PCTs are meaningful.

Related to the listed goals, we formulate several hypotheses that we then investigate throughout the thesis.

H1 Introducing few option nodes in a PCT improves its predictive performance compared to standard PCTs while retaining the interpretability potential.

H2 Introducing many option nodes in a PCT makes the option PCTs exhibit similar behavior to bagging ensembles of PCTs in terms of time complexity, size, and predictive performance.

H3 Oblique predictive clustering trees reduce learning time on high dimensional data compared to standard PCTs without deteriorating predictive performance.

H4 Oblique predictive clustering trees exploit sparse data to reduce learning time.

H5 Oblique predictive clustering trees are learned significantly faster than standard PCTs in a semi-supervised learning setting.

H6 Meaningful feature importances are calculated by using oblique PCTs.

1.3 Methodology

In this section, we present the methodology used to achieve the goals that we have set up. We started by studying the specifics of the SOP tasks (Bakır et al., 2007), the working of predictive clustering trees (Blockeel et al., 1998), and how they are applied to SOP tasks (Blockeel et al., 2002; Kocev et al., 2013). To develop option PCTs we studied the relevant literature on option trees (Buntine, 1992; Kohavi & Kunz, 1997). We implemented them into CLUS predictive clustering framework (available at http://source.ijs.si/ktclus/clus-public/). The tree induction algorithm was modified to introduce an option node and explore several alternative splits when selecting the best split is not an easy task, i.e., when there are multiple candidate splits with similar heuristic scores.

The empirical comparison of option PCTs to standard PCTs and ensembles of PCTs was performed on benchmark datasets for the following structured output prediction problems: multi-target regression (MTR), multi-label classification (MLC), and hierarchical multi-label classification (HMLC). We evaluated the algorithms on predefined train-test splits of the datasets where they were provided and used 10-fold cross-validation otherwise. We varied the number of option nodes introduced in option PCTs and studied how their performance and interpretability is affected.

We compared the predictive performance (using task-specific performance measures) as well model size (number of leaf nodes) and learning time. For the MTR datasets, we also calculated bias-variance decomposition of the mean squared error (Geman et al., 1992) to investigate the source of prediction errors. To compare and present the results of the evaluation of option PCTs and (ensembles of) standard PCTs, we used the statistical analysis suggested in (Demšar, 2006) and average ranking diagrams. While newer protocols to compare methods based on Bayesian statistics have been proposed (Corani et al., 2017), their properties make them a bad fit for our experiments. Specifically, they are computationally much more demanding, only work well on very many datasets (30-50), and provide no approach for comparing multiple methods (require pairwise comparisons).

To develop oblique PCTs, we additionally studied existing proposed oblique decision tree methods (Breiman et al., 1984; Menze et al., 2011; Murthy et al., 1994; Prabhu & Varma, 2014) and analyzed their strengths and weaknesses. Oblique splits require a fundamentally different tree representation and split optimization and were developed in a new software library because of that. The library is named *spyct* and it is implemented in the `Python` programming language. It is available under a free license, and compatible with the popular scikit-learn API (Pedregosa et al., 2011) to lower the barrier for user adoption. We implemented two variants for learning oblique splits:

- The support vector machine (SVM) variant first clusters the examples into two groups based on their output values. This clustering represents the 'ideal' splitting of examples. However, the actual test to split the examples must be based on the inputs. After the 'ideal' clustering is determined, we label each example with 0 or 1 based on the cluster it belongs to. We then learn a linear SVM that tries to predict these labels based on the inputs. The weights the SVM learns are used to define the test hyperplane for the oblique split.

- The gradient descent variant directly optimizes the hyperplane to minimize the impurity of outputs of examples on each side. We achieve this by calculating weighted variances of examples on each side of the hyperplane using fuzzy membership weights. This function is differentiable and can be efficiently minimized with gradient-based methods.

Both variants improve the scaling of learning time with the number of descriptive and clustering variables. They can also exploit sparse input and output spaces by using a sparse matrix format to store the data and perform matrix operations. We also developed bagging and random forest ensembles of oblique PCTs.

We evaluated the proposed oblique PCTs in both single-tree and ensemble setting on benchmark MTR, MLC, and HMLC datasets. Since the method is novel also for primitive output prediction tasks, we also performed an empirical comparison on single-target regression, binary classification, and multi-class classification datasets. We first performed experiments on a smaller set of datasets for each task to investigate the influence of different methods' parameters on its predictive performance and learning time. We determined default values that offer a solid trade-off between performance and efficiency. Next, we compared the oblique PCTs to standard PCTs, ensembles of PCTs, and other state-of-the-art methods for different predictive modeling tasks. To estimate the predictive performance of the learned models, we used 10-fold cross-validation and task-specific performance evaluation measures. We also compared model sizes and learning times.

Additionally, we used the same experimental design as described above to evaluate the performance and efficiency of oblique PCTs on SSL tasks and compare them to semi-supervised PCTs. This was done because the improved scaling of oblique PCTs is especially useful for the semi-supervised approach.

Finally, we learned oblique PCTs on datasets with added random features and investigated feature importance scores obtained with our algorithm. These experiments were designed to show whether the obtained scores are meaningful and whether the oblique PCTs are resilient to irrelevant features.

1.4 Contributions

The scientific contributions of the dissertation are summarized as follows:

1. A method for learning option predictive clustering trees that are capable of addressing structured output prediction (SOP) tasks.

2. An empirical evaluation of option PCTs on benchmark datasets for SOP and their comparison to standard PCTs and ensembles thereof.

3. SVM-based and gradient-descent-based methods for efficient learning of oblique PCTs capable of addressing SOP tasks and handling high dimensional and sparse data.

4. An empirical evaluation of oblique PCTs and ensembles thereof on benchmark datasets for standard classification and regression tasks as well as for SOP tasks.

5. An empirical comparison of oblique PCTs and standard PCTs on semi-supervised learning tasks.

Papers presenting these contributions have been published or submitted to conferences or journals. The details related to the publications are listed in the Bibliography section at the end of the thesis.

1.5 Organisation of the Thesis

This first section provided a general perspective and context of the thesis. The remainder is organized as follows. Chapter 2 presents the background and work related to the thesis in detail. This includes the description of predictive modeling in general and the specific tasks we considered. We present predictive clustering trees in detail and provide an overview of existing approaches for learning option trees and oblique trees.

Chapter 3 then presents the core ideas developed in the thesis. It provides a general overview of the developed methods for learning option PCTs and oblique PCTs, their inner workings, and their instantiations for different learning settings: different structured output prediction tasks, as well as supervised and semi-supervised learning. It also contains the theoretical analysis of their computational complexities.

Chapters 4-8 present the contributions of this thesis in detail. These chapters include papers (either published or under review) that present the proposed methods and their extensive experimental evaluations. Each chapter begins with a summary of the work presented in the paper included in the specific chapter.

Option predictive clustering trees come first. Chapter 4 treats option PCTs for multi-target regression, Chapter 5 presents how they are used for multi-label classification, and Chapter 6 how they work for hierarchical multi-label classification. Each chapter also includes all the relevant evaluation experiments.

Next, Chapter 7 describes both the SVM and the gradient descent variants of oblique PCTs, presents the results of their extensive evaluation for 6 predictive modeling tasks, analyzes the influences of different parameters, and the feature importance scores that can be extracted from the learned trees. Chapter 8 then presents how oblique PCTs can be learned in a semi-supervised manner and presents the experimental evaluation of the approach.

Finally, Chapter 9 concludes by discussing the contributions to science made with this thesis and outlines directions for future work.

Chapter 2

Background

In this chapter, we describe in detail the background for the thesis. We start by presenting the basics of predictive modeling (Section 2.1). We differentiate between *primitive output prediction* (Section 2.1.1) and *structured output prediction* (Section 2.1.2) based on the type of the output. The naming is derived from the International Organization for Standardization (ISO) naming of the datatypes (International Organization for Standardization, 2007) and the Generic ontology of datatypes (Panov, Soldatova, et al., 2016). Primitive datatypes are defined axiomatically and do not reference other datatypes (e.g., numeric and nominal values), whereas generated (or structured) datatypes are defined in terms of other datatypes (e.g., vectors of numeric values, sets of nominal values). We also present the difference between supervised and semi-supervised learning (Section 2.1.3). Next, we discuss the predictive clustering trees as a generalization of standard decision trees and their use in the predictive modeling tasks we introduced (Section 2.2). Finally, we present the related work on option trees (Section 2.3) and oblique trees (Section 2.4), the two main concepts that our work in this thesis develops further in the context of predictive clustering trees and structured output prediction.

2.1 Predictive Modeling

The goal in predictive modeling is to learn a model that predicts an output for a given example described with its input variables. In this thesis, we focus on tabular data, where each example is described with a fixed number of features. The features can be either numeric, meaning their domain is a subset of $\mathbb{R}$, or nominal, meaning their domain is a finite set of predefined values. In the simplest case, the output is a single target variable (or target) that can again be numeric or nominal. Features are traditionally denoted as x_i and the target as y. The data is called tabular because it can be naturally presented as a table, where rows correspond to examples and columns contain the values of the features and the target. Table 2.1 shows an example of a data table where examples are described with D features. Features x_1 and x_D are numeric, whereas feature x_2 and target y are nominal.

The learning algorithm constructs a model with the help of a learning set (also called a training set), which contains learning (training) examples with their associated targets. However, the model should be able to make accurate predictions for examples that the algorithm did not see during learning, i.e., it should generalize well. This predictive performance is measured with a performance measure that compares the model's predictions to real target values. For example, with nominal targets, we can look at the percentage of correctly classified examples. With numeric targets, we can calculate how far the predictions are from real values on average. Various performance measures have been proposed

Table 2.1: Example of a tabular dataset where each example is described with D numeric or nominal features and a binary target variable.

x_1	x_2	...	x_D	y
5.12	blue	...	25	yes
2.67	green	...	51	no
3.91	red	...	12	no
⋮				
4.48	green	...	29	yes

that are appropriate for problems with different characteristics (Kocev et al., 2013; Madjarov et al., 2012; Vens et al., 2008; Witten & Frank, 2002). The performance measure should be determined in advance, as it influences the selection of the learning algorithm or its parameters.

There are two standard strategies for estimating the predictive performance of a model. The first strategy is to split the available data into a training set and a test set. The model is learned on the training set and then used to make the predictions for the test set. The predictions are then compared to the real target values. This process can be repeated several times for different selections of the training and test sets to get more accurate estimations of the performance. The other strategy is called k-fold cross-validation, where the data is split into k parts (folds) of equal sizes (or as close to it as possible). One fold is selected for the test set and the union of the other folds as the training set. This is repeated k times so that each fold is selected for the test set once.

The predictive performance of the model is not necessarily the only criterion for evaluating the models. In many domains, it is important to have insight into the model to understand why it made a given prediction – the model needs to be *interpretable*. Interpretability typically improves the confidence in the predictions in sensitive applications (e.g., medical diagnosis) and supports the discovery of new knowledge. It can be seen from two viewpoints: interpreting the model as knowledge extracted from the data (for this, the model size can play an important role) and interpreting the predictions made by the model. For the former, one needs models that are readily interpretable (e.g., decision trees) or proxies such as feature ranking estimated from the model (Petković et al., 2019a; Petković et al., 2020). For the latter, one can use specifically designed methods for explaining individual predictions (Štrumbelj & Kononenko, 2014), feature importance estimation for making predictions for subsets of examples (Petković et al., 2019b), or novel methods such as SHAP (Lundberg & Lee, 2017) that are capable of explaining individual predictions for arbitrary models (note that they are still better suited to tree-based methods (Lundberg et al., 2020)).

Another important consideration can be computational efficiency, both for learning the model and using the model to make predictions. Better learning efficiency can allow the algorithms to process more learning data, which typically leads to more accurate models. Moreover, for some domains with large numbers of examples, features, and outputs, the computational cost of learning can be a limiting factor (for example, analysis of metagenomic data, virtual compound screens). Several problems also require models capable of making predictions very quickly (e.g., automated stock trading).

Some algorithms, like many decision tree implementations, can deal with both numeric and nominal variables. Others, like neural network methods, only accept numeric variables, which means nominal variables need to be encoded with one or more numeric variables. This can be done in a number of ways, and we here present the two most commonly used

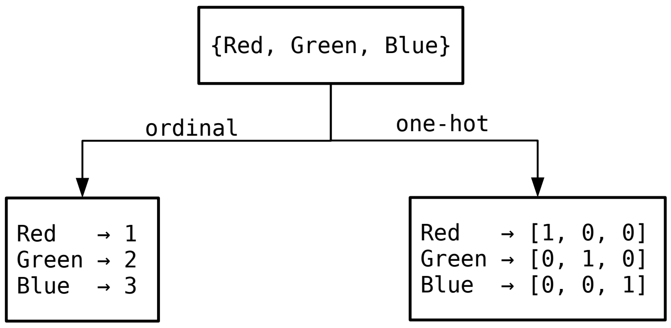


Figure 2.1: An example of ordinal and one-hot encoding strategies of a nominal variable with three possible values: red, green and blue.

strategies. Suppose we have a nominal variable with n possible values that we organized in an arbitrary order $[v_1, v_2, \ldots, v_n]$. In *ordinal encoding*, we encode each value of the nominal variable with a single integer, e.g., we can encode v_i with integer i. The second common encoding strategy is *one-hot encoding*, where the values are encoded with binary vectors of length n. Each component corresponds to a specific value, and all components are set to 0 except the component corresponding to the value that the vector encodes, which has a value of 1. For example, we encode the value v_i with an n-dimensional vector, where all components are 0 except the i-th one, which is 1. Figure 2.1 demonstrates ordinal and one-hot encodings on a simple example.

2.1.1 Primitive output prediction

The simplest predictive modeling tasks are the ones with a single target variable which has a primitive data type (Panov, Soldatova, et al., 2016). Depending on whether the target is numeric or nominal, we differentiate between different predictive modeling tasks.

If the target is nominal, the task is called *classification*, and the different values of the target are referred to as *classes*. For example, the target in Table 2.1 has two possible values, “yes” and “no”. Classification tasks are further divided based on the number of classes. When there are only two possible classes, like in the example, the task is known as *binary classification* (BC). And with three or more possible classes, we are referring to the task as *multi-class classification* (MCC). Some examples of classification tasks are predicting the winning team in a basketball match, predicting whether to buy or sell a stock, detecting the language of a text, identifying the animal in an image, etc.

Alternatively, if the target is numeric, the task is called *regression*. Examples of regression tasks are predicting the point difference in a basketball match, predicting the price increase/decrease of a stock, predicting the temperature tomorrow, predicting how much a molecule inhibits a protein, etc. As we hinted at with our examples, regression tasks are often naturally transformed into classification tasks by discretizing the numeric target into several bins and treating the different bins as classes. For example, predicting the point difference in a basketball match can be transformed into a classification problem of predicting which team will win. In the tasks presented above, the target is a single value. We will refer to this group of tasks as *primitive output prediction* tasks.

One can also encode a nominal target variable in the same way as the descriptive variables. We can thus approach a classification problem with regression algorithms. This is mostly used for binary classification problems, where the two classes can be encoded with values 0 and 1. Then the output of a regression model can be interpreted as the

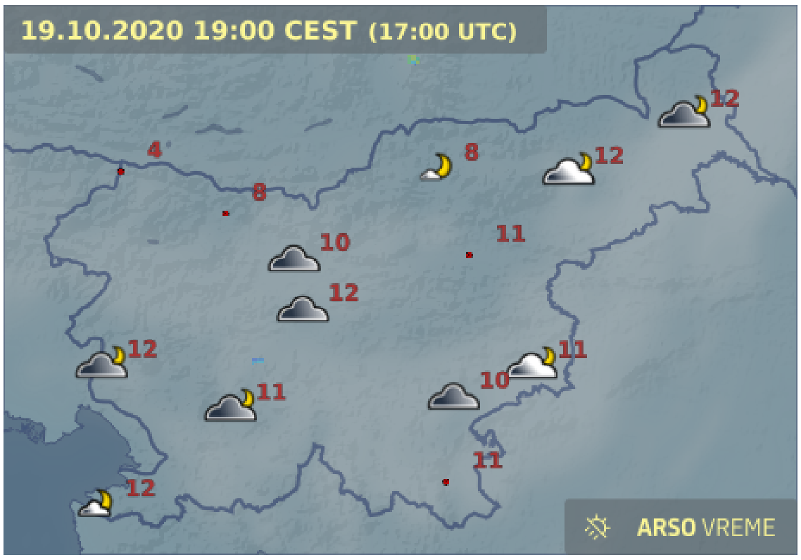


Figure 2.2: Weather forecasts can be a source of multi-target regression problems. For example, we may wish to predict the temperatures or atmospheric pressures for the next day at several locations. Source: meteo.arso.gov.si.

pseudo-probability (propensity) of an example belonging to class 1.

2.1.2 Structured output prediction

In the previous section, we discussed predictive modeling tasks where the output consisted of a single target variable. In this section, we will consider tasks that are more naturally described with more complex outputs, consisting of multiple target variables that may have an additional structure defined on top of them. Specifically, we will describe the tasks multi-target regression, multi-label classification, and hierarchical multi-label classification.

2.1.2.1 Multi-target regression

In the regression task described above, there was a single numeric target for the model to predict. From now on, we will refer to this task as single-target regression (STR) for clearer distinction. In multi-target regression (MTR), the task is to predict multiple numeric targets simultaneously. Some examples of multi-target regression problems are predicting the point differential of multiple basketball games (each game featuring our favorite team and some other opponent), predicting the temperature on multiple locations (e.g., Figure 2.2), predicting the height and weight of a person, etc. Additionally, multi-class classification problems can be treated as multi-target regression if we encode the nominal target with one-hot encoding. In the end, we predict the class for which the value predicted by the MTR algorithm was the largest.

A straightforward approach to MTR is problem transformation, where we transform the problem of predicting t numeric targets into t independent STR problems. This allows us to use any STR method to deal with MTR. However, by doing so, we overlook the potential interactions among the different targets (e.g., temperatures at neighboring locations are similar, if our team beats team A, it may be more likely to also beat team B, tall people tend to weigh more than short people, etc.). Another way to approach MTR is to use algorithms designed or adapted for dealing with multiple numeric targets. Such methods

Figure 2.3: Image tagging is typically a multi-label classification problem. This image was tagged with labels nature, mountain, water, lake, forest, and park. Source: hdwallpapers.in.

can exploit correlations among the targets. By building a single model, we can also benefit from improved time-complexity and interpretability (Kocev et al., 2013).

2.1.2.2 Multi-label classification

In binary and multi-class classification problems, examples are labeled with one label out of two or more options. In *multi-label classification* (MLC), examples are labeled with a subset of a finite set of possible labels. For example, we can look at the image tagging problem (Figure 2.3). Each image can be tagged with multiple labels, and hundreds of potential labels exist. Some other examples of MLC problems are topic classification of texts, movie genre classification, protein-protein interaction prediction, etc.

Label sets in MLC are most often encoded as *k-hot* binary vectors. It is similar to one-hot encoding we discussed in Section 2.1, except that multiple components can have value 1 because multiple labels can be present for each example. For many problems, the number of possible labels is very large, but for most examples only a very small subset of them is relevant. This leads to high-dimensional vectors that are *sparse* – the vast majority of components have a value 0.

Multiple problem transformation approaches to MLC have been proposed (Tsoumakas et al., 2009). Suppose we have a problem with L possible labels. The simplest approach is to construct L binary classifiers, each of which decides if a single label is present or not. This approach is known as binary relevance and its disadvantage is that it ignores any interactions between the labels. Another approach is to learn $L(L-1)/2$ models, one for each pair of labels. Each model decides which of the two labels is more appropriate, and individual predictions are then aggregated into the final decision. This improves upon binary relevance by taking some label interactions into account but is considerably more computationally intensive.

MLC can also be transformed into a multi-class classification problem by treating each combination of labels as a separate class. The disadvantage of this approach is that the number of classes grows exponentially with the size of the label set which makes all classes very rare. Most classes are not represented often in the learning set (if at all), and label sets not present in the learning set cannot be predicted.

As with MTR, algorithms also exist that were designed or adapted to be used for MLC

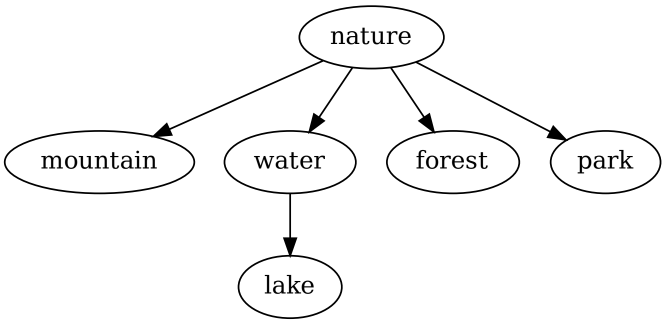


Figure 2.4: Example of a hierarchy constructed from the labels of the image presented in Figure 2.3.

directly. Some examples are predictive clustering trees (Kocev et al., 2020; Kocev et al., 2013), support vector machines (Elisseeff & Weston, 2002) and neural networks (Chen et al., 2019).

2.1.2.3 Hierarchical multi-label classification

Hierarchical multi-label classification (HMLC) is a generalization of the MLC task where a partial ordering among the labels is defined, i.e., some labels are special cases of other labels. This partial ordering organizes the labels in a hierarchy and imposes a hierarchical constraint: if an example has a label l, it also has all the parent labels of l in the hierarchy (the labels that generalize l). Predictive models can take advantage of this constraint.

The image tagging task we presented as an example of an MLC task can be posed as an HMLC task if we organize the labels into a hierarchy. Figure 2.4 presents one possible hierarchy constructed from the labels of Figure 2.3. It is just a part of the hierarchy that would be constructed from all the possible image labels. In this case, the hierarchy is in the form of a tree – each label has one parent. Partial ordering could also induce a hierarchy in the form of a directed acyclic graph, where a label can have multiple parents. Other examples of HMLC problems are classification of living beings into taxonomic ranks (Dimitrovski et al., 2012), classification of gene functions (Consortium, 2019), classification of medical paper topics with Medical Subject Headings (Minguet et al., 2017), etc.

A straightforward approach to HMLC is to ignore the hierarchy and treat it as an MLC problem. In the end, the predictions made with such an approach must be post-processed to take into account the hierarchical constraint. There are also several approaches that can take the hierarchy into account, both local, which decompose the problem into multiple smaller ones (for example, one model for each non-terminal node in the hierarchy, that decides which children nodes are appropriate), and global, where a single model is made to deal with all the labels. Local approaches can be used with any classification algorithm. Examples of global approaches include predictive clustering trees (Vens et al., 2008), kernel-based approaches (Rousu et al., 2006), and neural networks (Wehrmann et al., 2018).

2.1.3 Supervised and semi-supervised learning

So far, we focused on the *supervised learning* approach to predictive modeling. Here, the model is learned with the help of a learning set where each example is labeled, i.e., has a

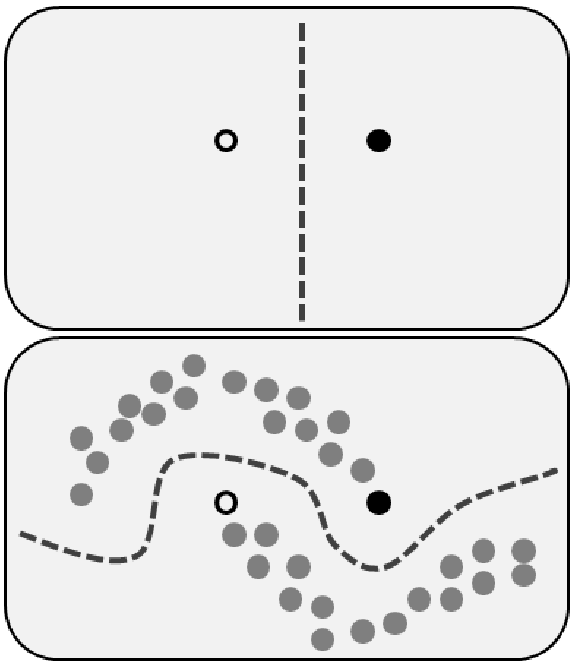


Figure 2.5: Motivation for semi-supervised learning. The information provided by the unlabeled examples (grey circles) influences the decision boundary between the two classes (black and white circles). Source: wikipedia.org.

target value assigned. The quality of the model very much depends on the quality of the learning set. If the learning examples do not represent the distribution of the data well enough, it is difficult to construct a model that generalizes well.

A problem that is present in many domains is that labeled examples can be difficult to obtain, which means that learning sets can be too small for accurate modeling. This is common for biological and chemical problems where the process of labeling examples often requires wet-lab experiments, which are time-consuming and expensive, or medical problems where purposeful labeling can be unethical.

For some problems where labeled examples are rare, examples without assigned target values (unlabeled examples) are abundant. For example, the efficiency of compounds for a specific condition is measured for a small set of compounds, while public databases exist containing millions of compounds and their descriptions. This is the setting that *semi-supervised learning* addresses. We are presented with a set of labeled examples and a set of unlabeled examples. Typically the number of labeled examples is (much) lower than the number of unlabeled examples.

Semi-supervised methods use the information about the distribution of the data in the input space provided by the unlabeled examples aiming to learn a better predictive model. The intuition is presented in Figure 2.5. Semi-supervised methods rely on some assumptions about the data. The most common assumptions are the *clustering assumption*, which assumes that points nearby in the input space have similar target values, and the *manifold assumption*, which assumes that examples lie on lower-dimensional manifolds in the input space (as is the case in Figure 2.5).

The most straightforward semi-supervised method is self-training (Rosenberg et al., 2005). It is an iterative procedure, where at each step we learn a model on the labeled examples. This model then makes predictions for the unlabeled examples, and the most confident predictions are then included among the labeled examples in the next iteration.

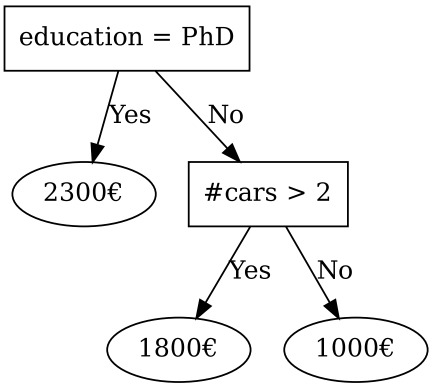


Figure 2.6: An example of a regression tree that predicts the monthly income of a person. The tree is interpreted in the following way: if a person has a PhD, they are expected to be wealthy and have a salary of 2300€ per month. Otherwise, having multiple cars indicates a higher expected salary (1800€) than having a single car or none at all.

A related approach is co-training (Zhou & Li, 2007), where, instead of a single model, we have two models that label the examples for each other. The models are usually different or are provided with different views of the data. There are also methods that take unlabeled examples into account to draw decision boundaries in low density regions in the input space. Examples include semi-supervised SVMs (Bennett & Demiriz, 1998) and semi-supervised predictive clustering trees (Levatić et al., 2017; Levatić et al., 2020).

2.2 Predictive Clustering Trees

In this section, we start by briefly presenting standard decision trees and their use in ensembles. We then describe predictive clustering trees in detail and show how they are used for structured output prediction tasks and semi-supervised learning.

2.2.1 Basics of decision trees

Tree-based predictive models have been popular for the last several decades (Breiman et al., 1984). Their use is widespread: they can solve both classification and regression tasks and can handle both numeric and nominal features. A small regression tree is presented in Figure 2.6.

Tree models are typically constructed with the recursive top-down induction algorithm outlined in Algorithm 2.1. The construction starts with the full set of learning examples. At each step, the algorithm goes through all the possible tests for all the features and selects the best one based on some heuristic score. The tests for numeric features x_i are typically of the form

$$x_i > t,$$

where $t \in \mathbb{R}$ is some threshold. For tests on nominal features, there are two common options:

$$x_i = v_i \quad \text{and} \quad x_i \in V,$$

Algorithm 2.1: Learning a decision tree: The inputs are a $N \times D$ table of features X describing the N learning examples, and a list of N targets y that are assigned to them.

```
Function grow_tree($X$, $y$):
    test = best_test($X$, $y$)
    if test $\neq$ None then
        rows1, rows2 = split($X$, $test$)
        left_subtree = grow_tree($X[rows1]$, $y[rows1]$)
        right_subtree = grow_tree($X[rows2]$, $y[rows2]$)
        return SplitNode($test$, $left\_subtree$, $right\_subtree$)
    else
        return LeafNode(prototype($Y$))

Function best_test($X$, $y$):
    best = None
    for $d = 1, \ldots, D$ do
        for test $in$ possible_tests($X$, $d$) do
            if acceptable($test$) and score($test$, $X$, $y$) > score($best$, $X$, $y$) then
                best = test
    return best

Function score($test$, $X$, $y$):
    rows1, rows2 = split($X$, $test$)
    n1 = num_rows($rows1$)
    n2 = num_rows($rows2$)
    n = n1 + n2
    return $n \cdot$ imp($y$) $- n1 \cdot$ imp($y[rows1]$) $- n2 \cdot$ imp($y[rows2]$)
```

where v_i is a specific value that this feature can have, and V is a subset of them.

The heuristic score is determined based on some measure of *impurity* of a set of examples that we aim to minimize. The main requirement for the impurity function is that it achieves its minimum on sets of examples with identical targets and increases with the diversity of the targets. For regression tasks, the most common choice is variance of the target values, i.e.,

$$\texttt{Var}(y) = \frac{1}{N}\sum_{i=1}^{N} y_i^2 - (\frac{1}{N}\sum_{i=1}^{N} y_i)^2. \tag{2.1}$$

For classification tasks, a common choice is Gini impurity, defined as

$$\texttt{Gini}(y) = \sum_{i=1}^{L} \nu_i(1-\nu_i), \tag{2.2}$$

where L is the number of classes and ν_i are the frequencies of classes in y. If we randomly choose an element of y and assign it a random class according to the class frequencies ν_i, Gini impurity is the probability the assignment would be incorrect.

The tests must also be *acceptable* in order to be considered, which depends on pre-pruning conditions we need to specify in advance. For example, we can limit the depth of the tree, require at least some number of examples in the resulting subsets of examples, or require the split to reduce the impurity by at least a certain amount. After the best acceptable test is selected, the data is split according to it, and the construction continues recursively on each subset of examples. If no acceptable test is found, a leaf node is constructed, where a prototype is stored, which is later used to make predictions. The selection of the prototype can depend on the performance measure that is used to evaluate the predictions of the model. For classification, using the majority class (i.e., the class with the highest frequency among the examples in the leaf) maximizes the probability that a randomly selected example from the leaf will be classified correctly. For regression, using the mean target value minimizes the mean squared error between the prototype and the examples in the leaf.

Because the tests only have two possible outcomes, the resulting trees are binary. The greediness of the algorithm means that the constructed tree is usually not optimal, however, this approach is necessary to keep the tree construction computationally manageable. Finding the optimal tree for a set of examples is a NP-hard problem (Laurent & Rivest, 1976).

Once a tree is constructed, it can be used to make predictions for previously unseen examples. A new example is first sorted down to a leaf node, based on its feature values. Then, the prototype stored in that leaf is predicted. A single tree can be inspected, and its predictions interpreted easily. However, tree learning algorithms are typically unstable, meaning that a small change in the learning set can lead to a completely different tree. This instability hurts the predictive performance of tree models (Breiman, 1996; Geman et al., 1992).

This problem is addressed by tree ensembles: instead of using a single tree to make predictions, we construct multiple trees and use them together. Each tree in the ensemble makes its own prediction, and the predictions of the individual trees are then aggregated into the final prediction of the ensemble. Typical aggregations are majority voting for classification problems and averaging for regression problems.

To profit from an ensemble, we need variety among the individual trees – if all the trees were the same, the ensemble prediction would be equal to the predictions of the individual trees. The diversity of the trees is achieved by introducing randomization in the learning process.

The original tree ensembles are *bagging ensembles* (Breiman, 1996), where each tree is learned on a different subset of the learning set, which is sampled with replacements (bootstrapping). Because tree learning is unstable, these changes to the learning sets lead to different trees, which benefits the ensemble. Nowadays, most commonly used ensembles are *random forests* (Breiman, 2001). In addition to bootstrapping the learning sets, they introduce randomization also in the individual tree construction by only considering a subset of features when searching for a test to split the data. This subset is drawn randomly for each split.

The performance boost gained by the ensembles comes at the cost of reduced interpretability. Ensembles often have 100 or more trees, and inspecting them individually is infeasible. To counter this, different feature ranking approaches based on tree ensembles have been developed (Breiman, 2001; Petković et al., 2019a; Petković et al., 2020). A feature ranking shows how much an ensemble relies on individual features to make predictions.

2.2.2 Basics of predictive clustering trees

Predictive clustering trees (PCTs) (Blockeel et al., 2002; Blockeel et al., 1998) generalize standard decision/regression trees by differentiating between three types of variables.

- **Features** are the variables that can be used in the internal nodes of the tree to make

Algorithm 2.2: Learning a PCT: The inputs are a $N \times D$ table of features X, a $N \times T$ table of targets Y, and a $N \times K$ table of clustering attributes Z.

```
Function grow_tree(X, Y, Z):
    test = best_test(X, Z)
    if test ≠ None then
        rows1, rows2 = split(X, test)
        left_subtree = grow_tree(X[rows1], Y[rows1], Z[rows1])
        right_subtree = grow_tree(X[rows2], Y[rows2], Z[rows2])
        return SplitNode(test, left_subtree, right_subtree)
    else
        return LeafNode(prototype(Y))

Function best_test(X, Z):
    best = None
    for $d = 1, \ldots, D$ do
        for test in possible_tests(X, d) do
            if acceptable(test) and score(test, X, Z) > score(best, X, Z) then
                best = test
    return best

Function score(test, X, Z):
    rows1, rows2 = split(X, test)
    n1 = num_rows(rows1)
    n2 = num_rows(rows2)
    n = n1 + n2
    return $n \cdot$ imp(Z) $- n1 \cdot$ imp(Z[rows1]) $- n2 \cdot$ imp(Z[rows2])
```

decisions. They are used in the same way as in standard decision trees.

- **Targets** are the variables that are predicted in the leaves of the tree. The important difference compared to standard decision trees is that there can be multiple targets, instead of just one. This requires a change in the prototype function.
- **Clustering attributes** are the variables that are used by the heuristic scoring function to evaluate the splits and therefore guide the tree construction. In standard decision trees, there is no difference between clustering attributes and the targets. In PCTs, they are theoretically independent from the features and the targets. However, since we want the tree to make accurate predictions for the targets in the leaves, some connection to the targets is desired. This requires a change in the impurity function.

Algorithm 2.2 presents a step-by-step procedure for constructing a PCT. It is very similar to Algorithm 2.1 but takes into account the differences we mentioned above. There can be multiple targets and the `best_test` and `score` functions use the clustering variables instead of the targets.

This differentiation of variables, support for multiple targets, and various instantiations of prototype and impurity functions give PCTs a lot of flexibility. They have been used for several structured output prediction tasks (Kocev et al., 2020; Kocev et al., 2013; Madjarov et al., 2012), semi-supervised learning (Levatić et al., 2017; Levatić et al., 2020; Levatić et al., 2018), and also unsupervised learning (Dimitrovski et al., 2016). We also explored using low-dimensional embeddings of label sets as clustering attributes to reduce the learning time of the models (Stepišnik & Kocev, 2020a).

2.2.3 PCTs for structured output prediction

In this section, we will present the impurity and prototype functions that PCTs use to address MTR, MLC, and HMLC tasks. The general form of the impurity that PCTs calculate is

$$\texttt{imp}(Z) = \frac{1}{K}\sum_{i=1}^{K} p_i \frac{\texttt{imp}(Z_i)}{\texttt{imp}(Z_i^{\text{train}})}, \tag{2.3}$$

where $\texttt{imp}(Z_i)$ is the impurity of the i-th clustering attribute, $\texttt{imp}(Z_i^{\text{train}})$ is the impurity of that attribute on the entire training dataset, and p_i is the priority weight we assign to this attribute. In other words, the total impurity is the weighted sum of relative impurities of the individual clustering attributes. The impurities are considered relative to the entire learning set to put the impurities of different variables on the same scale. For individual impurities, we use variance for numeric variables and Gini impurity for nominal variables.

In a supervised setting, the clustering attributes are typically the same as the targets. For the MTR task, assuming we do not prioritize some targets over others, this means that the impurity used is

$$\texttt{imp}(Z) = \texttt{imp}(Y) = \frac{1}{T}\sum_{i=1}^{T} \frac{\texttt{Var}(Y_i)}{\texttt{Var}(Y_i^{\text{train}})}. \tag{2.4}$$

Note that when there is only one numeric target, this is equivalent to the splitting criterion used by standard regression trees. For the prototype function, we simply use the vector of mean target values of the training examples in that leaf.

For MLC, the approach is similar to MTR. We represent the label sets with binary vectors using k-hot encoding (Section 2.1.2.2), so the number of targets is equal to the number of different labels. For binary variables, Gini impurity and variance are equivalent as impurity measures[1]. After the encoding, the tree construction is the same for MLC as it was for MTR. For the prototypes, we again use the averages of the target values, which in this case represent label frequencies in the leaves. These averages can be used directly for predictions and treated as pseudo-probabilities of individual labels or used to rank the labels in order of relevance for a given example. Alternatively, we can apply some threshold (typically 0.5) to the averages and only predict the labels for which the computed average exceeds the selected threshold.

The HMLC task is again approached in a similar way to MLC, except we also take into account that labels higher in the hierarchy are more important than labels lower in the hierarchy. Label sets are represented with binary vectors, and variance is used to measure the impurity for the individual labels. However, we also make use of weights to prioritize selected labels. The impurity used is

$$\texttt{imp}(Y) = \frac{1}{T}\sum_{i=1}^{T} p_i \frac{\texttt{Var}(Y_i)}{\texttt{Var}(Y_i^{\text{train}})}, \tag{2.5}$$

where $p_i = p_0^{\alpha_i}$ is the priority weight assigned to the i-th label, α_i is the depth of this label in the hierarchy, and p_0 some initial priority. If $p_0 < 1$, the importance of a label drops exponentially with its depth. Predictions are made in the same way as for MLC. Because the hierarchical constraint is satisfied for all learning examples, it is also satisfied for the prototypes calculated in the leaves – the frequency of a given label will always be lower than the frequency of its ancestors in the hierarchy.

[1]Simple exercise for the reader, it follows from the fact that $v^2 = v$ for $v \in \{0, 1\}$.

2.2.4 PCTs for semi-supervised learning

As we discussed in Section 2.1.3, semi-supervised methods look to exploit labeled and unlabeled examples to learn better models in situations where labeled examples are scarce. We start with a $N \times D$ table X_l containing the features of the N_l labeled examples, a $N_l \times T$ table containing the targets assigned to them, and a $N_u \times D$ table X_u containing the features of the N_u unlabeled examples. Suppose we concatenate the tables with features of labeled and unlabeled examples into a $(N_l + N_u) \times D$ table X.

The way that PCTs include the information from unlabeled examples in the learning process is by using both targets and features as the clustering attributes (Levatić et al., 2018). When calculating the impurity of individual clustering attributes, only labeled examples are taken into account for the targets ($\mathtt{imp}(Y_i)$). Following this information, Equation 2.3 transforms into

$$\mathtt{imp}(Z) = \frac{1-\omega}{D}\sum_{i=1}^{D} p_i^X \frac{\mathtt{imp}(X_i)}{\mathtt{imp}(X_i^{\text{train}})} + \frac{\omega}{T}\sum_{i=1}^{T} p_i^Y \frac{\mathtt{imp}(Y_i)}{\mathtt{imp}(Y_i^{\text{train}})}, \tag{2.6}$$

where p_i^X and p_i^Y are the priority weights assigned to the feature and target subsets of the clustering attributes, respectively, and $\omega \in [0, 1]$ is the parameter that determines the degree of supervision. There are three settings for ω to consider.

- If $\omega = 1$, the part of the impurity that is derived from the features is ignored because it is weighted with $1-\omega$. Since the impurity for the targets is only calculated only on the labeled examples, the unlabeled examples are effectively ignored when evaluating the splits and the tree construction is completely supervised.

- If $\omega \in (0, 1)$, both features and targets influence the impurity and learning is in fact semi-supervised.

- In the second extreme case, where $\omega = 0$, the impurities of targets are ignored. This means the tree is constructed in an unsupervised manner and examples are clustered together based on their feature values. However, in the leaves, the targets are still used to calculate the prototypes.

2.2.5 Time complexity of learning a split

Here, we analyze the time complexity of learning a split in PCTs. Suppose we have N examples with D features, T targets, and K clustering attributes. When searching for a split, we iterate over all D features and evaluate all the different splits by that feature. For each feature, we start by sorting the examples according to the values of that feature. This requires $O(N \log N)$ operations. Then we can evaluate all the tests for this feature in a single pass over the examples, but we need to keep track of some statistics needed to calculate the impurity for each clustering attribute. This takes $O(NK)$ operations. When the best test is found, it only takes $O(N)$ operations to split the data according to it, which is negligible. The total cost of learning a split is therefore $O(DN \log N + DNK)$ (Kocev et al., 2013).

Remember that for normal supervised learning $K = T$. This means that learning time scales with the product of the numbers of features and targets. Especially for MLC and HMLC problems, the number of labels can often reach hundreds and even thousands. If coupled with a high-dimensional input, the learning times become prohibitively long. Furthermore, for semi-supervised learning, clustering attributes include both features and targets, so $K = D + T$. Therefore, the learning time scales quadratically with the number of features, which is also problematic.

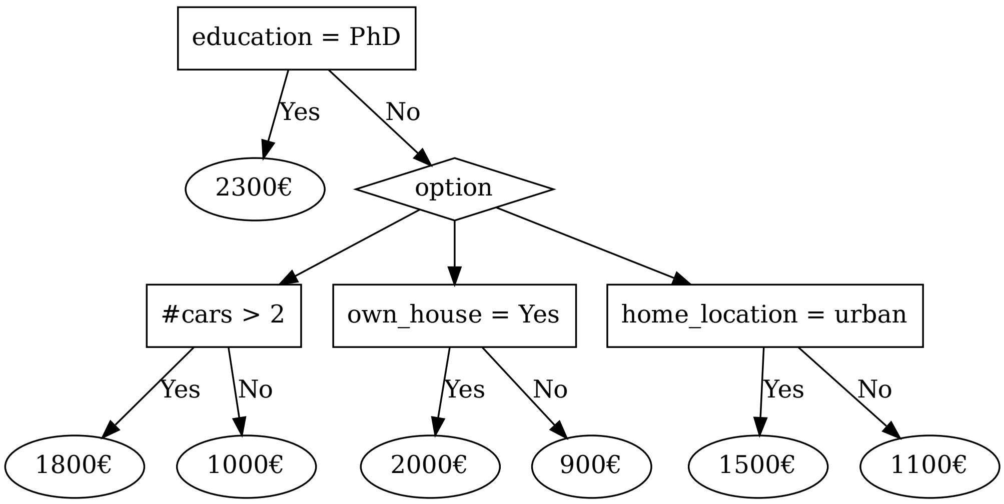


Figure 2.7: An example of an option decision tree predicting the monthly income of a person. Suppose we have a person without a PhD that owns 3 cars, owns a house, and lives in the countryside. The first test sends this person to the right subtree where it encounters an option node. Option nodes send examples down all its subtrees, each of which makes a prediction. In this case, the first subtree predicts the income of 1800€ because the number of cars owned is at least 2, the second tree predicts 2500€ because the person owns a house, the third subtree predicts 1100€ because the person does not live in the city. The prediction of the option tree is the mean of these options: 1800€.

2.3 Option Decision Trees

Option trees were first introduced by Buntine (1992). They generalize standard decision trees by allowing option nodes in addition to normal split and leaf nodes. When looking for a test to split the data, we can find multiple tests with heuristic scores similar to the best test. Instead of only selecting the best test and ignoring the rest, option trees create an option node, which combines several of the best splits. Each of the selected splits then continues into its own alternative subtree. An important note is that when considering the tests for an option node, only tests based on different features are taken into account. Having similar heuristic scores for multiple tests using the same feature (especially a numeric feature where the threshold can be slightly changed) is expected so such cases are not included in an option node.

Option trees make predictions in a manner similar to standard decision trees. Examples start at the top of the tree and are sorted downwards based on the tests in the split nodes. When an option node is encountered, the example is sent to each of the alternative subtrees. This means that an example can reach multiple leaf nodes where predictions are made. The prediction of the option tree is then the aggregation (average or majority class) of these individual predictions. Figure 2.7 presents an example of an option tree.

Compared to standard decision trees, option trees have two main advantages.

- The tree construction algorithm presented in Algorithm 2.1 is greedy and selects the test that appears the best at the moment, disregarding future options. If we have several tests with similarly good heuristic scores, the test with the highest score based on the learning set might only appear so due to some noise in the data or a couple of outliers. By creating an option node and keeping several good alternatives, we reduce the greediness and increase the chance of keeping the actually most fitting test.

- Because the tree construction is unstable, a small change in a test close to the root node leads to great changes lower in the tree. On the one hand, this means that selecting the wrong test early in the tree construction can be very harmful in standard trees. On the other hand, we can expect substantial differences among the alternative subtrees constructed from an option node. Like with ensembles, this is helpful when aggregating their predictions.

Option trees can be seen as a compact way of presenting a set of trees with common prefixes. In this sense, they are a bridge between single trees and tree ensembles. Increasing the number of options leads to better predictive performance but comes with an increased computational cost and lower interpretability.

In the experiments performed by Buntine (1992), option nodes were most often created lower in the trees where the uncertainty of split evaluation was greater due to smaller data samples. In a later study, Kohavi and Kunz (1997) analyzed option trees in detail and showed that option nodes closer to the root of the tree improve the performance more than option nodes lower in the tree. They go on to propose multiple strategies for limiting the number of option nodes. The first strategy is to only create option nodes when multiple tests have heuristic scores within some predefined relative margin of the best score. The second strategy limits option nodes to the first couple of levels of the tree. With the third strategy, each option node is only allowed to include up to a specified number of options. Buntine (1992) only used the first strategy and included all options that were within a factor 0.005 of the best test. Kohavi and Kunz (1997) increased this factor greatly, up to 0.5, but additionally used the other two strategies to limit the size of the trees.

2.4 Oblique Decision Trees

All the trees we have seen and discussed thus far used only a single feature in each test. This means that the split boundaries are hyperplanes that are parallel to one axis in the input space, which is why such trees are called *axis-parallel* trees. In contrast, *oblique* trees (also called multivariate trees) use linear combinations of features in the tests. Individual tests have the form

$$w \cdot x + b > 0,$$

where $x \in \mathbb{R}^D$ is the vector of features, $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the weights and the bias term defining the split hyperplane, and $w \cdot x$ is the scalar product of vectors. This makes splits more general because the boundary can be an arbitrary hyperplane in the input space. The increased flexibility can lead to models that fit the data much better, as we demonstrate in Figure 2.8. It also means that nominal features need to be encoded with numerical features in one of the ways we described in Section 2.1 and Figure 2.1.

However, despite this advantage, oblique trees receive significantly less attention from the machine learning community than axis-parallel trees. One reason is that oblique splits are more difficult to optimize. With axis-parallel trees, finding the optimal tree is NP-hard, but finding the optimal split for the greedy algorithm is easy, because we can iterate over all options. With oblique trees, even optimizing a single split is difficult. Finding a hyperplane that splits a set of examples with binary labels in a way that minimizes misclassified examples on each side of the hyperplane was shown to be an NP-hard problem (Heath,

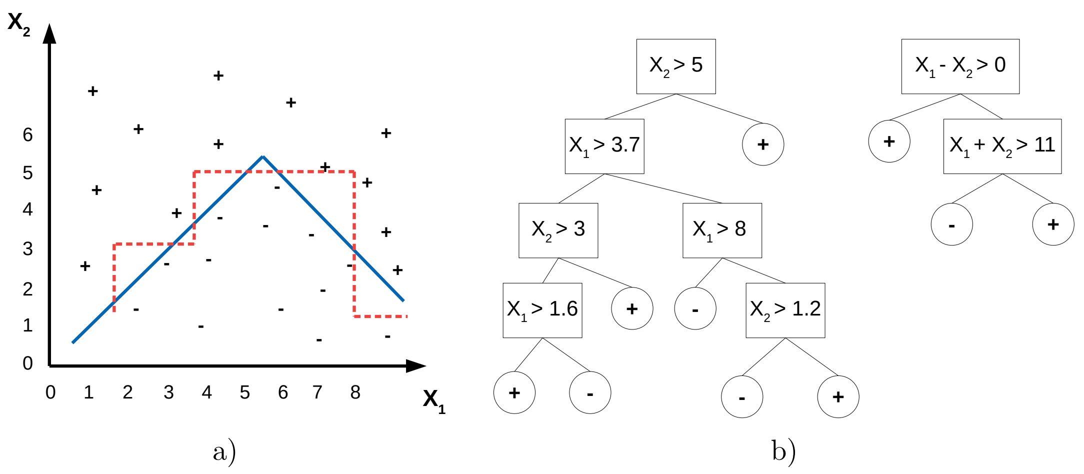


Figure 2.8: A toy dataset (a) with drawn decision boundaries learned by axis-parallel (red, dashed) and oblique (blue, solid) decision trees (b).

1993). Additionally, oblique splits also lose some of the interpretability of axis-parallel splits, especially if the number of features they use is large.

We found that existing oblique tree methods are specialized for specific predictive modeling tasks (mostly binary classification) and are often computationally inefficient. The initial proposals for the induction of oblique trees (Breiman et al., 1984; Murthy et al., 1994) relied on local search optimization. Such approaches scale poorly to contemporary problems that have thousands of examples and/or features. More recently, Yang et al. (2019) proposed Weighted Oblique Decision Trees that learn splits by optimizing weighted information entropy. They are relatively efficient and can be used for binary and multi-class classification tasks.

Oblique random forests (Menze et al., 2011) follow the standard random forest paradigm. They learn individual trees on different bootstrapped samples of the training set and use a different random subset of features for each split. Split learning is performed via ridge regression, which allows for efficient optimization of the hyperplanes, but their approach is limited to binary classification problems.

Additionally, FastXML (Prabhu & Varma, 2014) and PfastreXML (Jain et al., 2016) are methods based on oblique trees for multi-label classification. The focus of the methods is on problems with thousands of features and labels, so they specialize in working with sparse data. When learning a split, these methods optimize a complex objective function that combines L1 regularization, logarithmic loss, and normalized distributed cumulative gain (nDCG) of examples on each side of the hyperplane. The criterion function is optimized in two steps. First, the examples are partitioned in a way that minimizes nDCG in each partition; this partitioning is improved iteratively. Then, a hyperplane is learned that approximates this partitioning as best as it can. This is done by using GLMNET method to solve a logistic regression problem.

Chapter 3

Complex Nodes in Trees: the Basics

In this chapter, we present the core ideas behind the developed variants of predictive clustering trees with complex nodes. We first focus on developing option nodes and then on developing oblique splits. Our goal is to keep the developed methods applicable to all the tasks that PCTs could address originally, as presented in Section 2.2. The proposed methods extend PCTs in orthogonal directions - one allowing a more extensive exploration of the search space of possible splits, and the other by changing the split search space from the standard univariate splits to the space of multivariate splits. Conceptually, combining option nodes with oblique splits is straightforward. However, the methods we developed for optimizing oblique splits are poorly suited to this because they do not provide alternative splits that could be included in an option node. The details on the use of the developed variants for different structured output prediction tasks and their experimental evaluations can be found in Chapters 4-8.

3.1 Option Predictive Clustering Trees

Option predictive clustering trees (OPCTs) extend standard PCTs with option nodes. As we described in Section 2.3, option decision trees create option nodes when there are multiple tests with heuristic scores close to the maximum score. The main differences between PCTs and standard decision trees are the heuristic scoring and prototype functions. OPCTs use the same heuristic scores and prototypes as PCTs. Since the heuristic score used by PCTs still returns a single real value, we can use the same criterion for the introduction of option nodes as option decision trees. The prototype function does not interfere with option nodes in any way.

Algorithm 3.1 outlines the OPCT induction procedure. There are two main differences from the standard algorithm (Algorithm 2.2). First, the `best_test` function now finds the best test for each feature. Let t_{best} be the heuristic score of the overall best test (across all features). Among the best tests for individual features, the function then returns all tests t for which

$$\frac{\texttt{score}(t)}{\texttt{score}(t_{best})} \geq 1 - \varepsilon \cdot d^{level}, \tag{3.1}$$

where ε is the margin parameter, *level* is the current depth of the node in the tree and d the decay rate. The decay rate determines how fast the margin for the inclusion of tests in an option node is shrinking. This parameter is included so we can be more liberal with options higher in the tree where they have a greater impact and stricter lower in the tree to limit the tree size. Additionally, we can limit the maximum number of options per option node to O. This means only the best O tests satisfying the condition in Equation 3.1 are included. We can also limit option nodes to only the top L levels of the tree. If that is

Algorithm 3.1: Learning an OPCT: The inputs are a $N \times D$ table of features X, a $N \times T$ table of targets Y, and a $N \times K$ table of clustering attributes Z.

```
Function grow_tree(X, Y, Z):
    tests = best_test(X, Z)
    splits = [ ]
    for test ∈ tests do
        rows1, rows2 = split(X, test)
        left_subtree = grow_tree(X[rows1], Y[rows1], Z[rows1])
        right_subtree = grow_tree(X[rows2], Y[rows2], Z[rows2])
        splits.add(SplitNode(test, left_ subtree, right_ subtree))

    if length(splits) > 1 then
        /* If there are multiple splits, create an option node.      */
        return OptionNode(splits)
    else if length(splits) = 1 then
        /* If there is only one split, use it.                       */
        return splits[0]
    else
        return LeafNode(prototype(Y))

Function best_test(X, Z):
    best_feature_tests = [ ]
    for $d = 1, \ldots, D$ do
        best = None
        for test in possible_tests(X, d) do
            if score(test, X, Z) > score(best, X, Z) then
                best = test
        best_feature_tests.add(best)
    return almost_best(best_feature_ tests)

Function score(test, X, Z):
    rows1, rows2 = split(X, test)
    n1 = length(rows1)
    n2 = length(rows2)
    n = n1 + n2
    return $n \cdot$ imp(Z) $- n1 \cdot$ imp(Z[rows1]) $- n2 \cdot$ imp(Z[rows2])
```

the case, the function only returns the overall best test when $level > L$. This is equal to setting $O = 1$.

The second major difference is dealing with the results of the `best_test` function. A tree is grown for each test that was returned. If there was more than one, an option node is created containing all the alternative splits. If there was just one test, the algorithm creates a standard split node. Otherwise, no acceptable splits were found and a leaf node is created.

Once an OPCT is learned, the predictions are made in the same way they are made with option decision trees. The prototypes in the leaves are also the same as the ones used by PCTs and depend on the predictive modeling task that the tree is addressing.

Constructing an option node takes no additional time compared to a normal split node because the most time-consuming part is the evaluation of all the possible tests, which is done in both cases. However, the learning time of the entire tree is increased because the tree continues to grow from each available alternative split in a given option node. In the worst case (computationally), we introduce option nodes wherever possible at the first L levels, each including the maximum O options allowed. From that point on, we essentially construct O^L standard PCTs. On larger datasets, the total depth of the learned trees is typically significantly greater than L, so the worst-case learning time of an OPCT is similar to the learning time of a bagging ensemble of O^L standard PCTs.

3.2 Oblique Predictive Clustering Trees

In this section, we present our proposal for learning predictive clustering trees with oblique splits, named SPYCTs [1]. The development of oblique splits required bigger changes to the induction algorithm compared to OPCTs. While we kept the standard top-down recursive structure, we had to design a completely different way of optimizing the tests to split the data. The `grow_tree` function from Algorithm 2.2 remains the same, however the `best_test` function was redesigned to optimize oblique splits.

We will propose two variants for optimizing oblique splits in Section 3.2.1 and Section 3.2.2, but to start with, let us introduce the notation used throughout this chapter. As discussed in Section 2.4, oblique splits require the nominal features to be encoded with numerical features. The data can then be represented as matrices with real-valued elements. Let $X \in \mathbb{R}^{N\times D}$ be the matrix containing the D features of the N examples in the learning set, $Y \in \mathbb{R}^{N\times T}$ be the matrix containing the T targets associated with the N examples in the learning set, and $Z \in \mathbb{R}^{N\times K}$ be the matrix containing the C clustering attributes associated with the N examples in the learning set. Let $p \in \mathbb{R}^K$ be the priority vector containing the weights assigned to the clustering attributes (see Section 2.2 on PCTs for details on the usefulness of these weights). To include the semi-supervised learning scenario in the following definitions, we allow missing values in these matrices. The labeled and unlabeled examples can then be mixed together with the unlabeled examples, simply having missing values for targets in Y and Z (if targets are included among clustering attributes).

Let $M \in \mathbb{R}^{R\times C}$ be an arbitrary matrix. We define a missing value indicator matrix $I^M \in \{0,1\}^{R\times C}$ as a binary matrix with dimensions of matrix M where

$$I^M_{i,j} = \begin{cases} 0 & \text{if } M_{i,j} \text{ is missing,} \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

Analogously, let I^v be the missing value indicator for a vector v. The matrices and vectors can have any values stored in place of missing values, we can check in the indicator matrices/vectors to see if a value is real or a placeholder. Below, $M_{i,:}$ will refer to the i-th row and $M_{:,i}$ to the i-th column of the matrix M. Let $\texttt{colmean}(M)$ be a vector of column mean values of the matrix M calculated only from the non-missing values:

$$\texttt{colmean}(M_j) = \frac{\sum_{i=1}^{R} I^M_{i,j} M_{i,j}}{\sum_{i=1}^{R} I^M_{i,j}}, \quad 1 \leq j \leq C. \tag{3.3}$$

Oblique splits are defined by a general hyperplane in the input space. We parametrize the hyperplane with a vector of weights $w \in \mathbb{R}^D$ and a bias term $b \in \mathbb{R}$. We will refer to the subset of examples where $X_{i,:} \cdot w + b > 0$ as the positive subset, and the subset of examples

[1]Starting from PCTs, we added PY because they are implemented in Python, and SP because the initial motivation was efficiency on sparse data.

where $X_{i,:} \cdot w + b \leq 0$ as the negative subset. The goal is to find a hyperplane where the examples on each side have similar values of the clustering attributes (in Z). The details on the optimization of the splits are presented a little later, for now, we continue with the general overview of the algorithm.

After the split hyperplane has been optimized, the tree construction continues recursively on the positive and negative subsets. Similar to standard PCTs, the learning stops when the learned split is not acceptable, and a leaf node is made, where the prototype of the targets is stored. SPYCTs use the same prototypes as standard PCTs, which for numeric and encoded data are mean values of the targets over the (labeled) examples in the leaf, i.e., `colmean`(Y).

When making a prediction for a new example with features $x \in \mathbb{R}^D$, we calculate the value $x \cdot w + b$ in the root node. If it is non-negative, we repeat the process in the positive subtree, otherwise in the negative subtree. This is done until a leaf node is reached, where the stored prototype is predicted.

3.2.1 SVM-based split learning

The first proposal for learning oblique splits is based on linear support vector machines (SVMs). Here, we present the general formulation of the optimization problem, for special cases applicable specifically to supervised and semi-supervised structured output prediction problems see Chapter 7 and Chapter 8, respectively. The SVM variant performs the hyperplane optimization in two steps. In the first step, we group the examples into two subsets in such a way that the similarity of the clustering attributes in each subset is maximized, regardless of the values of the features. Therefore, only the matrix Z is needed for this step. To obtain the groups, we use k-means clustering (Hartigan & Wong, 1979) to cluster the rows (examples) in Z into two clusters.

For the initial centroids, two different rows in Z are randomly selected, i.e., $c^0 = Z_{i,:}$ and $c^1 = Z_{j,:}$ for some $1 \leq i, j \leq N$. We then calculate the weighted Euclidean distances between each row in Z and the two centroids, weighted with the priority weights p and again disregarding any missing values. Specifically, the distance between the i-th row and a centroid c is defined as

$$\mathtt{dist}(c, i) = \frac{\sum_{k=1}^{K} p_k I^Z_{i,k} I^c_k (Z_{i,k} - c_k)^2}{\sum_{k=1}^{K} p_k I^Z_{i,k} I^c_k}. \tag{3.4}$$

Next, we sort each row into the cluster based on which centroid is closer to it. Let s_i denote the cluster assigned to the i-th row, i.e., $s_i = \operatorname{argmin}_j \mathtt{dist}(c^j, i)$. The new centroids are then column mean values of examples in each cluster. This process is repeated for a fixed number of iterations.

After obtaining the final cluster assignments, we search for a hyperplane in the input space that approximates this partitioning. In essence, we convert the split hyperplane optimization problem into a binary classification problem. We solve the following problem:

$$\min_{w,b} \; \|w\|_1 + C \sum_{i=1}^{N} \max(0, 1 - s_i(X_{i,:} \cdot w + b))^2,$$

where parameter $C \in \mathbb{R}$ determines the strength of regularization. To solve this problem, we use the LIBLINEAR library (Fan et al., 2008), specifically their $L1$-regularized $L2$-loss support vector classification. This provides the w, b parameters that define the split.

3.2.2 Gradient-descent-based split learning

Unlike the SVM variant, where the split is optimized indirectly by first determining an ideal split and then trying to approximate it with a hyperplane, the gradient descent variant directly searches for a hyperplane split that minimizes the impurity on each side. We initialize the weight vector w randomly, then set $b = \text{median}(-Xw)$ so that the examples are initially split in half.

Next, we will define the optimization criterion for the split learning problem. Let $s = \sigma(Xw + b) \in [0,1]^N$, where the sigmoid function σ is applied component-wise. The vector s contains values from the $[0,1]$ interval, and we treat it as a fuzzy membership indicator. Specifically, the value s_i tells us how much the i-th example belongs to the positive subset, whereas the value $1 - s_i$ tells us how much it belongs to the negative subset.

To measure the impurity of the positive subset, we use the weighted variance of each column (variable) in Z, and we weigh each row (example) with its corresponding weight in s. To measure the impurity of the negative subset, we again use weighted column variance, this time with weights $1 - s$. Again, we ignore any missing values when calculating the variance. We define the weighted variance of j-th column in Z with arbitrary weights $a \in \mathbb{R}^{\mathbb{N}}$ as

$$\texttt{var}(Z, j, a) = \frac{\sum_{i=1}^{N} I_{i,j}^{Z} a_i (Z_{i,j} - \texttt{mean}(a, Z_{:,j}))^2}{\sum_{i=1}^{N} I_{i,j}^{Z} a_i}, \tag{3.5}$$

where

$$\texttt{mean}(a, Z_{:,j}) = \frac{\sum_{i=1}^{N} I_{i,j}^{Z} a_i Z_{i,j}}{\sum_{i=1}^{N} I_{i,j}^{Z} a_i}, \tag{3.6}$$

is the weighted mean of the j-th column in Z.

The impurity of a subset of examples is the weighted sum of weighted variances over all the clustering attributes. For weighing different attributes, we use the priority vector p. The impurity of the positive subset is then $\texttt{imp}(Z, p, s) = \sum_{j=1}^{L} p_j \texttt{var}(Z, j, s)$, and similarly $\texttt{imp}(Z, p, 1 - s)$ is the impurity of the negative subset. The split quality is estimated as

$$f(w, b) = S * \texttt{imp}(Z, p, s) + (N - S) * \texttt{imp}(Z, p, 1 - s), \tag{3.7}$$

where $s = \sigma(Xw + b)$ and $S = \sum_{i=1}^{N} s_i$. The terms S and $N - S$ represent the sizes of positive and negative subsets. This mirrors the heuristic score for evaluating the splits in standard PCTs (see Section 2.2).

Finally, we add regularization of the hyperplane weights, so the final optimization problem is

$$\min_{w,b} \; \|w\|_{\frac{1}{2}} + Cf(w, b),$$

where C again controls the strength of regularization and $||w||_{\frac{1}{2}} = (\sum_{i=1}^{D} \sqrt{|w_i|})^2$ is the L$\frac{1}{2}$ norm. We selected L$\frac{1}{2}$ regularization because it induces weight sparsity more aggressively than L1. Regularization makes the algorithm pick the features it uses for a split more carefully, which improves the interpretability of splits and reduces the model size.

The final optimization criterion is differentiable, which enables the use of efficient optimization methods. For this purpose, we selected the Adam gradient descent method (Kingma & Ba, 2014).

3.2.3 Time complexity analysis

In this section, we analyze the time complexities of learning oblique splits with both SPYCT variants. In the SVM variant, the first step is to perform a 2-means clustering on the clustering data $Z \in \mathbb{R}^{N \times K}$. In each iteration, we have to calculate the distances between the centroids and the examples, which requires $O(NK)$ operations. If I_c denotes the number of clustering iterations, the total cost of clustering is $O(I_c NK)$. Learning the linear SVM to differentiate the clusters costs $O(NDI_o)$, where I_o is the number of optimization iterations. The total cost is then $O(N(KI_c + DI_o))$.

For the gradient descent variant, the main operation is gradient calculation. We vectorize the operations whenever possible. The most expensive parts are the multiplication of X with w to get the fuzzy indicators s and the multiplication of Z with s when calculating the derivative of the objective function by s. These products require $O(ND)$ and $O(NK)$ operations, respectively. If I_o again denotes the number of optimization iterations, the time complexity of learning a split is $O(I_o N(D + K))$.

In comparison, the time complexity of learning a split in standard PCTs is $O(DN \log N + NDK)$ (Section 2.2). The important difference we can notice is that standard PCTs scale with DK, whereas both proposed variants of oblique PCTs scale with $D + K$. This is an important difference when addressing problems with many clustering attributes, i.e., high-dimensional output for supervised learning, and high-dimensional input or output for semi-supervised learning.

Additionally, the optimization of both SPYCT variants can exploit sparsity in data. When a feature or clustering value is 0, we can skip most operations it is involved in. Let $\hat{D} \ll D$ and $\hat{K} \ll K$ be the average numbers of non-zero elements per row in matrices x and Z, respectively. Then, the learning complexity reduces to $O(N(\hat{K}I_c + \hat{D}I_o))$ for the SVM variant and $O(NI_o(\hat{D} + \hat{K}))$ for the gradient descent variant.

3.2.4 Ensembles of oblique predictive clustering trees

Like other tree-based methods, SPYCTs can be used in ensembles to achieve better performance. In fact, since even individual oblique trees are difficult to interpret visually, we believe that SPYCTs are most naturally used in ensembles. The use of SPYCTs in both bagging and random forest ensembles is straightforward. For bagging ensembles, we simply construct each oblique PCT on a different bootstrapped sample of the learning set. For random forests, we additionally learn each split hyperplane on a random subset of features. After the hyperplane weights are learned, the weights w_i for the features that were not selected are set to 0. When making predictions, we again average the prototypes predicted by each tree in the ensemble to get the final prediction.

Chapter 4

Option Predictive Clustering Trees for Multi-Target Regression

In this chapter, we detail the first contributions of the thesis: option predictive clustering trees and their use for the multi-target regression task. In the introductory sections, we provided a high-level overview of OPCTs (Section 3.1) and described the MTR task (Section 2.1.2), where the goal is to predict multiple numeric targets. Specifically, the contributions in this context are:

1. Extension of PCTs with option nodes in the MTR context.
2. Extensive experimental evaluation of the proposed method, focusing on the influence of different parameter settings on the predictive performance, time complexity, and interpretability.
3. Bias-variance decomposition of the error for a deeper insight into the predictive performance and statistical analysis of the results.

The initial work was presented at the DS-2016 conference (Osojnik et al., 2016) and included the proposal of OPCTs for MTR aiming to address the myopia of standard PCTs. The experimental evaluation showed that OPCTs outperform single PCTs and achieve performance comparable to ensembles of PCTs (bagging and random forest). It is worth pointing out, however, that the selected parameter settings resulted in OPCTs with many option nodes which lead to learning times and model sizes on par with ensembles.

We later extended this work in a paper presented in the journal Computer Science and Information Systems (Stepišnik, Osojnik, et al., 2020), which we include in this chapter. This extension includes a formal analysis of the time complexity of learning OPCTs, and a more extensive experimental evaluation, including the bias-variance decomposition of errors. We focused the evaluation on the influence of the number of option nodes on the predictive performance and total model size, which increases the learning time and reduces the interpretability. We selected two parameter configurations: one very liberal, introducing many option nodes, and one with stricter conditions for including options. The configuration with many option nodes is similar to bagging ensembles in terms of predictive performance, model size, and learning time. The configuration with fewer option nodes presents a middle-ground between single trees and ensembles: it achieves significantly better performance than single trees at the cost of increased model sizes. The models are still smaller compared to ensembles (and the previously discussed OPCT configuration). However, they also trade some performance in exchange.

The bias-variance decomposition was performed for a wide spectrum of parameter configurations. The results show that including more options in an OPCT reduces the variance component of the model, whereas the bias component is rarely affected in a significant way. This means that adding option nodes to OPCTs has a similar effect to adding trees to tree ensembles.

From the hypotheses we defined in Section 1.2, this chapter addresses the following:

H1 Introducing few option nodes in a PCT improves its predictive performance compared to standard PCTs while retaining the interpretability potential.

H2 Introducing many option nodes in a PCT makes the option PCTs exhibit similar behavior to bagging ensembles of PCTs in terms of time complexity, size, and predictive performance.

Our paper confirmed H1 and H2 for the MTR task with the analysis of the two parameter configurations we described and the bias-variance decomposition. Random forest ensembles have similar performance and model sizes as large OPCTs, however, they have an advantage in time complexity. This advantage is achieved through feature sub-sampling when selecting a split, but this change is orthogonal to the introduction of option nodes – option trees can be combined with feature sub-sampling seamlessly.

The paper included in this Chapter is:

- Stepišnik, T., Osojnik, A., Džeroski, S., Kocev, D. (2020). Option predictive clustering trees for multi-target regression. *Computer Science and Information Systems*, 17(2), 459–486.

The contributions of Tomaž Stepišnik to this paper are as follows. He designed and carried out the experiments for bias-variance decomposition, processed the results, and performed statistical analysis on them. He wrote the sections describing the experimental results and revised the paper following the feedback from the reviewers.

Computer Science and Information Systems 17(2):459–486 https://doi.org/10.2298/CSIS190928006S

Option predictive clustering trees for multi-target regression[⋆]

Tomaž Stepišnik[1,2], Aljaž Osojnik[1,2], Sašo Džeroski[1,2], and Dragi Kocev[1,2]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
{tomaz.stepisnik,aljaz.osojnik,saso.dzeroski,dragi.kocev}@ijs.si

Abstract. Decision trees are one of the most widely used predictive modelling methods primarily because they are readily interpretable and fast to learn. These nice properties come at the price of predictive performance. Moreover, the standard induction of decision trees suffers from myopia: a single split is chosen in each internal node which is selected in a greedy manner; hence, the resulting tree may be sub-optimal. To address these issues, option trees have been proposed which can include several alternative splits in a new type of internal nodes called option nodes. Considering all of this, an option tree can be also regarded as a condensed representation of an ensemble. In this work, we propose to learn option trees for multi-target regression (MTR) based on the predictive clustering framework. The resulting models are thus called option predictive clustering trees (OPCTs). Multi-target regression is concerned with learning predictive models for tasks with multiple numeric target variables. We evaluate the proposed OPCTs on 11 benchmark MTR data sets. The results reveal that OPCTs achieve statistically significantly better predictive performance than a single predictive clustering tree (PCT) and are competitive with bagging and random forests of PCTs. By limiting the number of option nodes, we can achieve a good trade-off between predictive power and efficiency (model size and learning time). We also perform parameter sensitivity analysis and bias-variance decomposition of the mean squared error. Our analysis shows that OPCTs can reduce the variance of PCTs nearly as much as ensemble methods do. In terms of bias, OPCTs occasionally outperform other methods. Finally, we demonstrate the potential of OPCTs for multifaceted interpretability and illustrate the potential for inclusion of domain knowledge in the tree learning process.

Keywords: multi-target regression, option trees, interpretable models, predictive clustering trees, bias-variance decomposition of error.

1. Introduction

Supervised learning is one of the most widely researched and investigated areas of machine learning. The goal in supervised learning is to learn, from a set of examples with known class, a function that outputs a prediction for the (scalar-valued) class of a previously unseen example. However, in many real life problems of predictive modelling the output (target) is structured, e.g. it is a vector of classs values or a tuple of target variables. There can be dependencies between the class values/targets (e.g., they can be organized

[⋆] The authors wish it to be known that the first two authors should be regarded as joint first authors.

into a tree-shaped hierarchy or a directed acyclic graph) or some internal relations between the class values may exist (e.g., as in sequences).

In this work, we concentrate on the task of predicting multiple numeric variables. Examples thus take the form $(\mathbf{x_i}, \mathbf{y_i})$, where $\mathbf{x_i} = (x_{i1}, \ldots, x_{ik})$ is a vector of k input variables and $\mathbf{y_i} = (y_{i1}, \ldots, y_{it})$ is a vector of t target variables. This task is known under the name of *multi-target regression* (MTR) [4, 23, 24] (also known as multi-output or multivariate regression). MTR is a type of structured output prediction task which has applications in many real life problems, where we are interested in simultaneously predicting multiple numeric variables. Prominent examples come from ecology and include predicting the abundance of different species living in the same habitat [12] and predicting properties of forests [21, 29]. Due to its applicability to a wide range of domains, this task is recently gaining increasing interest in the research community.

Several methods for addressing the task of MTR have been proposed [24, 31]. These methods can be categorized into two groups of methods [2]: (1) local methods, that predict each of the target variable separately and then combine the individual model predictions to get the overall model prediction and (2) global methods, that predict all of the variables simultaneously (also known as 'big-bang' approaches). In the case of local models, for a domain with t target variables one needs to construct t predictive models – each predicting a single target. The prediction vector (that consists of t components) of an unseen example is then obtained by concatenating the predictions of the multiple single-target predictive models. Conversely, in the case of global models, for the same problem one needs to construct only one model. In this case, the prediction vector of an unseen example is obtained by passing the example through the model and getting its (complete) prediction.

In the past, several researchers proposed methods for solving the task of MTR directly and demonstrated their effectiveness [1, 8, 10, 21, 24, 20, 30]. The global methods have several advantages over the local methods. First, they exploit and use the dependencies that exist between the components of the structured output in the model learning phase, which can result in better predictive performance. Next, they are typically more efficient: it can happen that the number of components in the output is very large (e.g., predicting the bioactivity profiles of compounds described with their quantitative structure-activity relationships on a large set of proteins), in which case executing a basic method for each component is not feasible. Furthermore, they produce models that are typically smaller than the sum of the sizes of the models built for each of the components.

The state-of-the-art methods for MTR are based on tree and ensemble learning [8, 24, 27, 31]. Trees for MTR (from the predictive clustering framework) inherit the properties of regression trees: they are interpretable models, but their construction is greedy. The performance of trees is significantly improved when they are used in an ensemble setting [24, 20]. However, the myopia, i.e., greediness, of the tree construction process can lead to learning sub-optimal models. One way to alleviate this is to use a beam-search algorithm for tree induction [22], while another approach is to introduce option splits in the nodes [9, 25].

Beside the many appealing properties of trees, their predictive performance is often limited. In a variety of machine learning tasks including MTR, learning ensembles of trees typically significantly improve the predictive power of the single models [24]. However, learning ensembles has a significant drawback – lack of a possibility to interpret the obtained predictive model. It is not feasible to analyze each tree in an ensemble due to both the number of trees and the randomized procedure that is used to learn them. Typically, this means that there is a trade-off between interpretability and predictive performance.

To address this issue, we propose to extend predictive clustering trees (PCTs) for MTR towards option trees, i.e., we introduce option predictive clustering trees (OPCTs). An option tree can be seen as a condensed representation of an ensemble of trees which shares a common substructure. More specifically, the heuristic function for split selection can return multiple values that are close to each other within a predefined range. These splits are then used to construct an option node. For illustration, see Figure 1.

The contributions of this work can be summarized as follows:

– We introduce a new method for addressing the task of MTR that provides a balance between interpretability and predictive performance: An option PCT can be treated as an ensemble, or the best subtree can be extracted and analyzed as a single tree.
– We analyze the computational complexity of the proposed method and compare it to the complexity of the competing methods.
– We perform an empirical evaluation of the performance of the proposed method along three dimensions: predictive power, time efficiency and size of the models.
– We further analyse the predictive performance by examining the bias-variance decomposition of the models' errors. We perform the analysis for the proposed method as well as the competing methods. As far as we are aware, such an analysis has not been performed before for MTR methods.

This work extends our earlier work presented in [26] along two major directions. First, we perform a theoretical analysis of the computational complexity of the proposed method and compared it to the computational complexity of learning a single PCT and ensembles of PCTs. Second, we calculate the bias-variance decompositions of mean squared errors to determine which components of the error are addressed by different parameter selections and competing methods.

The remainder of this paper is organized as follows. Section 2 proposes the algorithm for learning option PCTs for MTR. Next, Section 3 outlines the design of the experimental evaluation and the details on the bias-variance decomposition of the error. Section 4 continues with a discussion of the results. Finally, Section 5 concludes and provides directions for further work.

2. Option predictive clustering trees

The predictive clustering trees framework views a decision tree as a hierarchy of clusters. The top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [3], which is available for download at http://clus.sourceforge.net.

PCTs are induced following the *top-down induction of decision trees* (TDIDT) algorithm [7]. There are however two major differences: the heuristic score and the prototype function are instantiated by the PCT algorithm based on the task that is being addressed (classification, regression or multi-target prediction etc.). The heuristic score used for selecting the tests in the internal nodes of a regular PCT is the maximization of the variance reduction resulting from the partitioning of the instances into subtrees corresponding to the outcome of the tests. By doing so, we also maximize cluster homogeneity, and, consequently, improve the predictive performance. The PCT algorithm also defines the prototype functions used in each tree leaf to give predictions for new examples based on the task at hand (e.g., uses averaging for MTR).

Option predictive clustering trees (OPCT) extend the usual PCT framework, by introducing option nodes into the tree building procedure outlined in Algorithm 1. Option decision trees were first introduced as classification trees by [9] and then analyzed in more detail by [25]. [17] analyzed regression option trees in the context of data streams.

The major motivation for the introduction of option trees is to address the myopia of the *top-down induction of decision trees* (TDIDT) algorithm [7]. Viewed through the lens of the predictive clustering framework, a PCT is a non-overlapping hierarchical clustering of the whole input space. Each node/subtree corresponds to a clustering of a subspace and prediction functions are placed in the leaves, i.e., lowest clusters in the hierarchy. An OPCT, however, allows the construction of an overlapping hierarchical clustering. This means that, at each node of the tree several alternative hierarchical clusterings of the subspace can appear instead of a single one.

When using an OPCT for prediction on a new example, we produce the prediction by aggregating over the predictions of the alternative subtrees (overlapping clusters) the example may encounter. However, as not all parts of the tree (hierarchical clustering) are necessarily overlapping, the example may encounter only nonoverlapping (sub)clusters. In that case, we produce the prediction as we would with a regular PCT.

When using TDIDT to construct a predictive clustering tree, and in particular when partitioning the data, all possible splits are evaluated by using a heuristic and the best one is selected. However, other splits may have very similar heuristic values. The best partition could be obtained with another split as a consequence of noise or of the sampling that generated the data. In this case, selecting a different split could be optimal. To address this concern, the use of option nodes was proposed [25].

An option node is introduced into the tree when it would be hard to determine the best split, that is when the best splits have similar heuristic values. When this occurs, instead of selecting only the best split, we select several of them. Specifically, we select up to 5 splits s, called *options*, that satisfy the following

$$\frac{\mathrm{Heur}(s)}{\mathrm{Heur}(s_{best})} \geq 1 - \varepsilon \cdot d^{level},$$

where s_{best} is the best split, ε determines how similar the heuristics must be, $d \in [0, 1]$ is a decay factor and $level$ is the level in the tree of the node we are attempting to split. This equation assumes that heuristic scores are to be maximized (higher value is better). After we have determined the candidate splits, we introduce an option node whose children are split nodes obtained by using the selected splits, i.e., an option node contains options as its children. Selecting more than 5 options is possible, but, uses more resources and is not advised [25]. A drawback of this method of split selection is that if some attributes are highly correlated to the attribute that produces the best split, those attributes will likely be selected for other splits in the option node. In this case the option node will only increase the learning time.

As usual, we define the level of a node to be the number of its ancestor nodes, however, we do not count option nodes. This is motivated by the predictive clustering viewpoint, i.e., the option nodes only mark that there are overlapping clusters and not that there is an additional level of clustering.

The use of a decay factor makes the selection criterion more stringent in the lower nodes of the tree. The intuition behind this is that higher up, the split selection is more important and a larger error would be inferred by introducing a non-optimal split. However, as we get deeper into the tree, the use of a non-optimal split makes decreasing impact. This intuition also allows us to prohibit the use of option nodes on levels 3 and greater, which severely mitigates the problem of combinatorial explosion.

Outline of the entire tree building process is presented in algorithm 1. The variable *candidates* is a list of at most 5 tests with the highest heuristic values. Every test is represented as a triplet (t, h, P), where t is the actual test function, h its heuristic value and P the partitions produced by the test.

When using a small ε, e.g., $\varepsilon = 0.1$, we are selecting only options whose heuristics are within 10% of the best split. However, the use of larger ε, in the extreme case even $\varepsilon = 1$, can also be motivated through the success of methods such as random forests and ensembles of extremely randomized trees. Allowing the selection of splits whose heuristic values are considerably worse than the heuristic value of the best split might not necessarily reduce the performance of the tree, but actually increase it.

Algorithm 1 The top-down induction algorithm for option PCTs.

```
Procedure OptionPCT
Input: A data set $E$, parameter $\varepsilon$, decay factor $d$, current tree level $l$
Output: An option predictive clustering tree
  $candidates = \mathrm{FindBestTests}(E, 5)$
  if $|candidates| > 0$ then
      if $|candidates| = 1$ or $l > 2$ then
          $(t^*, h^*, \mathcal{P}^*) = candidates[0]$
          for each $E_i \in \mathcal{P}^*$ do
              $tree_i = \mathrm{OptionPCT}(E_i, \varepsilon, d, l + 1)$
          return $\mathrm{node}(t^*, \bigcup_i \{tree_i\})$
      else
          $(t_0^*, h_0^*, \mathcal{P}_0^*) = candidates[0]$
          $nodes = \{\}$
          for each $(t_i^*, h_i^*, \mathcal{P}_i^*) \in candidates$ do
              if $\frac{h_i^*}{h_0^*} \geq 1 - \varepsilon \cdot d^l$ then
                  for each $E_j \in \mathcal{P}_i^*$ do
                      $tree_j = \mathrm{OptionPCT}(E_j, \varepsilon, d, l + 1)$
                  $nodes = nodes \cup \{\mathrm{node}(t^*, \bigcup_j \{tree_j\})\}$
          if $|nodes| > 1$ then
              return $\mathrm{option\_node}(nodes)$
          else
              return $nodes[0]$
  else
      return $\mathrm{leaf}(\mathrm{Prototype}(E))$
```

Once an OPCT is built, we want to use it for prediction. In a regular PCT, it is simple to produce a prediction for a new example. It is sorted into a leaf (reached according to the splits of the tree) where a prediction is made by using a prototype function. When traversing an example through an OPCT, we behave the same when we encounter a split or leaf node. If we traverse an example to an option node, however, we clone the example for each of the options and traverse one of the copies down each of the options. This means that in an option node an example is (by proxy of its copies) traversed to multiple leaves, where multiple predictions are produced. To obtain a single prediction in an option node, we aggregate the obtained predictions. When addressing multi-target regression this is generally done by averaging all the predictions per target.

An option tree is usually seen as a single tree, however, it can also be interpreted as a compact representation of an ensemble. To generate the ensemble of the *embedded trees*, we start recursively from the root node and move in a top-down fashion. Each time we encounter an option node we copy the tree above (and in "parallel") for each of the options and replace the option node with only the option, i.e., single split. This produces one tree for each option while removing the option node in question. We repeat this procedure on all the generated trees until we are left with no option nodes. This is illustrated in Figure 1. For this reason, we will sometimes refer to option trees as pseudo-ensembles.

A given OPCT is also an extension of the PCT that would be learned on the same data. By definition, whenever we introduce an option node, we include the best split (in terms of the heuristic)[3]. In regular construction of PCTs, this is the only split we consider. Consequently, the PCT is embedded in the OPCT. We can extract it if we, in a top-down fashion, select the best option in each option node.

Let's consider a full option node, that is one where we have the full 5 options. Let's assume that there are no option nodes further down in the tree. Since we have 5 options and no options lower in the tree, we have a total of 5 embedded trees. Now, let's consider the size of the embedded ensemble, when we add two such nodes under a split node, i.e., each leaf of a split node was extended into a full option node. To construct an embedded tree we can now choose one of the 5 options when the test is satisfied and one of 5 options when it is not. This results in 25 different embedded trees. It can be inferred, that to calculate the number of embedded trees for an option node we need to sum up the number of embedded trees for each of its options. In a (binary) split node, however, we multiply the numbers of embedded trees of the subtree that satisfies the split and of the subtree that does not, to obtain the total number of embedded trees.

Given the construction constraints described above, we know that option nodes with up to 5 options can appear only on the first three levels, i.e., levels 0, 1 and 2. If we now consider a full option tree, we calculate a maximum of $(5^2 \cdot 5)^2 \cdot 5 = 5^7 = 78125$ embedded trees. However, many of these trees overlap to a large extent.

Note that a given example will not traverse the entire tree. For example, in Figure 1, if an example reaches S_1 and is traversed into the left child L_1, the same result would happen in both the first and second embedded tree. The example is "agnostic" of any option nodes in the right child of S_1. Therefore, in a general option tree, a given example will visit only up to $5^3 = 125$ leaves, as it will only traverse down one side of the tree in each split node.

[3] Not only is the best split included, other splits are compared to it to determine their inclusion in the option tree.

In other words, there are a maximum of 125 different predictions that would be aggregated in order to obtain the final prediction of an option tree constructed this way. This can be compared to a single tree, where only 1 prediction will be made for each example, or to a tree ensemble, where 1 prediction would be made for each member of the ensemble and then aggregated to produce the final prediction.

[24] show that the computational complexity of building a multi-target PCT is $\mathcal{O}((S+\log N)DN\log N)$, where N is the number of examples in the data set, D is the number of descriptive variables and S is the number of target variables. Let A be the maximum number of options in an option node (in our case, we set A to 5) and B the maximum depth at which an option node can occur (in our case, we set B to 3). When constructing an OPCT, in the worst case scenario, computationally-wise, we have option nodes at all allowed levels with the maximum number of options. From that point onward we essentially construct A^B regular PCTs. Assuming that B is only a fraction of the final tree depth (i.e., $B << \log N$), we conclude that the computational complexity of building an OPCT is A^B times greater than the computational complexity of a regular PCT.

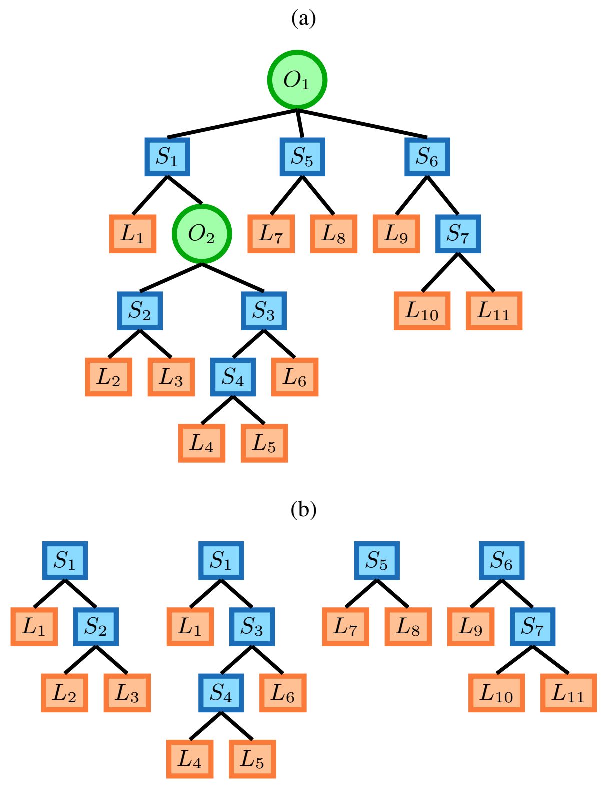

(a)
O_1
S_1 S_5 S_6
L_1 O_2 L_7 L_8 L_9 S_7
S_2 S_3 L_{10} L_{11}
L_2 L_3 S_4 L_6
L_4 L_5
(b)
S_1 S_1 S_5 S_6
L_1 S_2 L_1 S_3 L_7 L_8 L_9 S_7
L_2 L_3 S_4 L_6 L_{10} L_{11}
L_4 L_5

Fig. 1. An option tree (a) and the ensemble of its embedded trees (b). O_i are option nodes, S_j split nodes and L_k leaf nodes.

In comparison, a bagging ensemble constructs M independent PCTs on bootstrapped training sets, making their computational complexity M times that of a single PCT. Random forests, in addition to boostrapping, only consider $f(D)$ descriptive variables at each node. In the regular case, i.e., when $f(D) = \sqrt{D}$, this yields $\mathcal{O}((S+\log N)\sqrt{D}MN\log N)$ complexity. To summarize, OPCTs and bagging ensembles are constant factor-times slower (A^B and M respectively) than single PCTs, while random forests can be faster than a single PCT for data sets with large numbers of descriptive variables.

3. Experimental design

To evaluate the performance and efficiency of the OPCT method, we construct OPCTs with two parameter configurations, as well as standard PCTs and ensembles of PCTs. We first present the benchmark data sets used for evaluation of the methods and then give the specific experimental setup: parameter selections and evaluation measures.

3.1. Data description

The 11 data sets with multiple numeric targets used in this study come mainly from the domain of ecological modelling. Table 1 outlines the properties of the data sets. The selection contains data sets with various numbers of examples described with different numbers of attributes. For more details on the data sets, we refer the reader to the referenced literature, the repositories available at: http://mulan.sourceforge.net/datasets-mlc.html and http://kt.ijs.si/DragiKocev/PhD/resources/, as well as [24] and the references therein.

3.2. Evaluation of predictive performance and efficiency

We parameterize OPCTs by selecting values for the parameters ε and d. We consider two parameter configurations: $(\varepsilon = 1, d = 1)$ and $(\varepsilon = 0.2, d = 1)$. When $\varepsilon = 1$, there are no constraints on the heuristic value of the selected splits with regards to the best

Table 1. Properties of the data sets with multiple numeric targets (regression data sets): M is the number of instances, $|D|$ the number of descriptive attributes, and T the number of target attributes.

Name of data set	M	$\|D\|$	T
Collembola [18]	393	47	3
EDM [19]	154	16	2
Forestry-Kras [29]	60607	160	11
Forestry-Slivnica-LandSat [28]	6218	150	2
Forestry-Slivnica-IRS [28]	2731	29	2
Forestry-Slivnica-SPOT [28]	2731	49	2
Sigmea real [11]	817	4	2
Soil quality [12]	1944	142	3
Vegetation Clustering [16]	29679	65	11
Vegetation Condition [21]	16967	40	7
Water quality [14]	1060	16	14

test. However, since only the 5 best splits are selected, the risk that a split which would decrease the predictive performance would be selected is relatively low. This setting most resembles the ensemble setting, where greater variation is desired. The other parameter configuration provides a balance between predictive performance and efficiency. In our previous work [26], we used $\varepsilon = 0.5$ for the efficient version, but it turned out to be too close to the ensembles still. So in this paper, we selected $\varepsilon = 0.2$.

Next, we define the parameter values used in the algorithms for constructing single PCTs and ensembles of PCTs. The multi-target PCTs are obtained using F-test pruning. This pruning procedure uses the exact Fisher test to check whether a given split/test in an internal node of the tree results in a reduction in variance that is statistically significant at a given significance level. If there is no split/test that can satisfy this, then the node is converted to a leaf. An optimal significance level was selected by using internal 3-fold cross validation, from the following values: 0.125, 0.1, 0.05, 0.01, 0.005 and 0.001.

We consider two ensemble learning techniques: bagging [5] and random forests [6]. These are the most widely used tree-base ensemble learning methods. The construction of both ensemble methods takes as an input parameter the size of the ensemble, i.e., number of base predictive models to be constructed. We constructed ensembles with 100 base predictive models [24]. Furthermore, the random forests algorithm takes as input the size of the feature subset that is randomly selected at each node. For this purpose, we use the square root of the number of descriptive attributes $\lceil\sqrt{|D|}\rceil$.

We use 10-fold cross-validation to estimate the predictive performance of the used methods. We assess the predictive performance of the algorithms using several evaluation measures. In particular, since the task we consider is MTR, we employed three well known measures: the correlation coefficient (CC), mean squared error (MSE) and relative root mean squared error (RRMSE). We present here the bias-variance analysis of MSE and statistical comparison of methods according to RRMSE, where similar conclusions hold also for the other two measures. Finally, the efficiency of the proposed methods is measured with the time needed to learn a model and the size of the models (in terms of total number of leaf nodes). While OPCTs can be learned in parallel on node splitting level, our implementation does not support it yet. For this reason we used no parallelization in our experiments, to provide a fairer comparison.

In order to assess the statistical significance of the differences in performance of the studied algorithms, we adopt the recommendations by [13] for the statistical evaluation of the results. In particular, we use the Friedman test for statistical significance. Afterwards, to detect where statistically significant differences occur (i.e., between which algorithms), we use the Nemenyi post-hoc test.

We present the results of the statistical analysis with *average ranks diagrams*. The diagrams plot the average ranks of the algorithms and connect those whose average ranks differ by less than a given value, called *critical distance*. The critical distance depends on the level of the statistical significance (set in our case to 0.05). The difference in performance of the algorithms connected with a line is not statistically significant at the given significance level.

3.3. Parameter sensitivity analysis

This set of experiments is designed to provide more insight into how $varepsilon$ and d parameters affect the performance and efficiency of OPCTs. For ε we consider the values

$\{0.1, 0.2, 0.5, 1.0\}$, corresponding in order from the most stringent to the least stringent construction criterion. If we were to select $\varepsilon = 0$, the resulting OPCT would almost always directly coincide with a regular PCT, as no split would likely reach exactly the same heuristic value as the best split. Hence, the only way an option node would be induced is if two splits had the exact same heuristic value, therefore this configuration is not of interest.

As discussed above, the higher in the tree an option node is induced, the higher the variation in the learned subtrees. Induction of option nodes in lower levels of the tree not only contributes to the combinatorial explosion of the number of trees (and consequently the use of resources), but also generates less variation in the predictions, since the subtrees affected cover a smaller number of examples. Hence, we wish to curtail the number of options induced in option nodes lower in the tree. If we select a decay factor of $d = 1$, the depth of the option node induction will have no impact, while selecting a decay factor of 0.5 will effectively double the effective heuristic requirement $\varepsilon \cdot d^l$ at each level. For example, for $\varepsilon = 0.5$ and $d = 0.5$, the requirement would be 0.5 at the root level, 0.25 at the first level and 0.125 at the third level. We use the following values of the decay factor: $\{0.5, 0.9, 1\}$, these are the most to the least constrictive in terms of the construction of the OPCT.

Note that, on a given data set, all values of ε and d could produce the same OPCT, if the splits have very similar heuristic values, e.g., there could always be 5 splits that are within $10\% \cdot d^l$ of the heuristic value of the best split. Therefore, the evaluation of how ε and d affect both the predictive performance and efficiency must by design be evaluated on multiple data sets. The parameterized version of the OPCT method for a given ε and d is denoted OPCTeεdd, e.g., OPCTe0.5d0.9.

In order to facilitate replication of all the experiments, we implemented the method proposed in this paper in the latest version of CLUS, already available in the public CLUS repository at http://clus.sourceforge.net.

3.4. Bias-variance decomposition

We analyze the predictive performance of OPCTs as well as the performance of the competing methods by using bias-variance decomposition of the mean squared error [15]. The analysis allows us to investigate the source of errors of the methods. We perform it for each target separately. To calculate the decomposition, we need to predict the targets of a single data sample several times using models trained on different training sets.

Suppose we have a set of m predictive models learned using the same method on different, but related, training sets. They give m predictions for a target variable for each of the n data samples. Let y_i be the real target value of the i-th data sample and let f_{ij} be the models' prediction for y_i obtained on the j-th training set. Then the mean squared error (MSE), bias and variance can be calculated as follows:

$$\bar{f}_i = \frac{1}{m} \sum_{j=1}^{m} f_{ij}$$

$$\text{Bias} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{f}_i)^2$$

$$\text{Variance} = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}(f_{ij} - \bar{f}_i)^2$$

$$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}(f_{ij} - y_i)^2.$$

It is easy to show that MSE = Bias + Variance.

More complex models tend to have lower bias because they can better accommodate individual variations in the data, but this also increases their sensitivity to changes in the training set, leading to increased variance. Therefore, to minimize the error of the model a balance between bias and variance should be attained.

As mentioned earlier, the bias-variance decomposition procedure requires several predictions for each example obtained from models trained on different training sets using the same method. Standard 10-fold cross validation will therefore not suffice. To this end, after we split a data set into 10 folds, we select each of the folds for test set once and use the remaining non-test folds as a source of training sets. From this source we generate 20 training sets using bootstrapping and we learn predictive models on each bootstrap sample. The learned models are then applied to the examples from the test folds to obtain the multiple predictions for each data example.

4. Results and discussion

We discuss the results from the experimental evaluation along four major dimensions. First, we compare the performance of OPCTs with a single PCT and ensembles of PCTs. We compare them based on their predictive performance, learning time and size of the produced models. Next, present the effect of different parameter values on the construction of OPCTs. Third, we inspect the biases and variances of the algorithms to gain a better understanding of the comparisons of the predictive performance. Finally, we discuss the interpretability of OPCTs in juxtaposition to PCTs.

When analyzing the predictive power of algorithms in sections 4.2 and 4.1, ranking was done separately for every target of every data set, giving us a total of 59 samples. For efficiency we compared times needed to induce a model on entire data sets and the total sizes of these models. The Friedman test showed significant differences at $p < 1 \cdot 10^{-8}$ in all cases and the results of Nemeny post-hoc tests are presented as average ranking diagrams.

4.1. Predictive performance and efficiency

Figure 2 shows the results of the statistical evaluation of the predictive performance of the proposed OPCT method in comparison to a single PCT and ensembles of PCTs. First of all, we note that there is no statistically significant difference in the performance of bagging of PCTs, random forests of PCTs and OPCTe1d1. Notwithstanding this, random forests of PCTs have a slightly better average rank compared to the other two methods that are very close in rank. Second, while OPCTe0.2d1 is significantly worse than the three aforementioned methods, it is still significantly better than a single PCT.

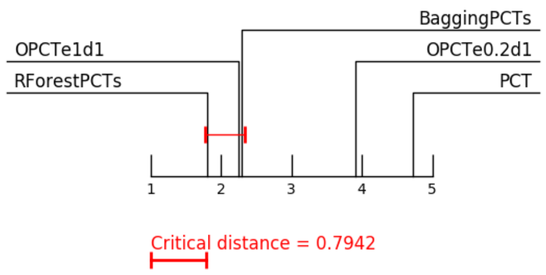
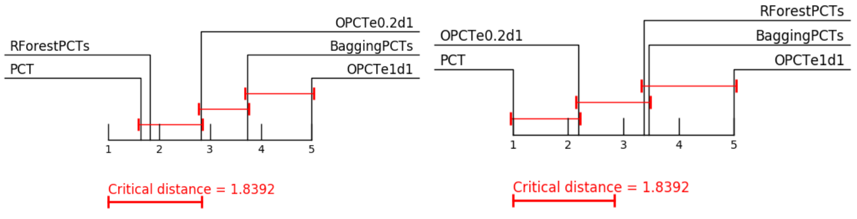


Fig. 2. Comparison of OPCTs predictive performance of OPCTs to the competing methods: Average rank diagram in terms of predictive performance.

In figure 3 we compare the methods in terms of their efficiency. It shows that learning PCTs is the most efficient method both in terms of time needed for model construction and model size, according to the average rank. However, learning OPCTe0.2d1 is not statistically significantly worse in terms of both time and size efficiency than learning a single PCT. Random forests of PCTs are close to single PCTs in terms of time, but not in terms of size, as they are significantly larger. Bagging of PCTs and OPCTe1d1 are the two slowest and largest methods.

Focusing on the efficiency of the OPCT models, we see that OPCTe1d1 is the least efficient: it produces models with the largest numbers of leaves and takes the most time for model construction. Recall that this is consistent with the analysis of computational complexity, as OPCTe1d1 can be seen as a condensed representation of what amounts to 125 PCTs, since the the algorithm selects the 5 best options at the first three levels. Furthermore, reducing the value of the ε parameter to 0.2 offers a good trade-off between predictive performance and efficiency. These models give significantly better predictive performance than a single PCT and are not significantly larger or slower to learn.



(a) Time efficiency (b) Size efficiency

Fig. 3. Comparison of the efficiency of OPCTs to that of the competing methods as measured by the time needed to learn a model and the size of the models (number of leaf nodes).

4.2. Parametrization of OPCTs

Figures 4, 5 and 6 show the average rank diagrams for predictive performance, learning time and size of different OPCTs, respectively, obtained using the experimental design outlined above. Notably, low values of both parameters lead to degradation in predictive performance. This is to be expected, since lower values of parameters force the algorithm to introduce fewer option nodes, resulting in smaller OPCTs. On the opposite side of the spectrum are the OPCTs obtained with large values of the parameters. These achieve the best predictive performance and the corresponding OPCTs are the largest in terms of the number of leaves.

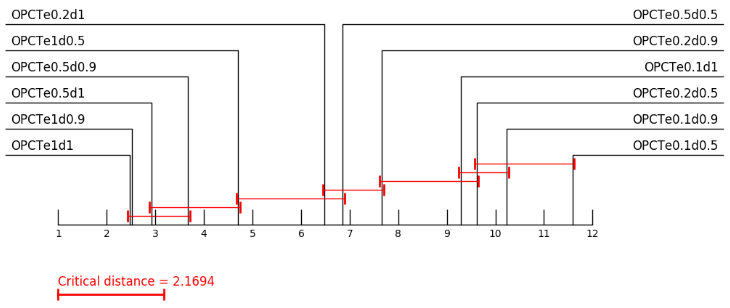


Fig. 4. Average rank diagram in terms of predictive performance for OPCTs obtained with different values of the parameters ε (e) and d.

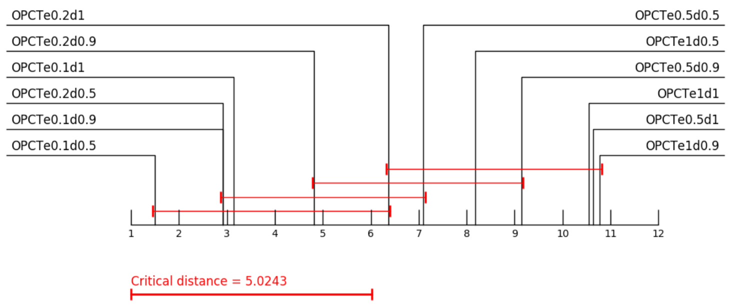


Fig. 5. Average rank diagram in terms of learning time for OPCTs obtained with different values of the parameters ε (e) and d.

Additionally, we observe that the ε parameter has a stronger influence on the performance than d. Namely, the OPCTs constructed using ε values of 0.1 and 0.2 (with the exception of OPCTe0.2d1) are the ones with the weakest predictive power. Furthermore, we also note that larger values for d also lead to better predictive performance. The best

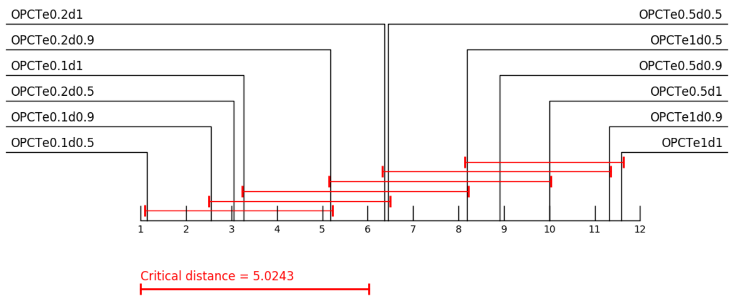


Fig. 6. Average rank diagram in terms of model size for OPCTs obtained with different values of the parameters ε (e) and d.

predictive performance is obtained when using 1 as the value of both parameters. This setting produces the largest OPCTs and requires the most learning time. Tables 2-4 in the Appendix present all values for learning times, model sizes and option node counts for different parameter values.

4.3. Bias-variance decomposition of the errors

Figures 7, 8 and 9 present the bias-variance decomposition of MSE for every target of the *Vegetation Condition*, *Sigmea real* and *Forestry-Slivnica-LandSat* data sets, respectively. The graphs for the remaining data sets are presented in Appendix.

Algorithms are presented in the following order: PCT, bagging of PCTs, random forest of PCTs, then OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1. For every target of every data set values were scaled to $[0, 1]$ by the largest MSE of all algorithms for that target, so that they can be presented on one figure. While the biases (and variances) of the different targets are presented on the same scale, only the values of different algorithms on the same target variable can be reasonably compared among each other.

The decompositions of the errors are presented on the same graph for all targets of a data set to show that they produce very similar results. That is, if a method has smaller bias (or variance) than another method on one target of a data set, it tends to have smaller bias (or variance) on the other targets of that data set as well.

The results for the *Vegetation Condition* data set given in Figure 7 represent the most common behavior. The bias of the methods is fairly similar, with less restrictive OPCTs (larger parameter values) being slightly better in this regard. PCTs have the largest variance (and consequently total error), while random forests of PCTs have the smallest. Increasing the parameter values of OPCTs also reduces the variance, bringing it on par with that of bagging and random forests.

The error decomposition for the *Sigmea real* data set given in Figure 8 stands out, since OPCTs have larger bias and variance than other algorithms for all parameter values. Increasing parameter values again reduces the variance although to a lesser extent, but bias for the second target is increased at the largest values.

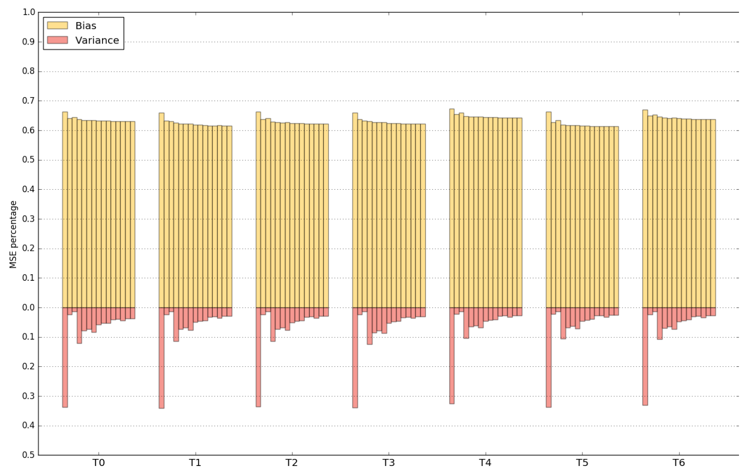


Fig. 7. Bias-variance decomposition of MSE for every target of the *Vegetation Condition* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

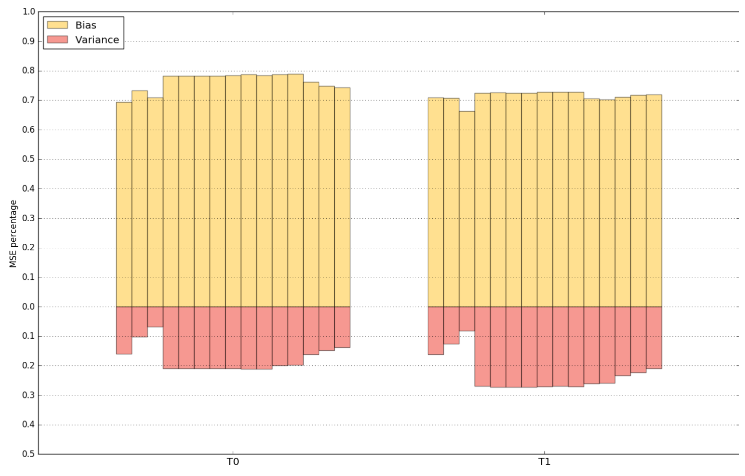


Fig. 8. Bias-variance decomposition of MSE for every target of the *Sigmea real* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

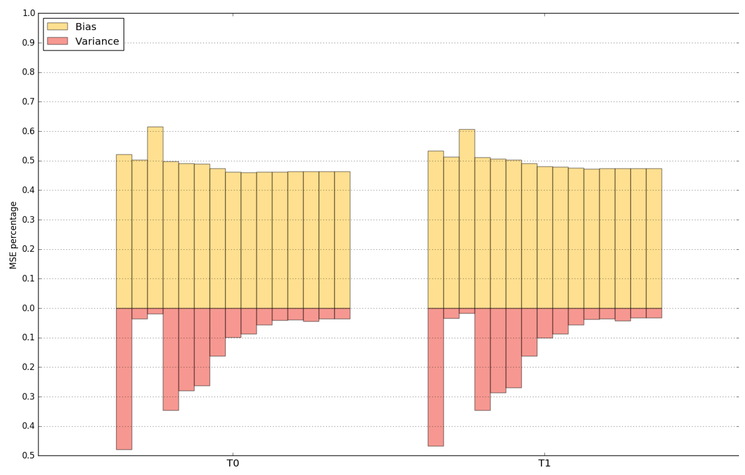


Fig. 9. Bias-variance decomposition of MSE for every target of the *Forestry-Slivnica-LandSat* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

The variance results for the *Forestry-Slivnica-LandSat* data set in Figure 9 are similar to the results for the *Vegetation condition* data set, with the notable difference that the bias of random forest of PCTs is substantially larger than the bias of the other algorithms. This occurred also in a few data sets which are shown in Appendix 5 and is not surprising, since individual trees in a random forest only use a fraction of all attributes to learn.

The results show that the difference in the predictive performance mainly arises from the difference in their variances. With a few exceptions, PCTs have the largest variance, followed by OPCTs with small parameter values. Increasing the parameter values greatly reduces the variance of OPCTs, while bagging and especially random forests have the smallest variances. The biases of different methods are generally very similar. Single PCTs and random forests have larger values on some data sets, while biases of OPCTs produced with large parameter values are occasionally smaller.

4.4. Interpretability of OPCTs

OPCTs, like option trees in general, offer a much higher degree of interpretability than ensemble methods. This is expressed through both the fact that the ”ensemble” of an option tree is represented in a compact form, i.e., a single tree, as well as the fact that many of the embedded trees overlap. Additionally, the regular PCT that would be learned from the same data is always present in the OPCT, as described in Section 2. A PCT and an OPCT learned on the *EDM* data set, and their relationship are illustrated in Figure 10.

Providing a domain expert with an option tree gives them a lot of choices with regards to the model. They can observe the selected options and attempt to determine which of the selected options were selected due to their actual importance as opposed to the sampling of the data set or other artifacts of the data. If they are able to discard all but one of the options in each of the option nodes, we can collapse the OPCT into a single tree, specifically a PCT, which corresponds not only to the data, but also to the knowledge of the domain expert. In terms of the predictive clustering framework, this means selecting only one of the overlapping hierarchical clusters when multiple clusters are presented. This approach also has the advantage that the domain expert need not be available for interaction when the model is learned, but can assess the OPCT and chose the preferred options later on.

This process can also be looked at through a different lens. As we have introduced option trees (in part) to address myopia, by considering more options and later deciding on only one of them in each option node, we are essentially ”looking ahead” of just the one split and utilizing, in the case of the domain expert, additional domain knowledge.

However, instead of using domain knowledge by proxy of interaction with a domain expert, we could also collapse the learned OPCT to a single PCT by using additional unseen data, i.e., by calculating an unbiased estimate of the predictive performance of the different options present in the OPCT. Since the collection and preparation of additional data examples could be expensive to the point of infeasibility, we could introduce a modified experimental setup. Part of the training data could be separated into a validation set which would not be used for the initial learning of the OPCT, but would be utilized to determine which of the selected options, and consequently embedded trees, has the best predictive performance. We would then collapse the OPCT into a PCT according to this validation set, after which we would test the obtained PCT on the test set. In this scenario, we could not only study the effect of myopia by comparing the collapsed OPCT to a PCT learned on the entire training data set, but also observe the effect of averaging multiple predictions on the predictive performance by comparing the collapsed PCT and the original (pseudo-ensemble) OPCT.

5. Conclusions

In this work, we propose an algorithm for learning option predictive clustering trees (OPCTs) for the task of multi-target regression (MTR). In contrast to standard regression, where the output is a single scalar value, in MTR the output is a data structure – a tuple/vector of numeric variables. We consider learning of a global model, that is a single model that predicts all target variables simultaneously.

More specifically, we propose OPCTs to address the myopia of the standard greedy PCT learning algorithm. OPCTs have the possibility to construct option nodes, i.e., nodes with a set of alternative sub-nodes, each containing a different split. These option nodes are constructed in the cases when the heuristic scores of the candidate splits are close to each other. Furthermore, OPCTs, and option trees in general, can be regarded as a condensed representation of an ensemble of trees.

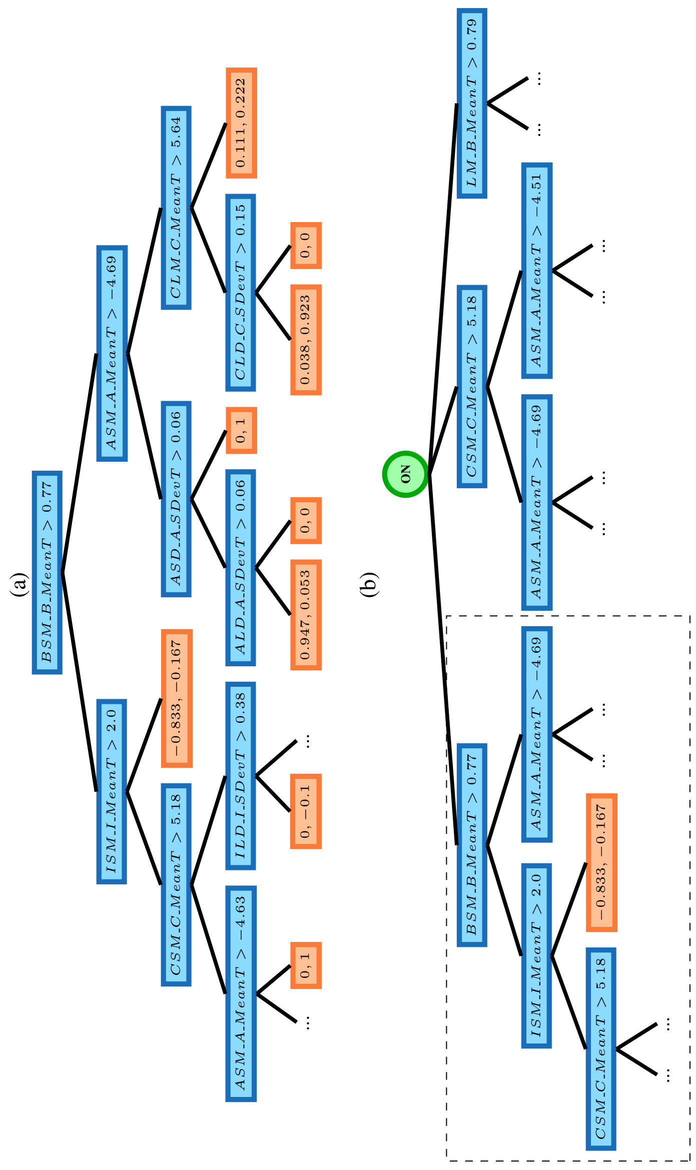


Fig. 10. A regular PCT (a) and an OPCT (b) learned on the EDM data set. The left child of a split node corresponds to the subtree where the test is satisfied. Note that the regular PCT is included in the OPCT as the subtree enclosed in the dashed rectangle.

The proposed method was experimentally evaluated on 11 benchmark MTR data sets. We selected two parameter configurations, OPCTe1d1 ($\varepsilon = 1, d = 1$), the largest OPCT which offers the best predictive performance, and OPCTe0.2d1 ($\varepsilon = 0.2, d = 1$), which provides a balance between predictive performance and efficiency. We compared this pair of OPCTs to regular PCTs and two ensemble learning methods – bagging and random forests of PCTs. The evaluation revealed that both OPCTe1d1 and OPCTe0.2d1 yield statistically significantly better predictive performance than single PCT. Next, the predictive performance of OPCTe1d1 is not statistically significantly different than that of the other two ensemble methods, however, it is slower to train and produces more leaves than the other compared methods. While the predictive performance of OPCTe0.2d1 was not on par with ensemble methods, it was not statistically significantly less efficient than PCTs, while offering significantly better performance.

We also performed parameter sensitivity analysis. The results show that both parameters that control the number of option nodes need large values to achieve good predictive performance. However, limiting the number of option nodes can lead to a more favorable trade-off between performance and efficiency.

We performed bias-variance decomposition of the mean squared error to determine the source of errors of the different methods. The results show the bias of the compared methods exhibits very similar behavior. OPCTs with larger parameter values were occasionally better in this regard, while random forests of PCTs sometimes had larger bias than other methods. On the other hand, the variance component often differed greatly between the algorithms. It was almost always the largest for single PCTs and smallest for random forests of PCTs. Increasing the parameter values for OPCTs reduced the variance and in some cases OPCTe1d1 achieved variance similar to that of the ensemble methods.

Finally, through an example, we illustrated the interpretability of the constructed OPCTs: they offer a multifaceted view on the data at hand.

We plan to extend this work along several directions. We will evaluate the OPCTs in the single tree context, i.e., we will use the induction of OPCTs as a beam-search algorithm for tree induction. Next, we will evaluate the influence of the two parameters at a more fine grained resolution. Finally, we will extend the algorithm towards other output types, i.e., machine learning tasks, such as multi-label classification, hierarchical multi-label classification and time series prediction.

Acknowledgments. We acknowledge the financial support of the European Commission through the grants ICT-2013-612944 MAESTRA and ICT-2013-604102 HBP, as well as the support of the Slovenian Research Agency through a young researcher grant (AO, TSP), the program Knowledge Technologies (P2-0103) and the project J2-9230.

References

1. Appice, A., Džeroski, S.: Stepwise induction of multi-target model trees. In: Machine Learning: ECML 2007, LNCS, vol. 4701, pp. 502–509. Springer (2007)
2. Bakır, G.H., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., Vishwanathan, S.V.N.: Predicting structured data. Neural Inf. Processing, The MIT Press (2007)
3. Blockeel, H., Struyf, J.: Efficient algorithms for decision tree cross-validation. Journal of Machine Learning Research 3, 621–650 (2002)

4. Borchani, H., Varando, G., Bielza, C., Larrañaga, P.: A survey on multi-output regression. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 5(5), 216–233 (2015)
5. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
6. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
7. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman & Hall/CRC (1984)
8. Breskvar, M., Kocev, D., Džeroski, S.: Ensembles for multi-target regression with random output selections. Machine Learning 107(11), 1673–1709 (2018)
9. Buntine, W.: Learning classification trees. Statistics and Computing 2(2), 63–73 (1992)
10. Corizzo, R., Pio, G., Ceci, M., Malerba, D.: DENCAST: distributed density-based clustering for multi-target regression. Journal of Big Data 6(1), 43 (2019)
11. Demšar, D., Debeljak, M., Džeroski, S., Lavigne, C.: Modelling pollen dispersal of genetically modified oilseed rape within the field. In: The Annual Meeting of the Ecological Society of America. p. 152 (2005)
12. Demšar, D., Džeroski, S., Larsen, T., Struyf, J., Axelsen, J., Bruns-Pedersen, M., Krogh, P.H.: Using multi-objective classification to model communities of soil. Ecological Modelling 191(1), 131–143 (2006)
13. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30 (2006)
14. Džeroski, S., Demšar, D., Grbović, J.: Predicting chemical parameters of river water quality from bioindicator data. Applied Intelligence 13(1), 7–17 (2000)
15. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Computing 4(1), 1–58 (Jan 1992)
16. Gjorgjioski, V., Džeroski, S., White, M.: Clustering analysis of vegetation data. Tech. Rep. 10065, Jožef Stefan Institute (2008)
17. Ikonomovska, E., Gama, J., Zenko, B., Dzeroski, S.: Speeding-up hoeffding-based regression trees with options. In: Proceedings of the 28th International Conference on Machine Learning, ICML 2011. pp. 537–544 (2011)
18. Kampichler, C., Džeroski, S., Wieland, R.: Application of machine learning techniques to the analysis of soil ecological data bases: relationships between habitat features and Collembolan community characteristics. Soil Biology and Biochemistry 32(2), 197–209 (2000)
19. Karalič, A.: First order regression. Ph.D. thesis, Faculty of Computer Science, University of Ljubljana, Ljubljana, Slovenia (1995)
20. Kocev, D., Ceci, M.: Ensembles of extremely randomized trees for multi-target regression. In: Discovery Science: 18th International Conference (DS 2015). LNCS, vol. 9356, pp. 86–100 (2015)
21. Kocev, D., Džeroski, S., White, M., Newell, G., Griffioen, P.: Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition. Ecological Modelling 220(8), 1159–1168 (2009)
22. Kocev, D., Struyf, J., Džeroski, S.: Beam search induction and similarity constraints for predictive clustering trees. In: Proc. of the 5th Intl Workshop on Knowledge Discovery in Inductive Databases KDID - LNCS 4747. pp. 134–151 (2007)
23. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Ensembles of multi–objective decision trees. In: ECML '07: Proceedings of the 18th European Conference on Machine Learning – LNCS 4701. pp. 624–631. Springer (2007)
24. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Tree ensembles for predicting structured outputs. Pattern Recognition 46(3), 817–833 (2013)
25. Kohavi, R., Kunz, C.: Option decision trees with majority votes. In: Proceedings of the 14th International Conference on Machine Learning. pp. 161–169. ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997)

26. Osojnik, A., Džeroski, S., Kocev, D.: Option predictive clustering trees for multi-target regression. In: Discovery Science: 19th International Conference (DS 2016). LNCS, vol. 9956, pp. 118–133 (2016)

27. Spyromitros-Xioufis, E., Tsoumakas, G., Groves, W., Vlahavas, I.: Multi-target regression via input space expansion: Treating targets as inputs. Machine Learning 104(1), 55–98 (2016)

28. Stojanova, D.: Estimating forest properties from remotely sensed data by using machine learning. Master's thesis, Jožef Stefan IPS, Ljubljana, Slovenia (2009)

29. Stojanova, D., Panov, P., Gjorgjioski, V., Kobler, A., Džeroski, S.: Estimating vegetation height and canopy cover from remotely sensed data with machine learning. Ecological Informatics 5(4), 256–266 (2010)

30. Struyf, J., Džeroski, S.: Constraint based induction of multi-objective regression trees. In: Proc. of the 4th International Workshop on Knowledge Discovery in Inductive Databases KDID - LNCS 3933. pp. 222–233. Springer (2006)

31. Tsoumakas, G., Spyromitros-Xioufis, E., Vrekou, A., Vlahavas, I.: Multi-target regression via random linear target combinations. In: Machine Learning and Knowledge Discovery in Databases: ECML-PKDD 2014, LNCS 8726. pp. 225–240 (2014)

Appendix

Bias-Variance Graphs

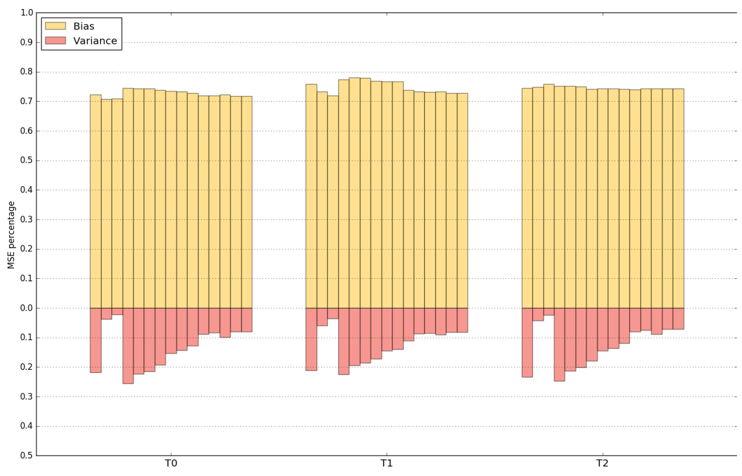


Fig. 11. Bias-variance decomposition of MSE for every target of the *Collembola* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

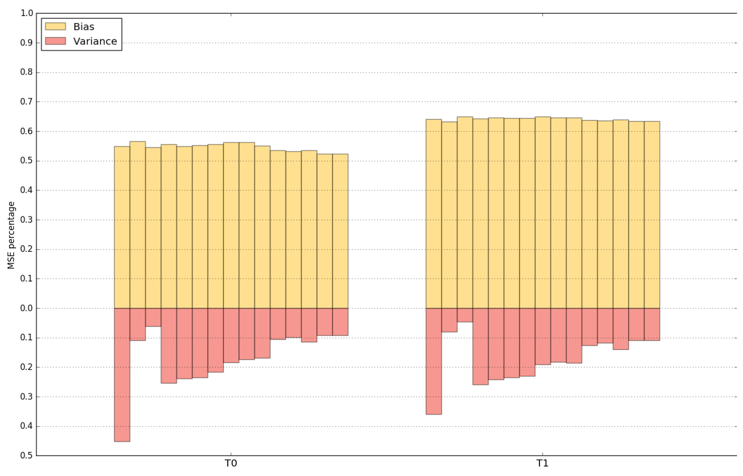


Fig. 12. Bias-variance decomposition of MSE for every target of the *EDM* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

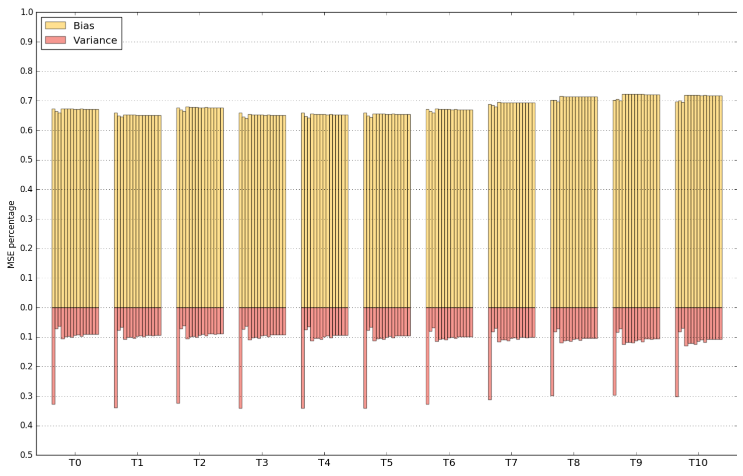


Fig. 13. Bias-variance decomposition of MSE for every target of the *Forestry-Kras* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

Additional tables

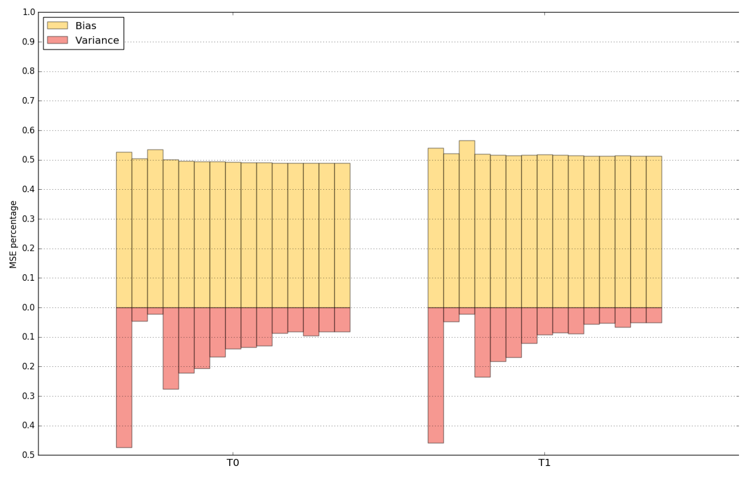


Fig. 14. Bias-variance decomposition of MSE for every target of the *Forestry-Slivnica-IRS* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

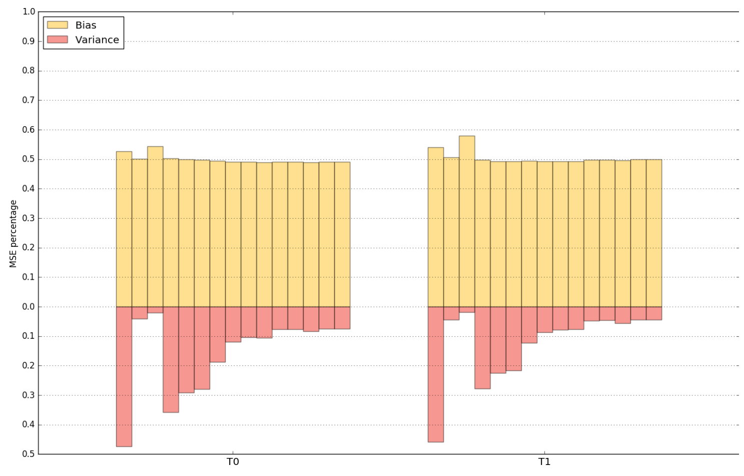


Fig. 15. Bias-variance decomposition of MSE for every target of the *Forestry-Slivnica-SPOT* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

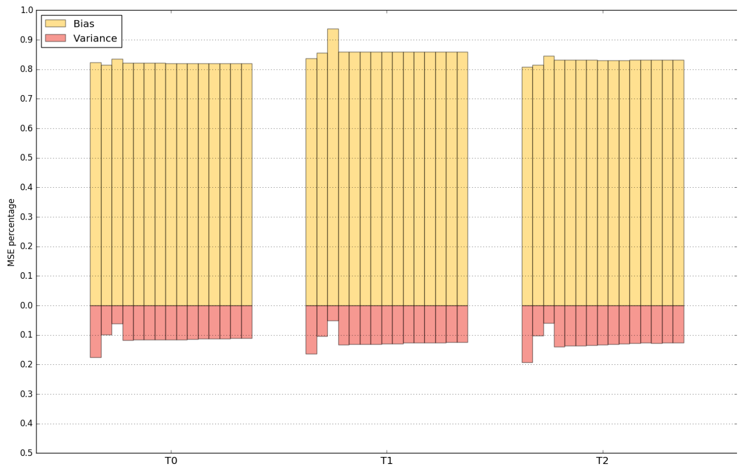


Fig. 16. Bias-variance decomposition of MSE for every target of the *Soil quality* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

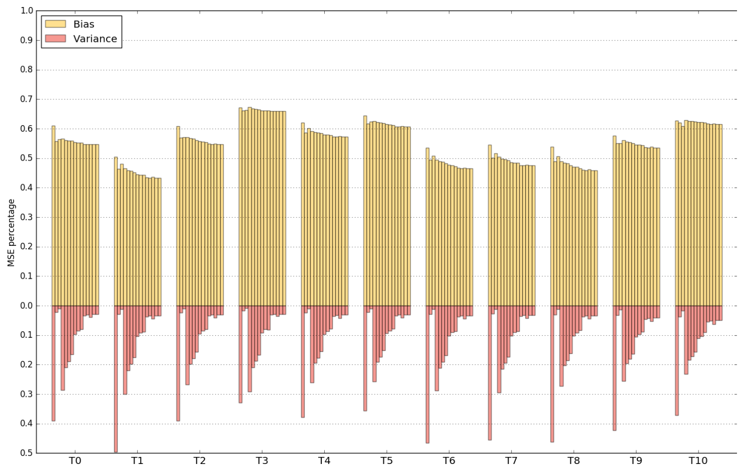


Fig. 17. Bias-variance decomposition of MSE for every target of the *Vegetation Clustering* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

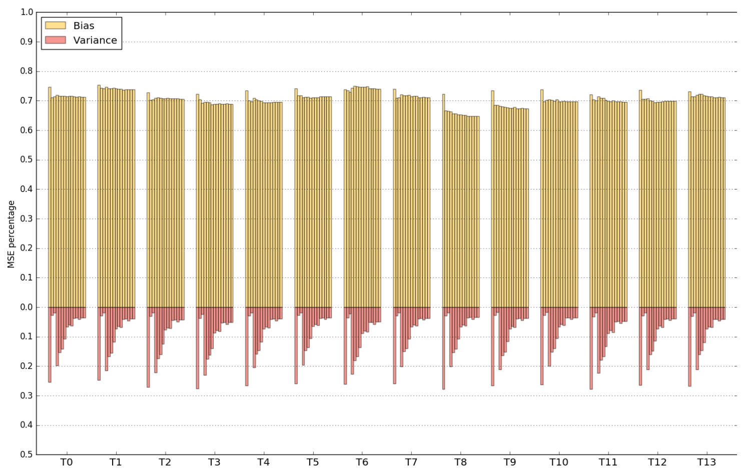


Fig. 18. Bias-variance decomposition of MSE for every target of the *Water quality* data set. Algorithms in order: PCT, bagging of PCTs, random Forest of PCTs and OPCTs with e0.1d0.5, e0.1d0.9, e0.1d1, e0.2d0.5, e0.2d0.9, e0.2d1, e0.5d0.5, e0.5d0.9, e0.5d1, e1d0.5, e1d0.9, e1d1.

	$\varepsilon = 0.1$			$\varepsilon = 0.2$			$\varepsilon = 0.5$			$\varepsilon = 1$		
	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$
collembolaV2	1689	4137	4137	4260	7513	7559	6691	14595	15836	12185	18151	18151
edm1	202	239	239	283	464	601	623	2683	3557	1882	4103	4103
Forestry_Kras	447646	830108	1002953	661259	1498692	1890340	1157139	2273266	2300007	2118682	2300007	2300007
Forestry_LIDAR_IRS	10804	18389	18921	28213	41939	45982	46523	125903	144777	91146	148377	148377
Forestry_LIDAR_Landsat	8372	31281	37406	33537	55194	70284	136907	244006	278533	200001	328002	332533
Forestry_LIDAR_Spot	4598	6932	6932	17374	58422	73037	71913	129843	143634	98279	148141	148339
sigmeareal	165	165	165	165	202	236	732	1164	1391	2484	21100	30855
soil_quality	3031	6385	8038	4617	9767	12184	23214	68372	79504	47981	104918	104918
VegetationCondition	81368	165625	192474	136516	294664	348911	368187	663801	714601	565032	761452	761452
vegetationV2	82363	86915	132580	137050	315931	344118	435360	1247710	1464212	1042378	1488376	1488376
water-quality	4009	7836	8736	6513	12895	16901	13614	51671	55516	37612	57792	57792

Table 2. Number of leaves in OPCTs with different parameter values.

	$\varepsilon = 0.1$			$\varepsilon = 0.2$			$\varepsilon = 0.5$			$\varepsilon = 1$		
	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$
collembolaV2	9	20	20	34	39	39	39	71	76	64	98	98
edm1	10	13	13	14	26	30	36	91	105	77	111	111
Forestry_Kras	28	62	80	44	95	109	87	111	111	111	111	111
Forestry_LIDAR_IRS	11	20	21	28	53	57	54	110	111	104	111	111
Forestry_LIDAR_Landsat	7	11	12	12	21	25	60	97	103	87	111	111
Forestry_LIDAR_Spot	5	5	5	12	52	67	60	107	111	96	111	111
sigmeareal	1	1	1	1	3	4	12	19	24	24	96	110
soil_quality	5	11	12	7	18	22	47	87	89	87	111	111
VegetationCondition	14	31	33	28	49	60	71	104	108	96	111	111
vegetationV2	10	12	15	18	33	35	51	95	108	94	111	111
water-quality	9	24	26	15	36	42	41	109	109	93	111	111

Table 3. Number of option nodes in OPCTs with different parameter values.

	$\varepsilon = 0.1$			$\varepsilon = 0.2$			$\varepsilon = 0.5$			$\varepsilon = 1$		
	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$	$d = 0.5$	$d = 0.9$	$d = 1$
collembolaV2	0.68	1.17	1.16	1.12	1.71	1.75	1.56	3.07	3.23	8.39	11.83	6.23
edm1	0.22	0.23	0.22	0.23	0.23	0.23	0.24	0.34	0.41	0.3	0.44	0.42
Forestry_Kras	449.71	781.2	942.74	653.2	1380.88	1723.76	1065.94	2083.34	2117.13	1920.28	2069.53	2110.88
Forestry_LIDAR_IRS	1.13	1.89	1.83	2.64	3.7	3.82	7.89	10.02	12.8	7.29	11.58	11.39
Forestry_LIDAR_Landsat	6.04	13.55	15.25	14.15	22.11	31.03	51.93	92.44	104.78	74.78	121.95	124.78
Forestry_LIDAR_Spot	0.9	1.19	1.18	2.59	11.17	10.07	11.84	18.36	19.03	13.03	18.37	18.75
sigmeareal	0.58	0.52	0.49	0.44	0.35	0.61	7.1	5.99	6.09	6.05	5.19	1.12
soil_quality	1.0	1.5	1.68	1.25	2.0	12.55	12.02	19.25	19.35	10.47	21.25	20.01
VegetationCondition	20.15	40.44	41.29	30.65	64.23	75.09	80.19	144.58	154.24	121.42	174.21	166.67
vegetationV2	24.45	24.55	37.32	39.35	84.73	96.15	116.34	324.78	381.27	273.54	373.55	375.06
water-quality	0.57	0.74	0.76	0.65	0.98	1.22	1.02	5.83	5.73	2.31	11.96	6.07

Table 4. Learning times for OPCTs with different parameter values.

Chapter 5

Option Predictive Clustering Trees for Multi-Label Classification

This chapter details our contributions in the context of option predictive clustering trees and their use for the multi-label classification task. In the introductory sections, we provided a high-level overview of OPCTs (Section 3.1) and described the MLC task (Section 2.1.2), where examples are labeled with sets of labels. Specifically, the contributions are the following:

1. Extension of PCTs with option nodes in the MLC context.
2. Experimental evaluation of the proposed method with a focus on investigating the trade-off between predictive performance and model sizes obtained with different parameters.
3. Evaluation of standard PCTs extracted from OPCTs by selecting specific options in option nodes.

The initial work was presented at the ETAI-2018 conference (Stepišnik Perdih et al., 2018). It included the proposal of OPCTs for MLC and a brief evaluation, which showed that OPCTs with many option nodes outperform standard PCTs and are on par with ensembles thereof also in the MLC setting. This work was later greatly extended and presented in the journal APH Journal of Applied Sciences (Stepišnik, Kocev, & Džeroski, 2020). We include the resulting paper in this chapter. It includes a detailed experimental comparison of OPCTs with different numbers of option nodes and bagging ensembles of PCTs with different numbers of trees. We show that OPCTs exhibit a similar trade-off between predictive performance and model size as bagging ensembles. However, OPCTs have an advantage in interpretability over ensembles of similar size because they can be represented as a single tree.

Additionally, we investigated the extraction of a standard PCT from an OPCT by selecting specific options in the option nodes. We explored two strategies: one based on the training set performance and another one based on the performance on a separate validation set. A standard PCT is always one of the trees that can be extracted from an OPCT in this way because the best split (which the standard tree selects) is always one of the options in an option node. Our results show that the extracted trees can outperform standard PCTs, due to the larger space of trees that are considered this way. Interestingly, the extracted trees are often the same as PCTs built with the standard algorithm and may occasionally give a worse performance on the test set.

Furthermore, we investigated where in the tree are option nodes most useful. We found that when the entire OPCT is used, option nodes are more useful closer to the root node.

However, when extracting standard PCTs from an OPCT, it is better to spread the option nodes also to lower levels. A possible reason for this is that while splits higher in the tree are more impactful, the evaluation of tests lower in the trees is less reliable because fewer learning examples are present. Keeping multiple options available on such occasions seems beneficial when extracting the final tree. We also presented a use-case for labeling music emotions, where the extracted tree performed better than the standard PCT despite being much smaller.

From the hypotheses we defined in Section 1.2, this chapter addresses the following:

H1 Introducing few option nodes in a PCT improves its predictive performance compared to standard PCTs while retaining the interpretability potential.

H2 Introducing many option nodes in a PCT makes the option PCTs exhibit similar behavior to bagging ensembles of PCTs in terms of time complexity, size, and predictive performance.

For the H1 hypothesis, we point out that the strictest parameter configuration for OPCTs allowed very few options nodes, in some cases even none at all, i.e., a standard PCT was constructed. When the parameters were relaxed so that a few more option nodes were created, the performance immediately improved. This confirms the H1 hypothesis in the MLC context. Hypothesis H2 was directly confirmed in the MLC setting by the comparison to bagging ensembles described above.

The paper included in this Chapter is:

- Stepišnik, T., Osojnik, A., Džeroski, S., Kocev, D. (2020). Option Predictive Clustering Trees for Multi-label classification. *Acta Polytechnica Hungarica – Journal of Applied Sciences*, 17(10), 109–128.

The contributions of Tomaž Stepišnik to this paper are as follows. He designed and implemented the investigated methods. He participated in the design of the study, carried out the experiments, and analyzed the results. He drafted the paper and revised it following the feedback from the co-authors and the reviewers.

Option Predictive Clustering Trees for Multi-label Classification

Tomaž Stepišnik, Dragi Kocev, Sašo Džeroski

Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Jožef Stefan Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia

{tomaz.stepisnik, dragi.kocev, saso.dzeroski}@ijs.si

Abstract: In this work, we focus on the task of multi-label classification (MLC), where every example is associated with a set of labels. We present an algorithm for learning option predictive clustering trees (OPCTs) for MLC, based on the predictive clustering framework. The algorithm addresses the myopia of the standard tree induction algorithm by considering alternative splits in the internal nodes of the tree and introducing option nodes where appropriate. An option tree can be viewed as a compact representation of an ensemble, as well as, used as a pool of candidates from which a single tree can be extracted. This broadens the space of trees that is searched and reduces the myopia, compared to the standard tree induction. We evaluate the proposed OPCTs on 12 benchmark MLC datasets from different domains. Results show that OPCTs as ensembles can achieve performance similar to the bagging ensembles of PCTs, while the single trees extracted from OPCTs can outperform standard PCTs. We also perform parameter sensitivity analysis and provide avenues for future work.

Keywords: predictive clustering trees; option trees; multi-label classification; myopia

1 Introduction

The most widely studied machine learning task is binary classification where the goal is to predict whether an example belongs to a group/class or not. If the examples can belong to a single class from a given set of *m* classes ($m > 2$) the task is known as multi-class classification. In this work, we focus on the multi-label classification (MLC) task, where a single example can be assigned several labels (i.e., a subset of a given set of possible labels).

MLC is a well-established predictive modelling task. The methods addressing this task belong in two groups: problem transformation and algorithm adaptation methods [4]. The problem transformation methods transform the multi-label learning problem into one or more single-label classification problems, which a great number of machine learning algorithms are capable of solving.

The problem transformation methods can be further split into three categories: binary relevance, label power-set and pair-wise methods. Binary relevance methods use the one-against-all strategy to convert the problem into several binary classification problems. A closely related method is the classifier chain method and its ensemble extension [9]. Label power-set (LP) methods transform the problem into a single multi-class classification problem, where a separate class is created for every possible subset of original labels. In this way, LP based methods directly take into account the label correlations. Representative methods include HOMER [5] and RAkEL [3]. Pair-wise methods perform pair-wise or round robin classification, with binary classifiers using Q(Q - 1)/2 classifiers covering all pairs of labels [17]. To combine these classifiers, the pairwise classification method uses majority voting.

The algorithm adaptation methods customize existing machine learning algorithms for the task of MLC. There are extensions of the following machine learning algorithms: boosting, k-nearest neighbors, decision trees and neural networks. The extended methods are able to directly handle multi-label data. AdaBoost.MH and AdaBoost.MR [8] are two extensions of AdaBoost for multi-label data. While AdaBoost.MH is designed to minimize Hamming loss, AdaBoost.MR is designed to find a hypothesis which ranks the correct labels at the top. Several variants for multi-label learning (ML-kNN) of the popular *k*-Nearest Neighbors (kNN) lazy learning algorithm have been proposed [1]. Decision tree extension was proposed within the predictive clustering framework [22]. A single predictive clustering tree (PCT) is constructed by using a splitting criterion that considers all of the labels. The PCTs for MLC were also used in an ensemble setting [14]. Neural networks have been adapted for MLC by introducing a new error function that takes multiple labels into account [18].

An extensive experimental comparison [12] of 12 MLC methods on 11 datasets using 16 performance measures showed that ensembles of PCTs are a state-of-the-art method for the MLC task. However, as already pointed out, a common concern with decision tree based models is their myopia, resulting from the greedy induction algorithm used to learn them. For this reason, several alternatives were proposed, such as beam search induction [15] and option trees [19], aimed at reducing the myopia of the trees. Both approaches showed that they can improve the performance of standard decision trees, while [12] also showed that very large option trees achieve the performance of bagging ensembles of decision trees. Conversely, [10] argues that exhaustive searching of the model space often leads to an inferior generalization.

In this work, we extend predictive clustering trees (PCTs) for MLC with option nodes, thus forming option predictive clustering trees (OPCTs). An option tree can be seen as a condensed representation of an ensemble of trees which share a common substructure. For illustration, see Figures 2 and 6. We also examine different techniques for selecting the optimal embedded tree from the OPCT. In this way, we increase the space of trees searched by the algorithm. We evaluate both OPCTs as ensembles and the best embedded trees extracted from OPCTs on several datasets from different domains. The goal of this paper is to see whether OPCTs as ensembles can achieve the performance of bagging ensembles for the MLC task, and if selecting the best embedded tree can reduce the myopia of the standard decision trees.

The remainder of this paper is organized as follows. Section 2 describes the algorithm for learning OPCTs for MLC. Next, Section 3 outlines the design of the experimental evaluation. Section 4 continues with a discussion of the results. Finally, we conclude and provides possible directions for further work.

2 Option Predictive Clustering Trees

The predictive clustering trees framework views a decision tree as a hierarchy of clusters. The top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [23] available at *http://clus.sourceforge.net*.

OPCTs extend the PCT framework by introducing option nodes into the tree building procedure. Option decision trees were first introduced as classification trees by Buntine [19] and then analyzed in more detail by Kohavi and Kunz [13]. Ikonomovska et al. [16] analyzed regression option trees in the context of data streams. We also evaluated OPCTs for the multi-target regression task [11] and hierarchical multi-label classification task [7].

The motivation for the introduction of option trees is to address the myopia of the *top-down induction of decision trees* (TDIDT) algorithm [20]. From the perspective of the predictive clustering framework, a PCT is a non-overlapping hierarchical clustering of the whole input space. Each node (subtree) corresponds to a clustering of a subspace and prediction functions are placed in the leaves, i.e., the lowest clusters in the hierarchy. In contrast, an OPCT allows the construction of an overlapping hierarchical clustering. This means that at each node of the tree several alternative hierarchical clusterings of the subspace can appear instead of a single one. When using TDIDT to construct a predictive clustering tree, all possible splits are evaluated by using a heuristic, and the best split is selected. However, other splits may have very similar heuristic values and the difference between them could be a consequence of noise or sampling that generated the data. In this case, selecting a different split could be optimal. To address this concern, the use of option nodes was proposed [13].

Figure 1 presents the TDIDT algorithm modified for the induction of OPCTs. The function call *FindBestTests(E,O)* returns the *O* best tests according to the heuristic score in descending order (best first). Every test is represented as a triplet *(t,h,P)*, where t is the actual test function, h is its heuristic value and P is the set of partitions produced by the test.

```
Procedure OptionPCT
Input: A data set $E$, parameter $\varepsilon$, maximum number of options $O$, current tree level
  $l$, maximum level for option nodes $L$
Output: An option predictive clustering tree
  $candidates = \text{FindBestTests}(E, O)$
  if $|candidates| > 0$ then
      if $|candidates| = 1$ or $l > L$ then
          $(t^*, h^*, \mathscr{P}^*) = candidates[0]$
          for each $E_i \in \mathscr{P}^*$ do
              $tree_i = \text{OptionPCT}(E_i, \varepsilon, O, l+1, L)$
          return $\text{node}(t^*, \bigcup_i \{tree_i\})$
      else
          $(t_0^*, h_0^*, \mathscr{P}_0^*) = candidates[0]$
          $nodes = \{\}$
          for each $(t_i^*, h_i^*, \mathscr{P}_i^*) \in candidates$ do
              if $\frac{h_i^*}{h_0^*} \geq 1 - \varepsilon$ then
                  for each $E_j \in \mathscr{P}_i^*$ do
                      $tree_j = \text{OptionPCT}(E_j, \varepsilon, O, l+1, L)$
                  $nodes = nodes \cup \{\text{node}(t^*, \bigcup_j \{tree_j\})\}$
          if $|nodes| > 1$ then
              return $\text{option\_node}(nodes)$
          else
              return $nodes[0]$
  else
      return $\text{leaf}(\text{Prototype}(E))$
```

Figure 1

The top down induction algorithm for option PCTs

The main component of the algorithm is the heuristic score used to evaluate the splits. For the MLC task, sets of labels are presented as binary vectors, where every component denotes the presence (1) or absence (0) of one label. For every component the Gini index is calculated, and their average is the final heuristic score. The algorithm introduces an option node into the tree when the best splits have similar heuristic values. Instead of selecting only the best split, we select every split s that satisfies the condition:

$$\frac{Heur(s)}{Heur(s_{best})} \geq 1 - \varepsilon \tag{1}$$

where s_{best} is the best split and ε determines how similar the heuristics must be. E.g., when ε=0.1, we are selecting only splits whose heuristics are within 10% of the best split. Typical values of ε are in the [0, 1] interval. After we have determined the candidate splits, we introduce an option node whose children are split nodes containing the selected splits. When no suitable test is found, a leaf node is created that stores the prototype of examples sorted to that leaf. For the MLC task, the *i*-th component of the prototype is the average value of the *i*-th components of examples in that leaf.

Introducing an option node with a large number of options is not advised [13] as it can lead to the explosion of model sizes. Therefore, we limit the maximum number of options included in a single option node (parameter *O*). To further limit the exponential growth of the tree, we also set the maximum depth in the tree where an option node can be introduced (parameter *L*). This means that on levels deeper than *L*, the tree is built following the standard TDIDT algorithm. This follows our intuition that splits lower in the tree, where clusters are already more homogeneous, are less important than splits higher in the tree.

Once an OPCT is learned, we use it to make predictions. In a regular PCT an example is sorted into a leaf (reached according to the tests in the nodes of the tree) where the prototype stored in that leaf is predicted. Traversing an example through an OPCT is the same for split nodes and leaves. When we encounter an option node, however, we traverse the example down each of the options. This means that in an option node an example is sorted to multiple leaves, where multiple predictions are produced. To obtain a single prediction in an option node, we aggregate the obtained predictions. For the MLC task, the aggregation is simple component-wise averaging of the vectors, as it is done in PCT ensembles. The resulting vector gives us pseudo-probabilities of every label, to which a threshold can be applied to select the actual labels.

An option tree is usually observed as a single tree, however, it can also be interpreted as a compact representation of an ensemble. We can extract *embedded trees* out of an option tree by replacing every option node with one of its options (Figure 2). A given OPCT is also an extension of the PCT learned on the same data. By definition, whenever we introduce an option node, the best split is one of the options included. Consequently, the PCT is an embedded tree in the OPCT, resulting from replacing all option nodes with the best option.

In addition to using an option tree as an ensemble of embedded trees, we can also extract a single embedded PCT out of it. This increases the space of trees searched by the TDIDT algorithm and directly addresses its myopia. The simplest way to select a single embedded PCT is to use the error on the training dataset as the selection criterion. This is an attractive option since it requires very little extra work and no extra data, but can lead to overfitting.

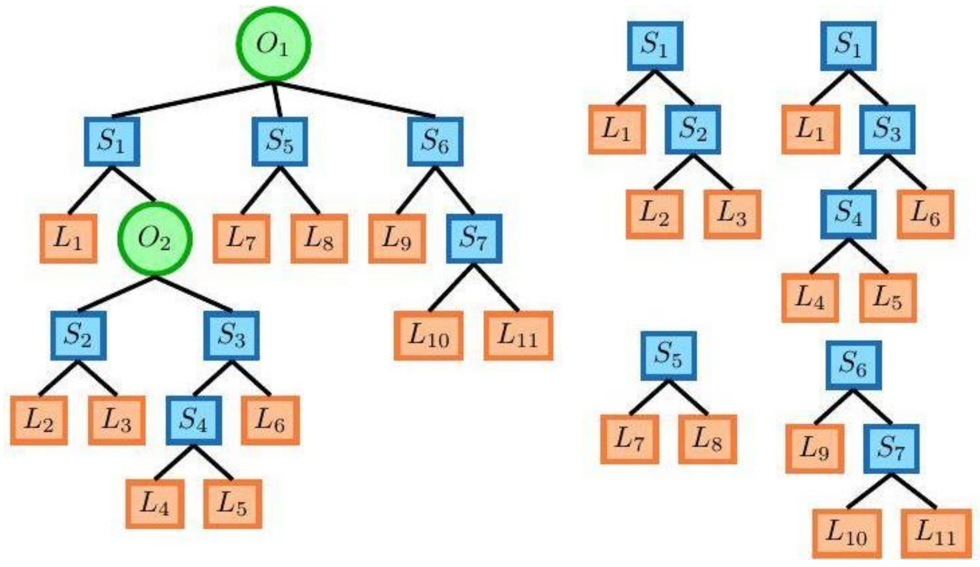


Figure 2
An option tree (left) and the ensemble of its embedded trees (right). O_i are option nodes, S_j split nodes and L_k leaf nodes.

Another option is to use a part of the training data as a separated validation set, which would not be used for the initial learning of the OPCT, but would be utilized to determine which of the embedded trees has the best predictive performance. We can also use expert knowledge to select the embedded tree. Providing a domain expert with an option tree gives them a lot of choices with regards to the model. This approach also has the advantage that the domain expert need not be available for interaction when the model is learned, but can assess the OPCT and chose the preferred options later on.

3 Experimental Design

The experimental evaluation was performed on 12 datasets from biology, text classification and multimedia domains. They are described in Table 1. All datasets except *CAL500* were retrieved pre-divided into training and testing sets and we used them in their original format to facilitate easier comparison of the results. The *CAL500* dataset was randomly split on training and test sets. All 12 datasets can be found at *http://mulan.sourceforge.net/datasets-mlc.html*.

There are many measures of performance used for the MLC task [12]. In our comparison we focus on Area Under the Average Precision-Recall Curve ($AU\overline{PRC}$) [2]. It combines the pseudo-probabilities of all the different labels as if they belonged to a single binary classification problem, and calculates the area under the precision-recall curve from them. A nice feature of $AU\overline{PRC}$ is that it is a threshold independent measure, so we do not have to worry about setting the threshold at which the labels are predicted in the leaves (as explained in Section 2). In the appendix we include the results for two additional performance measures (Ranking Loss and One Error [12]), from which similar conclusions can be made.

Table 1
Properties of the datasets used in the study: number of examples in the training/testing datasets (Ntr/Nte), number of descriptive attributes (discrete/continuous, *D*/*C*), the total number of labels (*Q*) and label cardinality (l_c)

	Ntr/Nte	*D*/*C*	*Q*	l_c
bibtex	4880/2515	1836/0	159	2.40
birds	322/323	2/258	19	1.01
CAL500	302/200	0/68	174	26.0
corel5k	4500/500	499/0	374	3.52
emotions	391/202	0/72	6	1.87
enron	1123/579	1001/0	53	3.38
flags	129/65	9/10	7	3.39
genbase	463/199	1185/0	27	1.25
medical	645/333	1449/0	45	1.25
scene	1211/1196	0/294	6	1.07
tmc2007-500	21519/7077	500/0	22	2.16
yeast	1500/917	0/103	14	4.24

We used the 12 benchmark datasets to evaluate OPCTs. Firstly, we wanted to see how the OPCTs as ensembles of embedded PCTs compare to the bagging ensembles of PCTs [21]. Specifically, we were interested in the trade-off between the performance and the size of the model. For the size of the model we looked at the total number of leaves in the tree(s). For the bagging ensembles of PCTs, the number of trees in the ensemble is the primary way to control the size of the model. We ran the experiments with 10, 25, 50, 100 and 125 PCTs in the ensemble.

For OPCTs, we can influence the number of option nodes introduced into the tree by using different parameter values. We considered different values for the ε parameter from the set {0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0}. When ε=1, option nodes with the best *O* splits are always introduced at the top *L* levels of the tree. We also tried different settings of the *O* and *L* parameters. Specifically, we used pairs (*O*, *L*) from the set {(10, 2), (5, 3), (3, 4)}. In addition to a better understanding of the performance vs. size trade-off, this also gives us insight into what is more valuable in OPCTs: more options at the top of the tree, or allowing option nodes lower in the tree. The pairs were selected in a way that at the maximum amount of option nodes introduced (e.g., at ε=1), the OPCTs would aggregate a similar amount of predictions compared to bagging ensembles of 100

PCTs. An important thing to note here is that different selections of parameters can still produce the same OPCT, if for a given dataset the same splits satisfy both criteria.

We also compared standard PCTs to the best embedded trees extracted from OPCTs (from here on referred to as BestEmbedded trees). We tried two approaches to extracting the BestEmbedded trees from OPCTs: selecting the embedded tree based on the training set performance, and based on the performance on a separate validation set not used to build the original OPCT. Both PCTs and BestEmbedded trees were post pruned using the training set in the first set of experiments, and using the validation set in the second set. This way, standard PCTs also benefited from the validation set, thus enabling a fair comparison. Note that BestEmbedded trees therefore use the validation set both for embedded tree selection and pruning. BestEmbedded trees were extracted from various OPCTs built with the same parameter settings described in the previous paragraph. This gave us insight into how different parameters influence the performance of the BestEmbedded trees. As a validation set, we randomly selected 20% of examples from the initial training set.

4 Results and Discussion

4.1 Comparison of OPCTs and Bagging Ensembles

Figure 3 shows the trade-off between the performance and size of the model for bagging ensembles of PCTs and OPCTs as ensembles. We can see that increasing the number of options improves the performance of OPCTs. In contrast to adding more trees to an ensemble, this is not guaranteed, since additional options include split nodes with lower heuristic scores. The performance improvement saturates at the largest OPCTs and often reaches the performance of bagging ensembles (on 7 datasets). All models seem to perform equally well on the *genbase* dataset, and much better than single PCTs (results on Figure 4).

Note that the comparison of the sizes of the trees is closely related to the comparison of running times. The most time consuming items when learning a tree are the evaluations of the candidate splits and then splitting the data. This is done in the same way in both PCTs (and ensembles thereof) and OPCTs, therefore, similar number of nodes in the trees indicates similar time needed for their induction.

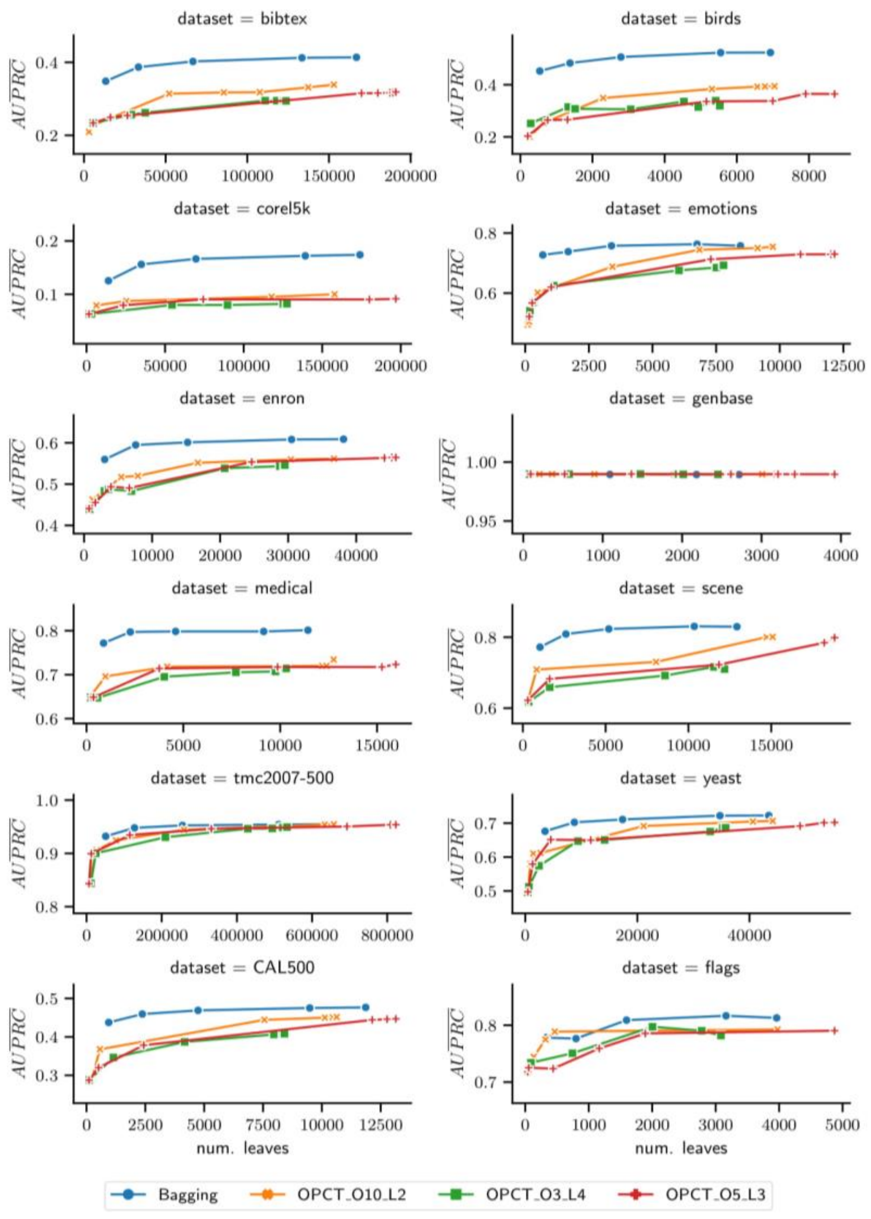


Figure 3
Trade-off between size and performance of bagging ensembles of PCTs and OPCTs. The size of ensembles is influenced by the number of trees, whereas the size of OPCTs is mainly influenced by the number of option nodes introduced into the tree. Note that every graph is on a different scale.

By comparing the results of OPCTs with different settings of parameters *O* and *L*, we can see that the pair (*O*, *L*) = (10, 2) offers the best performance, followed by (*O*, *L*) = (5, 3). This confirms our intuition that option nodes lower in the tree are less useful and do not offer enough improvement in performance to justify the increased model size. This is especially evident on the *bibtex*, *birds*, *emotions* and *scene* datasets.

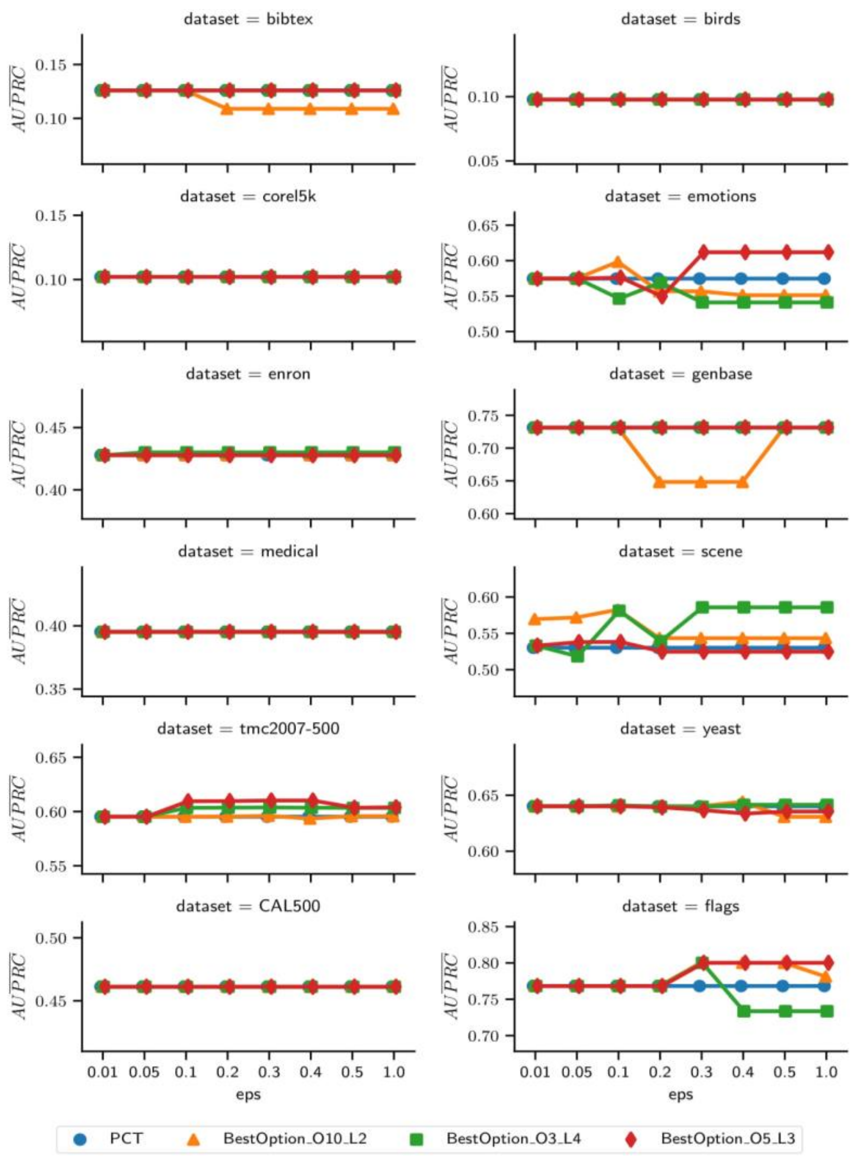


Figure 4

Performance of BestEmbedded trees produced by different parametrizations of OPCTs and selected based on training set performance, compared to the performance of a single PCT. Note that every graph is on a different scale.

4.2 Comparison of BestOptions and Standard PCTs

Figures 4 and 5 show the comparison of standard PCTs and the BestEmbedded trees extracted from OPCTs with different parameters. Figure 4 shows the results for BestEmbedded trees selected based on the training set performance, and Figure 5 based on the validation set performance. First thing we can notice is that the BestEmbedded trees and PCTs often have very similar or identical performance. There are usually no significant differences in size as well. In fact, often the BestEmbedded tree is the same as the standard PCT, especially when it is selected based on the training set. This is not surprising, since standard PCTs choose splits that perform best on the training set at each step. We can also see that having a separate validation set for pruning can greatly improve the performance of both standard PCTs and BestEmbedded trees (the *bibtex*, *genbase*, *medical* and *tmc2007* datasets).

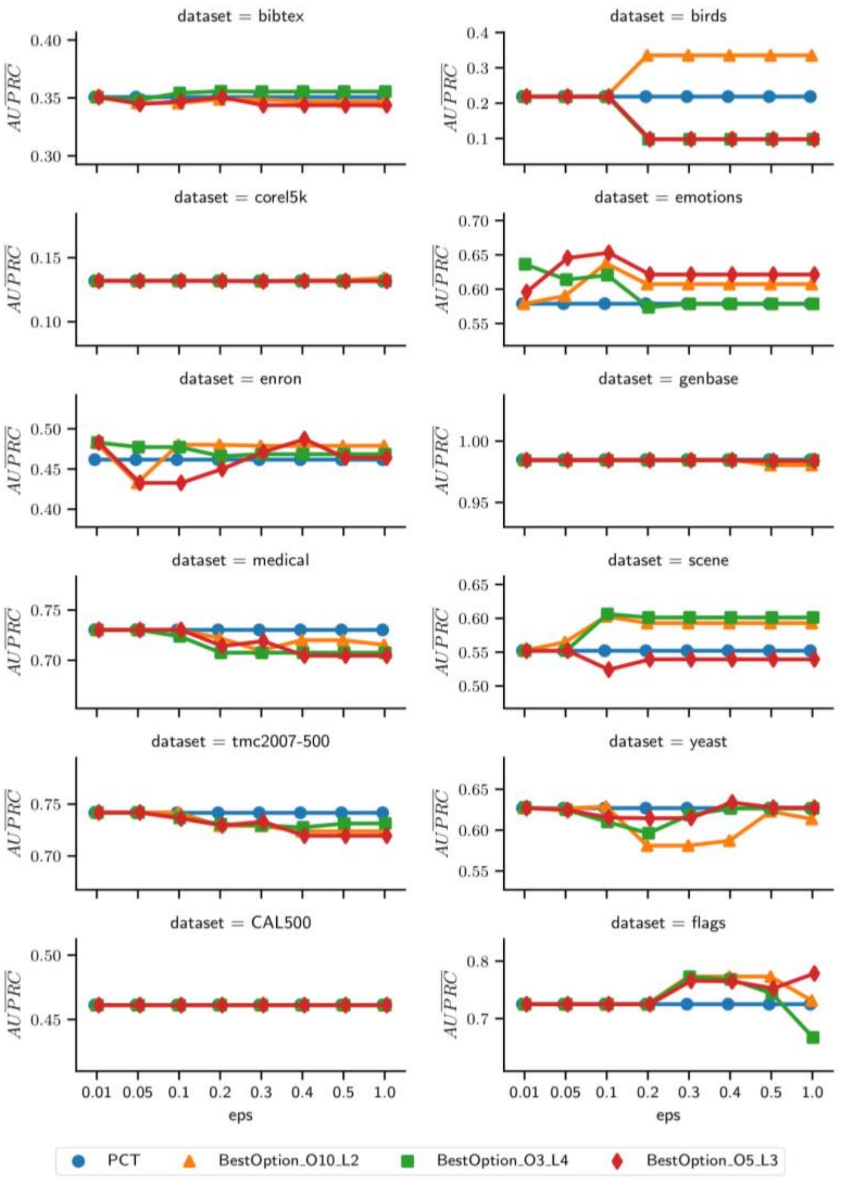


Figure 5
Performance of BestEmbedded trees produced by different parametrizations of OPCTs and selected based on validation set performance, compared to the performance of a single PCT. Note that every graph is on a different scale.

Another thing that the results show is that for extracting BestOptions, option nodes deeper in the tree are still helpful. While the parameter values $(O, L) = (10, 2)$ always provided the best results for OPCTs as ensembles, even parameter values $(O, L) = (3, 4)$ often lead to the best selection of BestEmbedded trees.

With larger OPCTs a larger space of trees is searched when selecting BestEmbedded trees, and that is where there are more differences compared to standard PCTs. Even when selected based on the training set performance, BestEmbedded trees often perform better than standard PCTs (on the *scene*, *tmc2007* and mostly *flags* datasets). But it also frequently happens that the BestEmbedded tree generalizes worse than the standard PCT (the *bibtex*, *emotions*, *genbase* and *flags* datasets).

Using a validation set to select the BestEmbedded tree mostly helps in selecting a tree that generalizes better, when compared to the train set selection. This can be nicely seen on the *bibtex*, *emotions*, *enron*, *scene* and *flags* datasets. However, in some cases it leads to a worse selection relatively to the standard PCT, as seen on the *medical*, *tmc2007* and *yeast* datasets. The cause for this may be oversearching, as described in [10]. By searching the space of trees more exhaustively, we have a higher chance of discovering "fluke" theories that fit the data well, but generalize poorly. As opposed to what is commonly understood as overfitting, these "fluke" theories are not overly complex. For example, when selecting the BestEmbedded tree, the standard PCT is always one of the options. It often happens that the tree with the best performance on the validation set is smaller (represents a simpler theory) than the standard PCT, but it still has a poorer performance on the test set.

4.3 Use Case Demonstration

We demonstrate the usefulness of OPCTs on the *emotions* dataset [6]. The examples in the *emotions* dataset are 30 seconds long music samples extracted from 100 different songs from different genres. The goal is to label the music samples with the emotions present in the music. There are 6 possible labels corresponding to 6 emotional clusters. The music samples are described with 8 rhythmic features and 64 timbre features.

Figure 6 shows an example of an OPCT trained on the *emotions* dataset for illustrative purpose. It was pruned to reduce the tree size to a humanly manageable proportions and contains only one option node with three options.

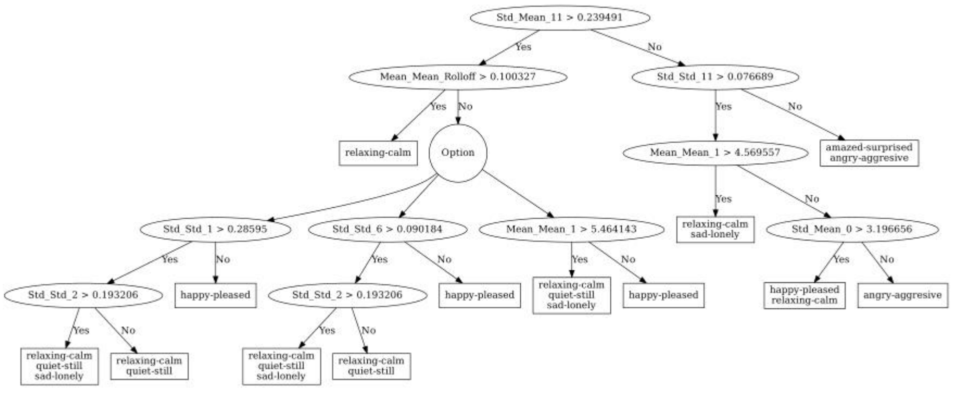


Figure 6

A heavily pruned OPCT constructed on the *emotions* dataset. Option nodes are represented as circles, standard split nodes have oval shape and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Figure 7 shows a PCT trained using the standard top down induction algorithm.

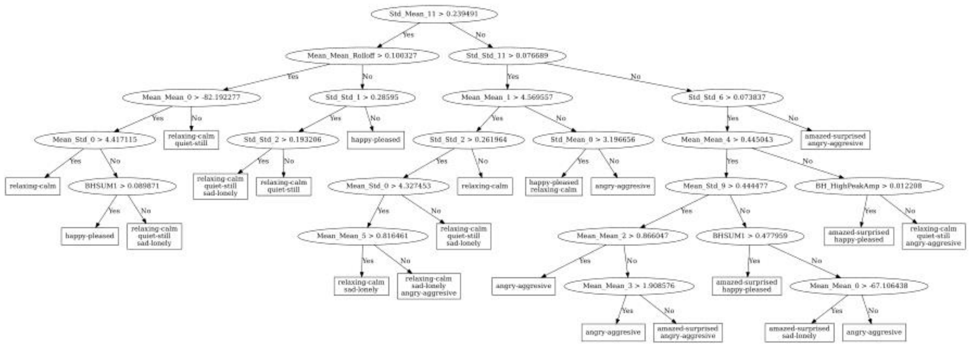


Figure 7

A standard PCT constructed on the *emotions* dataset. Split nodes have oval shapes and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Figure 8 shows the BestEmbedded tree selected from OPCT_O5_L3 with ε=0.2. They were both pruned using the validation set. The standard PCT has 22 leaf nodes and it achieved 0.57 $AU\overline{PRC}$, while the BestEmbedded tree is much smaller with only 9 leaves and has better predictive performance with 0.62 $AU\overline{PRC}$. We can see that they differ already at the root node.

This demonstrates the ability of OPCTs to help us find better performing trees, which can even be smaller than standard trees. Ideally, the BestEmbedded trees can be selected by experts, looking at the OPCT and selecting the options that are consistent with their knowledge.

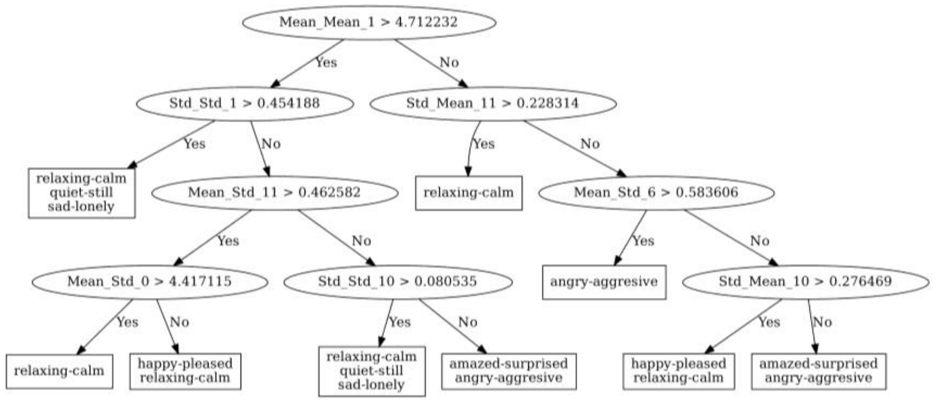


Figure 8
The BestEmbedded tree selected from OPCT_O5_L3 with ε=0.2 constructed on the *emotions* dataset. Standard split nodes have oval shapes and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Conclusions

In this work, we present an algorithm for learning option predictive clustering trees for the task of multi-label classification, where every example is associated with a set of labels. OPCTs are global models that predict the entire label sets simultaneously and belong in the algorithm adaptation group of methods. The main purpose of the proposed algorithm is to reduce the myopia of the standard greedy algorithm for learning PCTs. During tree construction, if multiple tests have heuristic scores similar to that of the best test, an option node is created that contains a set of alternative sub-nodes, called options, i.e., a set of the best performing splits. When predicting the labels of an example, each option produces a prediction, and predictions from different options are then aggregated in the option node. This leads us to consider option trees as condensed representations of an ensemble of trees.

In addition to the pseudo-ensemble aspect of OPCTs, we can also look at them as a set of possible PCTs (embedded trees), and select one of them as the final model. This broadens the search over the space of possible trees and decreases the myopia of the algorithm. There are multiple ways of selecting the best embedded trees. In our experiments, we used the performance on the training set, and the performance on a validation set. Another option is to use expert knowledge to select the best options in the option tree.

We performed an experimental evaluation of the OPCT method and the BestEmbedded trees, and compared them to standard PCTs and bagging ensembles of PCTs. The evaluation was performed on 12 datasets appropriate for the MLC task, and we measured model performance with $AU\overline{PRC}$. The results show that large OPCTs can often reach the performance of tree ensembles, and that BestEmbedded trees can outperform standard trees. Unfortunately, we can also fall in the trap of oversearching and find BestEmbedded trees with worse predictive performance compared to standard PCTs. We also looked into how different parameters affect the performance of OPCTs and BestEmbedded trees. We found that allowing more option nodes higher in the tree, while limiting them on the lower levels (parameter values (O, L) = (10, 2)) works best for OPCTs as ensembles. For BestEmbedded trees, option nodes lower in the tree are still useful and can help find better trees. We demonstrated the potential of OPCTs on a music emotion labelling dataset, where the BestEmbedded tree was much smaller than the standard PCT, yet had a better predictive performance.

There are several paths for further work. We would like to evaluate OPCTs and BestEmbedded trees in a domain where an expert would help select the best splits in the option nodes. Additionally, we plan to investigate the use of OPCTs for feature ranking and selection for MLC datasets.

Acknowledgement

We acknowledge the financial support of the Slovenian Research Agency via the grants P2-0103 and a young researcher grant to TS, as well as the European Commission, through the grants MAESTRA (Learning from Massive, Incompletely annotated, and Structured Data) and HBP (The Human Brain Project), SGA1 and SGA2. SD and DK also acknowledge support by Slovenian Research Agency (via grants J4-7362, L2-7509, N2-0056 and J2-9230), the European Commission (the LANDMARK project). The computational experiments presented here were executed on the computing infrastructure from the Slovenian Grid (SLING) initiative.

References

[1] M. L. Zhang and Z. H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognition,* Vol. 40, pp. 2038-2048, 2007

[2] C. Vens, J. Struyf, L. Schietgat, S. Džeroski and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Machine Learning,* Vol. 73, pp. 185-214, 2008

[3] G. Tsoumakas and I. Vlahavas, "Random k-Labelsets: An Ensemble Method for Multilabel Classification," in *Proc. of the 18th European conference on Machine Learning,* 2007

[4] G. Tsoumakas, I. Katakis and I. Vlahavas, "Mining Multi-label Data," in *Data Mining and Knowledge Discovery Handbook*, Springer Berlin / Heidelberg, 2010, pp. 667-685

[5] G. Tsoumakas, I. Katakis and I. Vlahavas, "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels," in *Proc. of the ECML/PKDD Workshop on Mining Multidimensional Data*, 2008

[6] K. Trochidis, G. Tsoumakas, G. Kalliris and I. Vlahavas, “Multi-label classification of music into emotions,” 2008

[7] T. Stepišnik Perdih, A. Osojnik, S. Džeroski and D. Kocev, “Option Predictive Clustering Trees for Hierarchical Multi-label Classification,” in *Discovery Science*, 2017

[8] R. E. Schapire and Y. Singer, “BoosTexter: A Boosting-based System for Text Categorization,” *Machine Learning,* Vol. 39, No. 2, pp. 135-168, 2000

[9] J. Read, B. Pfahringer, G. Holmes and E. Frank, “Classifier chains for multi-label classification,” *Machine Learning,* Vol. 85, pp. 333-359, 2011

[10] J. R. Quinlan and R. M. Cameron-Jones, “Oversearching and Layered Search in Empirical Learning,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2,* San Francisco, CA, USA, 1995

[11] Osojnik, S. Džeroski and D. Kocev, “Option predictive clustering trees for multi-target regression,” in *Discovery Science: 19th International Conference (DS 2016),* 2016

[12] G. Madjarov, D. Kocev, D. Gjorgjevikj and S. Džeroski, “An extensive experimental comparison of methods for multi-label learning,” *Pattern Recognition,* Vol. 45, pp. 3084-3104, 2012

[13] R. Kohavi and C. Kunz, “Option Decision Trees with Majority Votes,” in *Proceedings of the 14th International Conference on Machine Learning,* San Francisco, CA, USA, 1997

[14] D. Kocev, C. Vens, J. Struyf and S. Džeroski, “Tree ensembles for predicting structured outputs,” *Pattern Recognition,* Vol. 46, pp. 817-833, 2013

[15] D. Kocev, J. Struyf and S. Džeroski, “Beam search induction and similarity constraints for predictive clustering trees,” in *Proc. of the 5th Intl Workshop on Know. Disc. in Inductive Databases KDID - LNCS 4747*, 2007

[16] E. Ikonomovska, J. Gama, B. Zenko and S. Dzeroski, “Speeding-Up Hoeffding-Based Regression Trees With Options,” in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011

[17] J. Fürnkranz, “Round robin classification,” *Journal of Machine Learning Research,* Vol. 2, pp. 721-747, 2002

[18] K. Crammer and Y. Singer, “A family of additive online algorithms for category ranking,” *Journal of Machine Learning Research,* Vol. 3, pp. 1025-1058, 2003

[19] W. Buntine, "Learning classification trees," *Statistics and Computing*, Vol. 2, pp. 63-73, 1992

[20] L. Breiman, J. Friedman, R. A. Olshen and C. J. Stone, Classification and Regression Trees, Chapman & Hall/CRC, 1984

[21] L. Breiman, "Bagging Predictors," *Machine Learning*, Vol. 24, pp. 123-140, 1996

[22] H. Blockeel, L. D. Raedt and J. Ramon, "Top-down induction of clustering trees," in *Proceedings of the 15th International Conference on Machine Learning*, 1998

[23] H. Blockeel and J. Struyf, "Efficient algorithms for decision tree cross-validation," *Journal of Machine Learning Research*, Vol. 3, pp. 621-650, 2002

Appendix – additional results

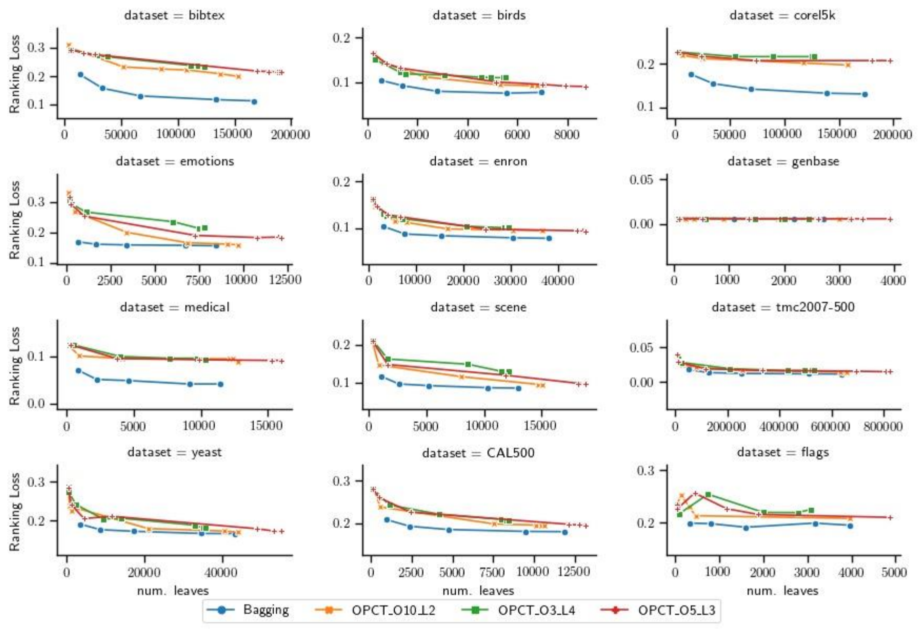


Figure 9

Comparison of bagging and OPCTs using the ranking loss measure. For details see Figure 3.

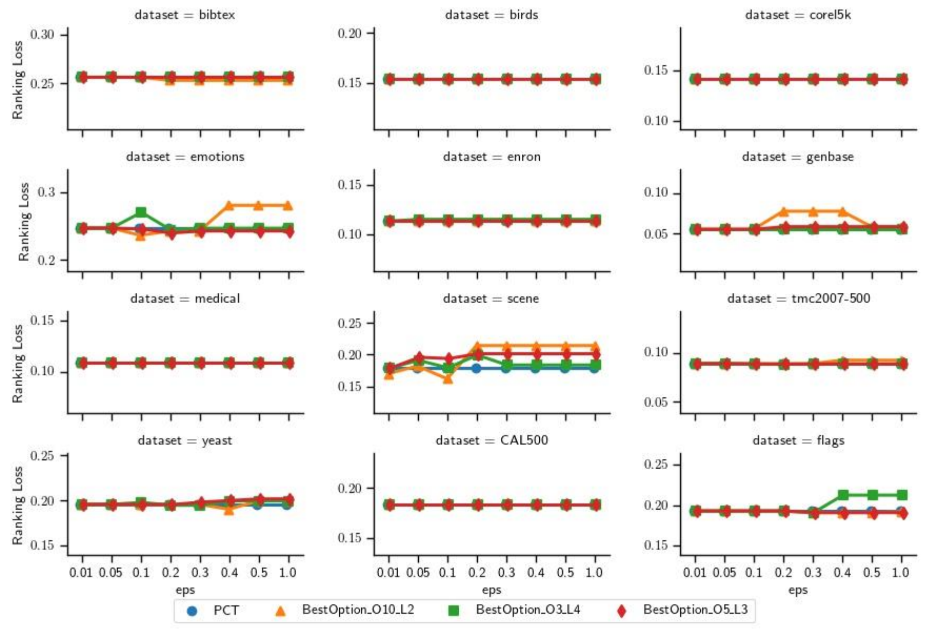


Figure 10

Comparison of PCTs and BestOption trees selected based on the training set performance using the ranking loss measure

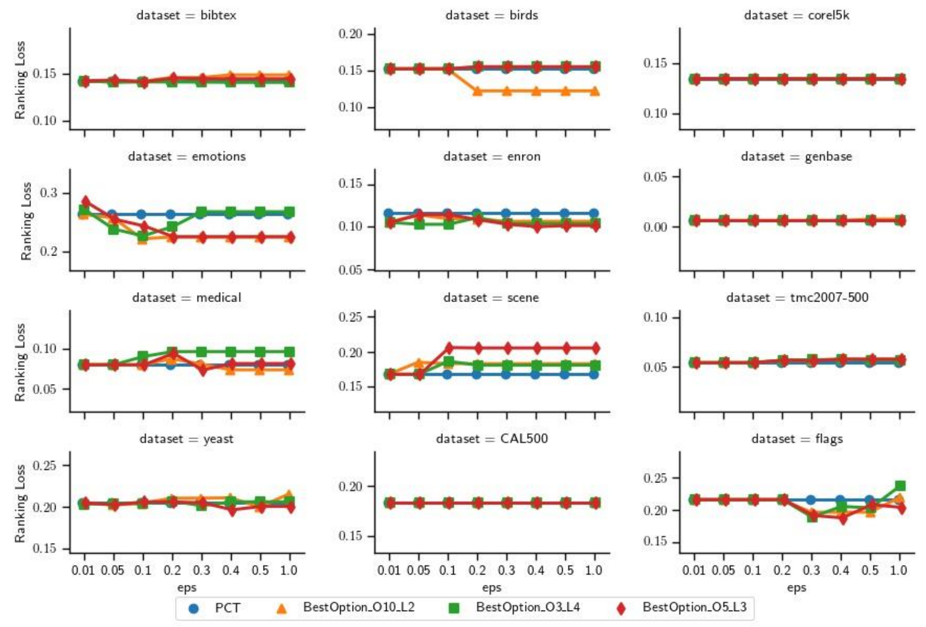


Figure 11

Comparison of PCTs and BestOption trees selected based on the validation set performance using the ranking loss measure

Chapter 6

Option Predictive Clustering Trees for Hierarchical Multi-Label Classification

In this chapter, we detail the contributions related to option predictive clustering trees and their use for the hierarchical multi-label classification task. In the introductory sections, we provided a high-level overview of OPCTs (Section 3.1) and described the HMLC task (Section 2.1.2), where examples are again labeled with sets of labels. Additionally, we have a partial ordering defined among the labels. Our contributions in this context are:

1. Extension of PCTs with option nodes in the HMLC context.
2. Experimental evaluation of the proposed method, focusing on the influence of different parameter settings on the predictive performance, time complexity, and interpretability.

The listed contributions were presented with a paper at the DS-2017 conference (Stepišnik Perdih et al., 2017), which we include in this chapter. Chronologically, we performed this research between the MTR and MLC papers presented in the previous two chapters. We include it here after our work on MLC because HMLC generalizes MLC. We proposed OPCTs for HMLC and experimentally evaluated them with different parameter settings controlling the number of option nodes introduced to the trees. We identified three interesting settings. The least strict setting again resulted in OPCTs that were similar to bagging ensembles in terms of predictive performances and model sizes. The middle setting traded a small amount of predictive performance for much smaller models. In the setting that allowed the fewest option nodes, OPCTs came closer to PCTs in terms of size while maintaining a noticeable advantage in predictive performance. Therefore, OPCTs solidified their place as a bridge between single trees and ensembles thereof.

From the hypotheses we defined in Section 1.2, this chapter addresses the following:

H1 Introducing few option nodes in a PCT improves its predictive performance compared to standard PCTs while retaining the interpretability potential.

H2 Introducing many option nodes in a PCT makes the option PCTs exhibit similar behavior to bagging ensembles of PCTs in terms of time complexity, size, and predictive performance.

Our results confirmed hypotheses H1 and H2 also for the HMLC task: different parameter settings result in OPCTs with various amounts of option nodes, providing different trade-offs between model size/interpretability and predictive performance.

The paper included in this Chapter is:

- Stepišnik Perdih, T., Osojnik, A., Džeroski, S., Kocev, D. (2017). Option Predictive Clustering Trees for Hierarchical Multi-label Classification. In A. Yamamoto, T. Kida, & T. Kuboyama (Eds.), *Discovery Science* (pp. 116–123). Springer International Publishing.

The contributions of Tomaž Stepišnik to this paper are as follows. He contributed to the design and implementation of OPCTs for HMLC. He participated in the design of the study, carried out the experiments, and analyzed the results. He drafted the paper and revised it following the feedback from the co-authors and the reviewers.

Option Predictive Clustering Trees for Hierarchical Multi-label Classification

Tomaž Stepišnik Perdih[1,2]( ), Aljaž Osojnik[1,2], Sašo Džeroski[1,2], and Dragi Kocev[1,2]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
{tomaz.stepisnik,aljaz.osojnik,saso.dzeroski,dragi.kocev}@ijs.si
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

Abstract. In this work, we address the task of hierarchical multi-label classification (HMLC). HMLC is a variant of classification, where a single example may belong to multiple classes at the same time and the classes are organized in the form of a hierarchy. Many practically relevant problems can be presented as a HMLC task, such as predicting gene function, habitat modelling, annotation of images and videos, etc. We propose to extend the predictive clustering trees for HMLC – a generalization of decision trees for HMLC – toward learning option predictive clustering trees (OPCTs) for HMLC. OPCTs address the myopia of the standard tree induction by considering alternative splits in the internal nodes of the tree. An option tree can also be regarded as a condensed representation of an ensemble. We evaluate OPCTs on 12 benchmark HMLC datasets from various domains. With the least restrictive parameter values, OPCTs are comparable to the state-of-the-art ensemble methods of bagging and random forest of PCTs. Moreover, OPCTs statistically significantly outperform PCTs.

1 Introduction

Supervised learning is one of the most widely researched areas of machine learning, where the goal is to learn, from a set of examples with known class, a function that outputs a prediction for the class of a previously unseen example. The most widely studied machine learning task is binary classification where the goal is to classify the examples into two groups. The task where the examples can belong to a single class from a given set of m classes ($m \geq 3$) is known as multi-class classification. The case where the output is a real value is called regression.

In many real life problems of predictive modelling the target is structured (e.g., the target is a vector of values with dependencies between them, or a time series). In this work, we focus on the task of hierarchical multi-label classification (HMLC). HMLC is a variant of classification, where a single example may belong to multiple classes at the same time and the classes are organized in the form of a hierarchy. An example that belongs to some class c automatically belongs to all super-classes of c: This is called the hierarchical constraint. Problems of this kind can be found in many domains including text classification, functional genomics,

A. Yamamoto et al. (Eds.): DS 2017, LNAI 10558, pp. 116–123, 2017.
DOI: 10.1007/978-3-319-67786-6_9

and object/scene classification. Silla and Freitas [19] give a detailed overview of the possible application areas and the different approaches to HMLC.

Decision tree based methods take a very notable place among approaches to HMLC. When used as base predictive models in an ensemble, they can yield a state-of-the-art performance [13,18]. A prominent global tree method for HMLC is a predictive clustering tree (PCT) for HMLC [20]. PCTs for HMLC inherit the properties of decision trees: they are interpretable models, but learning them is greedy. The performance of the trees is significantly improved when they are used in an ensemble setting [13]. However, the greediness of the tree construction process can lead to learning sub-optimal models. One way to alleviate this is to use a beam-search algorithm for tree induction [12], while another approach is to introduce option splits in the nodes [5,14].

In this work, we propose to extend predictive clustering trees (PCTs) for HMLC towards option trees, hence we propose to learn option predictive clustering trees (OPCTs). An option tree can be seen as a condensed representation of an ensemble of trees which share a common substructure. More specifically, the heuristic function for split selection can return multiple values that are close to each other within a predefined range. These splits are then used to construct an option node. For illustration, see Fig. 1.

The remainder of this paper is organized as follows. Section 2 proposes the algorithm for learning option PCTs for HMLC. Next, Sect. 3 outlines the design of the experimental evaluation. Section 4 continues with a discussion of the results. Finally, Sect. 5 concludes and provides directions for further work.

2 Option Predictive Clustering Trees

The predictive clustering trees framework views a decision tree as a hierarchy of clusters. The top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [1], which is available at http://clus.sourceforge.net.

Option predictive clustering trees (OPCT) extend the usual PCT framework, by introducing option nodes into the tree building procedure. Option decision trees were first introduced as classification trees by Buntine [5] and then analyzed in more detail by Kohavi and Kunz [14]. Ikonomovska et al. [10] analyzed regression option trees in the context of data streams. We also evaluated OPCTs for the multi-target regression task [16].

The major motivation for the introduction of option trees is to address the myopia of the *top-down induction of decision trees* (TDIDT) algorithm [4]. Viewed through the lens of the predictive clustering framework, a PCT is a non-overlapping hierarchical clustering of the whole input space. Each node/subtree corresponds to a clustering of a subspace and prediction functions are placed in the leaves, i.e., lowest clusters in the hierarchy. An OPCT, however, allows the construction of an overlapping hierarchical clustering. This means that, at each node of the tree several alternative hierarchical clusterings of the subspace can appear instead of a single one. When using TDIDT to construct a predictive clustering tree, and in particular when partitioning the data, all possible splits are evaluated by using a heuristic and the best one is selected. However, other splits may have very similar heuristic values. The best partition could be obtained with another split as a consequence of noise or of the sampling that generated the data. In this case, selecting a different split could be optimal. To address this concern, the use of option nodes was proposed [14].

The procedure of PCT learning for the HMLC task is presented in [13]. We modify it by introducing an option node into the tree when the best splits have similar heuristic values. Instead of selecting only the best split, we select several of them. Specifically, we select splits s, that satisfy the condition:

$$\frac{\text{Heur}(s)}{\text{Heur}(s_{best})} \geq 1 - e \cdot d^l, \tag{1}$$

where s_{best} is the best split, e determines how similar the heuristics must be, $d \in [0, 1]$ is a decay factor and l is the depth of the node we are attempting to split. E.g., when $e = 0.1$, we are selecting only splits whose heuristics are within 10% of the best split at the top level. We define the depth of a node to be the number of its ancestor nodes, excluding option nodes, as they do not split the data. The use of a decay factor makes the selection criterion more stringent in the lower nodes of the tree, where the impact of the split selection is also lower. After we have determined the candidate splits, we introduce an option node whose children are split nodes obtained by using the selected splits.

Introducing an option node with a large number of options is not advised [14] as it can lead to the explosion of model sizes. Therefore, we limit the maximum number of options for a single option node to 5 and also prohibit the induction of option nodes on depth 3 and greater.

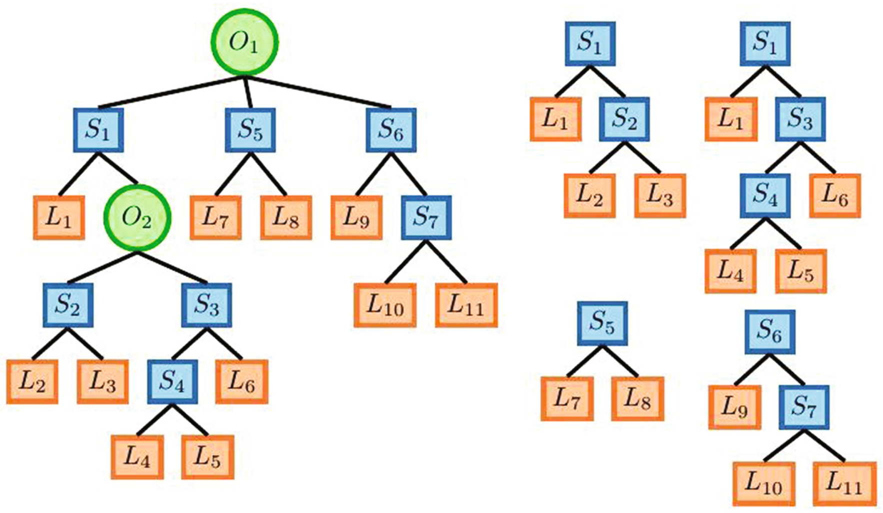


Fig. 1. An option tree (left) and the ensemble of its embedded trees (right). O_i are option nodes, S_j split nodes and L_k leaf nodes.

Once an OPCT is learned, we use it for prediction. In a regular PCT an example is sorted into a leaf (reached according to the tests in the nodes of the tree) where a prediction is made using a prototype function. Traversing an example through an OPCT is the same for split nodes and leaves. When we encounter an option node, however, we traverse the example down each of the options. This means that in an option node an example is sorted to multiple leaves, where multiple predictions are produced. To obtain a single prediction in an option node, we aggregate the obtained predictions.

An option tree is usually observed as a single tree, however, it can also be interpreted as a compact representation of an ensemble. We can extract *embedded trees* out of an option tree by replacing every option node with one of its options (Fig. 1). A given OPCT is also an extension of the PCT learned on the same data. By definition, whenever we introduce an option node, we include the best split. Consequently, the PCT is an embedded tree in the OPCT, resulting from replacing all option nodes with the best option.

3 Experimental Design

We evaluated the performance and efficiency of the proposed OPCT method with different parameter values and compared it to the standard PCTs and ensembles of PCTs. Evaluation was done on 12 datasets from biology, text classification and image annotation domains. They are described in Table 1. The datasets came pre-divided into training and testing sets and we used them in their original format, for easier comparison of the results.

OPCTs are evaluated for various values of parameters e and d. For e we consider values 0.1, 0.2, 0.5 and 1.0, while d takes values 0.5, 0.9 and 1.0. Notably, different selections of parameters can produce the same OPCT, if for a given dataset the same splits satisfy both criteria. Hereafter, the OPCT method with specific parameter values is denoted OPCT_eX_dY (e.g., for $e = 0.5, d = 0.9$, OPCT_e0.5_d0.9). The border case OPCT_e1_d1 always selects the 5 best options regardless of their heuristic score, making this setting similar to ensembles.

For PCTs and OPCTs we use the F-test as a pruning mechanism. Specifically, we check if a split results in a statistically significant improvement over the single node. If no split satisfies the F-test, the learning in the node stops. The significance level for the test was selected from the set of values $\{0.125, 0.1, 0.05, 0.01, 0.005, 0.001\}$ using internal 3-fold cross validation on the training set.

For ensembles, we considered bagging [2] and random forests [3]. For both methods we used 100 trees in the ensemble. Random forests algorithm also takes as input the size of the feature subset randomly selected at each node. For this we used the square root of the number of descriptive variables ($\lceil\sqrt{|D| + |C|}\rceil$).

Performance was measured using Area Under the Average Precision-Recall Curve ($AU\overline{PRC}$) [20]. For efficiency, we looked at the model size (number of leaves in a tree/ensemble). For statistical comparison of the methods we adopted the recommendations by Demšar [7]. Specifically, we used the Friedman test for statistical significance and Nemenyi post-hoc test to detect between which algorithms the significant differences occur. For both tests we selected confidence

Table 1. Descriptions of datasets used for the evaluation. The table shows the number of examples in the training and testing sets (N_{tr}/N_{te}), number of descriptive attributes (discrete/continuous, D/C), number of labels in the hierarchy ($|\mathcal{H}|$), maximal depth of the labels in the hierarchy ($\mathcal{H}_d$) and average number of labels per example ($\overline{\mathcal{L}}$).

	N_{tr}/N_{te}	$\|D\|/\|C\|$	$\|\mathcal{H}\|$	$\mathcal{H}_d$	$\overline{\mathcal{L}}$
Diatoms [9]	2065/1054	0/371	377	3.0	1.95
Enron [11]	988/660	0/1001	54	3.0	5.30
Expression–FunCat [6]	2494/1291	4/547	475	4.0	8.87
Exprindiv–FunCat [6]	2314/1182	1252	261	4.0	3.36
ImCLEF07A [8]	10000/1006	0/80	96	3.0	3.0
ImCLEF07D [8]	10000/1006	0/80	46	3.0	3.0
Interpro–FunCat [6]	2455/1264	2816	263	4.0	3.34
Reuters [15]	3000/3000	0/47236	100	4.0	3.20
SCOP-GO [6]	6507/3336	0/2003	523	5.5	6.26
Sequence-FunCat [6]	2455/1264	2/4448	244	4.0	3.35
WIPO [17]	1352/358	0/74435	183	4.0	4.0
Yeast-GO [6]	2310/1155	5588/342	133	6.3	5.74

level 0.05. The results of the statistical analysis are presented with *average ranking diagrams*. They plot the average ranks of the algorithms and connect those whose average ranks differ by less than the *critical distance*. The performance of the algorithms connected with a line is not statistically significantly different.

4 Results and Discussion

We present our experimental results as graphs with size on the horizontal axis and performance on the vertical axis. Figure 2 shows the results on four datasets. The remaining graphs are very similar and are omitted for brevity. Notably, the figures are on separate scales and on some figures the differences in performance between the different models are very small, e.g., on the SCOP-GO dataset.

Observing the points representing the results of OPCTs, the trade-off between size and performance is clearly visible. This trade-off is achieved as a consequence of different choices of the parameter values. The models' predictive performance generally rises with increasing model size, indicating that even the largest OPCTs do not overfit the training set, or possibly, different options overfit different parts of the input space. The increase in predictive performance in terms of increasing size also appears to saturate at the higher values of the observed parameter settings. This indicates that learning even larger less-restrictive OPCTs is not likely to provide a significant boost to predictive performance.

Compared to a PCT, OPCTs generally produce more accurate models that are mostly much larger. However, the increase in predictive performance is often

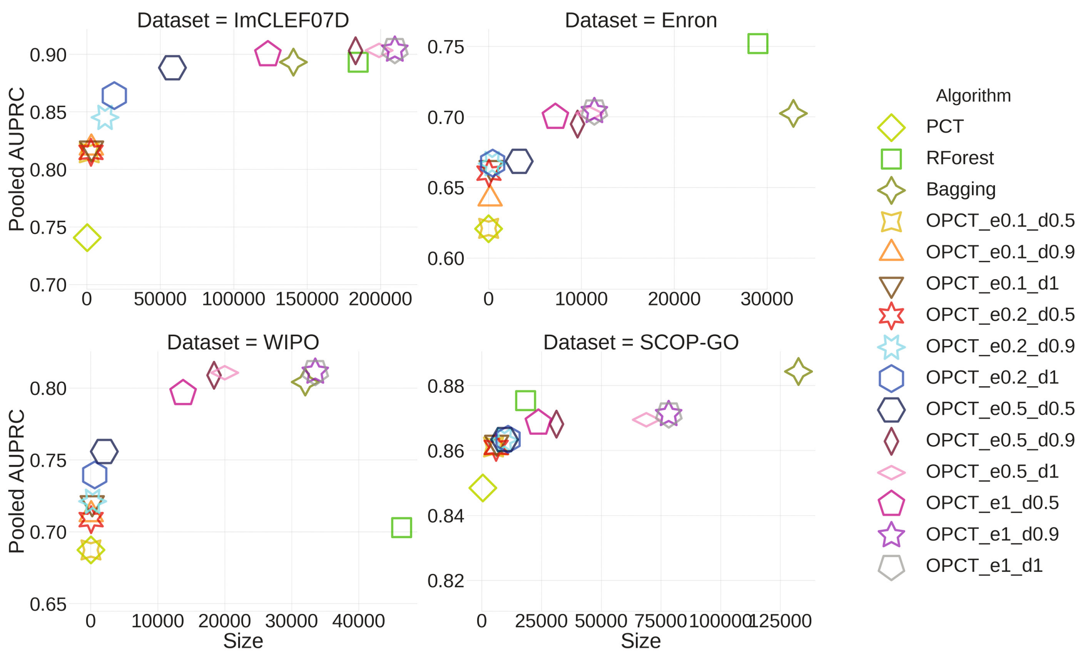
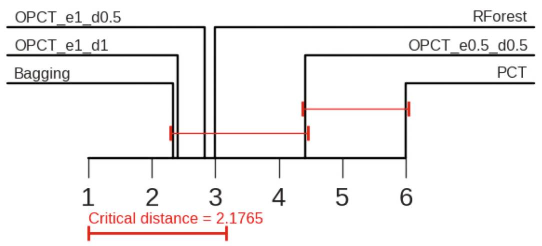


Fig. 2. Performances and sizes of models produced by different methods

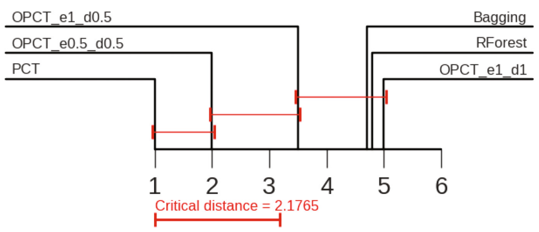
noticeable even for the lowest parameter values when the difference in size is relatively small. The comparison between OPCTs and ensembles of PCTs is more varied. Bagging of PCTs is usually better than OPCTs (SCOP-GO), though often very slightly (Enron) and sometimes worse (IMCLEF07D). However, the size of a bagging ensemble can considerably surpass the size of even the largest OPCTs. On the Enron dataset, random forests of PCTs outperform all other methods by a solid margin. They also provide good performance on the SCOP-GO dataset with relatively small trees, however, on the WIPO dataset they produce the largest model which only outperforms a PCT.

We selected 3 parameter configurations as trade-off points between predictive performance and model size: OPCT_e1_d1, as it offers the best performance, OPCT_e1_d0.5, as its performance was similar to that of OPCT_e1_d1 but it often produced noticeably smaller models, and OPCT_e0.5_d0.5, as it consistently produced much smaller models than other two selected configurations, albeit at the cost of some performance.

We compared the performance and size of these three configurations to that of a PCT and their ensembles, using Friedman test to check if there is a significant difference between the algorithms and the Nemenyi post-hoc test to show where the differences occur. Results are presented in Fig. 3. The performance of a PCT and its size is significantly lower than that of ensembles of PCTs, OPCT_e1_d1 and OPCT_e1_d0.5. Additionally, the size of OPCT_e0.5_d0.5 is significantly lower than that of the four aforementioned methods, but its performance is not. We also observe that the average rank of OPCT_e1_d0.5 in performance is on par with ensembles of PCTs (it placed between bagging and random forests), while its average rank in size is noticeably better. As expected, a PCT always produced the smallest model with the worst performance.



(a) Performance



(b) Size

Fig. 3. Average ranking diagrams of the performance and size of selected methods

5 Conclusions

In this work, we proposed an algorithm for learning option predictive clustering trees (OPCTs) for the hierarchical multi-label classification task. The purpose of OPCTs is to address the greediness of the standard algorithm for PCT learning. We experimentally evaluated the proposed method with various parameter values and compared it to PCTs and ensembles of PCTs (bagging and random forests). The results show that increasing the values of e and d increases the model performance and size compared to PCTs. At the highest parameter values of $e = 1$, $d = 1$, OPCTs are comparable to the state-of-the-art ensemble methods of bagging and random forest of PCTs.

We identified three interesting parameter selections for OPCTs and performed statistical comparison of these three methods and regular PCTs and their ensembles. The results show that regular PCTs have significantly lower performance and size than other methods with the exception of OPCT_e0.5_d0.5. Additionally, OPCT_e0.5_d0.5 produces significantly smaller models than bagging of PCTs, random forests of PCTs and OPCT_e1_d1. Average performance ranks of bagging, random forests, OPCT_e1_d1 and OPCT_e1_d0.5 are very similar, while average size rank of OPCT_e1_d0.5 is noticeably lower than that of the other three methods. Based on these results, we suggest the parameter values of $e \in \{0.5, 1\}$ and $d \in \{0.5, 1\}$ for future analyses.

There are several avenues for further work. Notably, the OPCT methodology described in this paper can be easily applied to the task of multi-label classification. In the future, we also plan to use the OPCT methodology as a part of a guided process to produce regular PCTs though either input from a domain expert, or through the use of additional validation data. Finally, we will investigate the use of OPCTs for performing feature ranking and selection for HMLC.

Acknowledgments. We acknowledge the financial support of the European Commission through the grants ICT-2013-612944 MAESTRA and ICT-2013-604102 HBP, as well as the support of the Slovenian Research Agency through young researcher grants and the program Knowledge Technologies (P2-0103).

References

1. Blockeel, H., Struyf, J.: Efficient algorithms for decision tree cross-validation. J. Mach. Learn. Res. **3**, 621–650 (2002)
2. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
3. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
4. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman & Hall/CRC, London (1984)
5. Buntine, W.: Learning classification trees. Stat. Comput. **2**(2), 63–73 (1992)
6. Clare, A.: Machine learning and data mining for yeast functional genomics. Ph.D. thesis, University of Wales Aberystwyth, Aberystwyth, Wales, UK (2003)
7. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
8. Dimitrovski, I., Kocev, D., Loskovska, S., Dzeroski, S.: Hierarchical annotation of medical images. Pattern Recogn. **44**(10–11), 2436–2449 (2011)
9. Dimitrovski, I., Kocev, D., Loskovska, S., Dzeroski, S.: Hierarchical classification of diatom images using ensembles of predictive clustering trees. Ecol. Inf. **7**(1), 19–29 (2012)
10. Ikonomovska, E., Gama, J., Zenko, B., Dzeroski, S.: Speeding-up hoeffding-based regression trees with options. In: Proceedings of the 28th International Conference on Machine Learning, ICML 2011, pp. 537–544 (2011)
11. Klimt, B., Yang, Y.: The enron corpus: a new dataset for email classification research. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS, vol. 3201, pp. 217–226. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30115-8_22
12. Kocev, D., Struyf, J., Džeroski, S.: Beam search induction and similarity constraints for predictive clustering trees. In: Džeroski, S., Struyf, J. (eds.) KDID 2006. LNCS, vol. 4747, pp. 134–151. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75549-4_9
13. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Tree ensembles for predicting structured outputs. Pattern Recogn. **46**(3), 817–833 (2013)
14. Kohavi, R., Kunz, C.: Option decision trees with majority votes. In: Proceedings of the 14th International Conference on Machine Learning, ICML 1997, pp. 161–169. Morgan Kaufmann Publishers Inc., San Francisco (1997)
15. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A new benchmark collection for text categorization research. J. Mach. Learn. Res. **5**, 361–397 (2004)
16. Osojnik, A., Džeroski, S., Kocev, D.: Option predictive clustering trees for multi-target regression. In: Calders, T., Ceci, M., Malerba, D. (eds.) DS 2016. LNCS, vol. 9956, pp. 118–133. Springer, Cham (2016). doi:10.1007/978-3-319-46307-0_8
17. Rousu, J., Saunders, C., Szedmak, S., Shawe-Taylor, J.: Kernel-based learning of hierarchical multilabel classification models. J. Mach. Learn. Res. **7**, 1601–1626 (2006)
18. Schietgat, L., Vens, C., Struyf, J., Blockeel, H., Kocev, D., Džeroski, S.: Predicting gene function using hierarchical multi-label decision tree ensembles. BMC Bioinform. **11**(2), 1–14 (2010)
19. Silla, C., Freitas, A.: A survey of hierarchical classification across different application domains. Data Min. Knowl. Disc. **22**(1–2), 31–72 (2011)
20. Vens, C., Struyf, J., Schietgat, L., Džeroski, S., Blockeel, H.: Decision trees for hierarchical multi-label classification. Mach. Learn. **73**(2), 185–214 (2008)

Chapter 7

Oblique Predictive Clustering Trees

This chapter presents our contributions concerning oblique predictive clustering trees. We provided a general overview of SPYCTs in Section 3.2 and described various predictive modeling tasks in Section 2.1. Here, we present both the SVM and the gradient variants of SPYCTs for supervised learning and their experimental evaluation for six predictive modeling tasks, including both structured and primitive outputs, as the approach is novel for both. The main contributions are the following:

1. Development of PCTs with oblique splits: SVM-based and gradient-descent-based methods proposed.

2. Extensive experimental evaluation of the proposed methods on benchmark datasets for six predictive modeling tasks: binary classification (BC), multi-class classification (MCC), multi-label classification (MLC), hierarchical multi-label classification (HMLC), single-target regression (STR), and multi-target regression (MTR).

3. Parameter sensitivity analysis of the proposed methods.

4. An algorithm for extracting feature importances from the learned oblique trees.

5. Experimental evaluation of the proposed feature importance extraction approach.

The initial work was presented at the ISMIS-2020 conference (Stepišnik & Kocev, 2020b). It contained an initial proposal of the gradient SPYCT variant for classification tasks. The proposed method was experimentally evaluated on 12 benchmark datasets, 3 for each of the 4 classification tasks (BC, MCC, MLC, HMLC). The results show that SPYCTs mostly achieve better performance than standard axis-parallel PCTs both in single-tree and ensemble settings and confirm the theoretical advantage in computational complexity.

The conference publication was greatly expanded and submitted to the Knowledge-Based Systems journal (Stepišnik & Kocev, 2020c). The paper presenting the extended work is included in this chapter. It includes the proposal of the SVM SPYCT variant, a revised version of the gradient variant (GRAD-SPYCT), and implementations that can exploit sparsity in data. The experimental evaluation was much more extensive, including STR and MTR tasks and 10 benchmark datasets per task, for a total of 60 datasets. Additional datasets were used for parameter sensitivity analysis. For comparison, we used different state-of-the-art baselines for different tasks as well as previously proposed oblique decision tree methods.

Overall, the results show that the performance of SPYCT ensembles is at least on par with other state-of-the-art methods. Bagging ensembles of GRAD-SPYCTs achieved the lowest average ranks on 3 tasks and were close to the top on the other 3. The performance of the SVM-SPYCTs was slightly behind the GRAD-SPYCTs when used in ensembles, however, it was slightly better in single-tree settings. This indicates that GRAD-SPYCT trees might overfit the learning data more than SVM-SPYCTs trees, which is helpful for ensembles but hurts individual trees. On datasets with high-dimensional inputs or outputs, the learning times for both variants were orders of magnitudes lower than for standard PCTs. We also showed that using sparse data representation where appropriate gives additional computational benefits. The experimental results confirmed our theoretical analysis and additionally indicated that using matrix operations to do the most computationally intensive tasks can be beneficial. While the theoretical complexity is the same, the hardware is very well optimized for such operations. We also showed that the feature importances extracted from oblique trees with the proposed method are meaningful and successfully identify irrelevant features.

We also briefly explored random forests of SPYCTs, where each oblique split is learned on a different random subset of features. In principle, the advantage over bagging ensembles is better scaling with the number of features, same as for axis-parallel splits. In the majority of cases, the results were good and similar to bagging ensembles. However, on some very sparse datasets and datasets with several one-hot encoded attributes, the results were surprisingly poor. An unlucky subset selection in such scenarios prevented the learning of a useful split and the trees stopped growing. This issue can be addressed in different ways, including using different attribute encoding, giving the algorithm multiple tries to find a split, changing the pre-pruning settings, etc. This makes random forest ensembles a promising avenue for further work.

From the hypotheses we defined in Section 1.2, this chapter addresses the following:

H3 Oblique predictive clustering trees reduce learning time on high dimensional data compared to standard PCTs without deteriorating predictive performance.

H4 Oblique predictive clustering trees exploit sparse data to reduce learning time.

H6 Meaningful feature importances are calculated by using oblique PCTs.

All three hypotheses were confirmed. For H3, not only did predictive performance not deteriorate, SPYCTs generally performed better than standard PCTs in addition to their clear computational advantages. The computational advantage addressed in H3 and H4 was confirmed with both theoretical and empirical analyses. The proposed methods are therefore computationally efficient, versatile, and good predictors.

The paper included in this Chapter is:

- Stepišnik, T., Kocev, D. (2020c). Oblique predictive clustering trees. *Knowledge-Based Systems [Under Review].*

The contributions of Tomaž Stepišnik to this paper are as follows. He designed and implemented the SVM and gradient variants of oblique predictive clustering trees. He participated in the design of the study, carried out the experiments, and analyzed the results. He drafted the paper and revised it following the feedback from the co-authors and the reviewers.

Oblique Predictive Clustering Trees

Tomaž Stepišnik[a,b,*], Dragi Kocev[a,b]

[a] *Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia*
[b] *Jožef Stefan International Postgraduate School, Ljubljana, Slovenia*

Abstract

Predictive clustering trees (PCTs) are a well-established generalization of standard decision trees, which can be used to solve a variety of predictive modeling tasks, including structured output prediction. Combining them into ensembles of PCTs yields state-of-the-art performance. Furthermore, the ensembles of PCTs can be interpreted by calculating feature importance scores from the learned models. However, their learning time scales poorly with the dimensionality of the output space. This is often problematic, especially in (hierarchical) multi-label classification, where the output can consist of hundreds or thousands of potential labels. Also, learning of PCTs can not exploit the sparsity of data to improve computational efficiency. Data sparsity is common in both input (molecular fingerprints, bag of words representations) and output spaces (in multi-label classification, examples are often labeled with only a handful of labels out of a much larger set of potential labels). In this paper, we propose oblique predictive clustering trees, capable of addressing these limitations. We design and implement two methods for learning oblique splits that contain linear combinations of features in the tests, hence a split corresponds to an arbitrary hyperplane in the input space. The resulting oblique trees are efficient for high dimensional data and are capable of exploiting sparse data. We experimentally evaluate the proposed methods on 60 benchmark datasets for 6 predictive modeling tasks: binary classification, multi-class classification, multi-label classification, hierarchical multi-label classification, single-target regression, and multi-target regression. The results of the experiments show that oblique predictive clustering trees achieve performance on par with state-of-the-art methods and are orders of magnitude faster than standard predictive clustering trees. We also show that meaningful feature importance scores can be extracted from the models learned with the proposed methods.

Keywords: Oblique decision trees, predictive clustering trees, ensembles, structured output prediction, sparse data

[*] Corresponding author
Email addresses: `tomaz.stepisnik@ijs.si` (Tomaž Stepišnik), `dragi.kocev@ijs.si` (Dragi Kocev)

1. Introduction

In predictive modeling, a set of learning examples is used to induce a model that can be used to make accurate predictions for unseen examples. The examples are described with features and are associated with a target variable. The model uses the features to predict the value of the target variable. In most common tasks, there is only one target variable. If it is numeric, the task is called regression; if the target is discrete, the task is called classification.

However, many real-life problems can be more naturally represented with more complex targets composed of multiple variables that can have additional structure or dependencies among them. Such problems are encountered in a variety of disciplines: life sciences (e.g., gene and protein function prediction), environmental sciences (e.g., soil functions, habitat modeling), text and image analysis (e.g., document classification, image annotation) and others. These problems are addressed with the task of *structured output prediction* (SOP). In this work, we focus on three types of SOP tasks:

- *Multi-target regression* (MTR) is a predictive modeling task where the target is a vector with numeric components/variables.
- *Multi-label classification* (MLC) is a generalization of the standard classification where each example can be a member of multiple classes: the targets are subsets of a finite set of possible labels.
- *Hierarchical multi-label classification* (HMLC) further extends the task of MLC by additionally considering a partial order on the set of possible labels, i.e., some labels are specialized cases of other labels. The partial order organizes the labels into a hierarchy that can be represented with a directed acyclic graph.

Predictive clustering trees are a variant of decision trees that have been successfully applied to various predictive modeling tasks, including structured output prediction and semi-supervised learning. However, they scale poorly to problems with many target variables and cannot take advantage of sparsity in the data. Both of these properties are quite common, especially in structured output prediction problems. For example, in (H)MLC problems there are often hundreds or thousands of possible labels (many target variables). Additionally, each example is typically labeled with only a handful of labels, which leads to sparse target matrices. Data can also be sparse on the input side, e.g., compounds described with binary fingerprints, text described with bag of words representations, etc.

In this paper, we propose two methods for learning oblique predictive clustering trees, designed to improve the efficiency of learning on high dimensional and/or sparse data. The first variant of oblique predictive clustering trees is based on SVM and the second on gradient descent. We experimentally evaluate the proposed methods and show that they achieve state-of-the-art performance and are learned orders of magnitude faster than axis-parallel trees. We also perform parameter sensitivity analysis and demonstrate that meaningful feature importance scores can be extracted from the learned models.

Initial experiments on classification with the gradient descent variant were presented in [1]. The method has since been revised and improved with better support sparse data and regularization to reduce model sizes. In addition to the initial study, this paper proposes a new method (the SVM-based) and more extensive experiments that include single-target classification and regression as well as the three above-mentioned SOP tasks (MTR, MLC, and HMLC).

The remainder of the paper is organized as follows. In Section 2, we present the background related to this paper. In Section 3, we describe and analyze our proposed methods in detail. Next, we present our experimental setup (Section 4) and results from the benchmarking experiments (Section 5). We then illustrate the meaningfulness of the feature importances extracted from our models (Section 6) and present the results of parameter sensitivity analysis (Section 7). In Section 8, we conclude the paper with a summary of the main findings.

2. Background

The research of tree-based predictive models was popularized in the 1980s [2]. Their use is widespread: they can be used for classification and regression tasks and can handle both numeric and nominal features. A single tree can be inspected and its predictions interpreted easily. When used in ensembles [3, 4] they can achieve state-of-the-art performance. The performance boost comes at the cost of reduced interpretability, hence, different feature ranking approaches based on trees and tree ensembles have also been developed [4, 5].

Predictive clustering trees [6, 7] generalize standard decision/regression trees by differentiating between three types of attributes: features, clustering attributes, and targets. Features are used to divide the examples; these are the attributes encountered in the split nodes. Clustering attributes are used to calculate the heuristic which guides the search of the best split at a given node. Targets are the attributes predicted in the leaves. The algorithm for learning PCTs follows the top-down induction algorithm described by [2], and is presented in Algorithm 1.

The algorithm takes as input matrices of features (X), clustering attributes (Y), and targets (Z). It then goes through all features and searches for a test that maximizes the heuristic score. The heuristic that is used to evaluate the tests is the reduction of impurity caused by splitting the data according to a test. It is calculated on the clustering attributes. If no acceptable split is found (e.g., no test reduces the variance significantly, or the number of examples in a node is below a user-specified threshold), then the algorithm creates a leaf and computes the prototype of the targets of the instances that were sorted to the leaf. The selection of the impurity and prototype functions depends on the types of clustering attributes and targets (e.g., variance and mean for regression, entropy and majority class for classification).

In theory, clustering attributes can be completely independent of the features and the targets. In practice, the ultimate goal is to make accurate predictions

Algorithm 1 Learning a PCT: The inputs are matrices of features $X \in \mathcal{R}^{N \times D}$, targets $Y \in \mathcal{R}^{N \times T}$ and clustering attributes $Z \in \mathcal{R}^{N \times K}$.

```
procedure GROW_TREE(X, Y, Z)
    test = best_test(X, Z)
    if acceptable(test) then
        rows1, rows2 = split(X, test)
        left_subtree = grow_tree(X[rows1], Y[rows1], Z[rows1])
        right_subtree = grow_tree(X[rows2], Y[rows2], Z[rows2])
        return Node(test, left_subtree, right_subtree)
    else
        return Leaf(prototype(Y))
procedure BEST_TEST(X, Z)
    best = None
    for $d = 1, \ldots, D$ do
        for test ∈ possible_tests(X, d) do
            if score(test, X, Z) > score(best, X, Z) then
                best = test
    return best
procedure SCORE(test, X, Z)
    rows1, rows2 = split(X, test)
    n1 = num_rows(rows1)
    n2 = num_rows(rows2)
    n = n1 + n2
    return $n\cdot$ impurity(Z) - $n1\cdot$ impurity(Z[rows1]) - $n2\cdot$ impurity(Z[rows2])
```

for the targets, and the splitting heuristic should reflect that. The most basic (and common) approach is to use the targets also as clustering attributes. For example, in a classification problem, doing so makes PCTs equivalent to standard decision trees. But the attribute differentiation gives PCTs a lot of flexibility. They have been used for predicting various structured outputs [8]. In addition to targets, we can also include features among the clustering attributes. This makes leaves homogeneous also in the input space, which is helpful if the targets are noisy, and can also be used for semi-supervised learning [9, 10]. Embeddings of the targets have also been used as clustering attributes in order to reduce the time complexity of tree learning [11].

Most of the research focus, including all of the above-mentioned work, is on *axis-parallel* trees. The tests in axis-parallel trees only use single features and the splits are axis-parallel hyperplanes in the input space. Alternatively, *oblique* trees (also called multivariate trees) use linear combinations of features in the tests, which allow for splits corresponding to an arbitrary hyperplane in the input space. They receive a lot less attention than axis-parallel trees, possibly due to the increased complexity of optimizing the split. Finding a hyperplane that splits a set of examples with binary labels in a way that minimizes misclassified examples on both sides of the hyperplane is an NP-hard problem [12]. But the

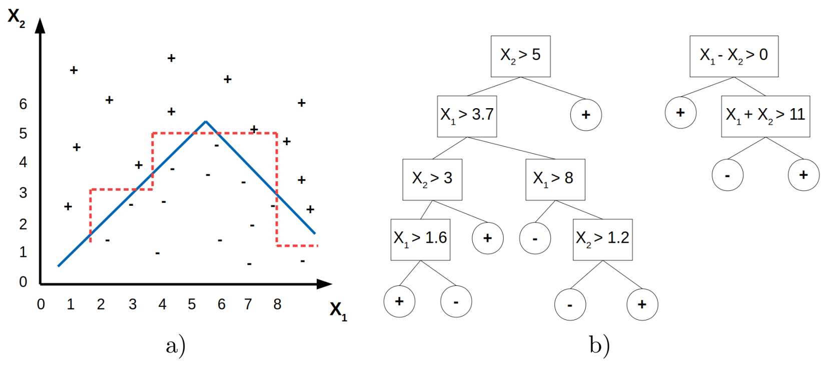


Figure 1: A toy dataset (a) with drawn decision boundaries learned by axis-parallel (red, dashed) and oblique (blue, solid) decision trees (b).

increased flexibility of splits can lead to models that fit the data much better as illustrated in Figure 1.

The initial proposals [2, 13] for the induction of oblique trees relied on local search optimization and scale poorly to contemporary problems with thousands of examples and/or features. Weighted Oblique Decision Trees [14] learn splits by optimizing weighted information entropy and are a relatively efficient method for binary and multi-class classification. Oblique random forests [15] follow the standard Random Forest paradigm of learning trees on different bootstrapped samples of the training set and searching for each split in a different subset of features. Split learning is performed via ridge regression, which allows for efficient optimization of the hyperplanes, but their approach is limited to binary classification problems. Additionally, FastXML [16] and PfastreXML [17] present methods based on oblique trees for multi-label classification which work efficiently even with thousands of features and labels. When learning a split, these methods optimize a complex objective function that combines L1 regularization, logarithmic loss and normalized Distributed Cumulative Gain (nDCG) of examples on each side of the hyperplane. The criterion function optimization is performed in two steps. First, the examples are partitioned in a way that minimizes nDCG in each partition; this partitioning is improved iteratively. Then, a hyperplane is learned that approximates this partitioning as best as it can. This is done by using GLMNET method to solve a logistic regression problem.

Existing oblique tree methods are specialized for specific predictive tasks and/or computationally inefficient. In this paper, we present a method that combines the flexibility of PCTs, uses oblique splits, applicable to a variety of predictive modeling tasks, and is computationally efficient.

3. Method description

In this section, we describe oblique predictive clustering trees. We also follow the top-down induction algorithm presented in Algorithm 1, but we modify the split searching procedure (the BEST_TEST function) to learn oblique tests. We propose two modifications of the BEST_TEST function based on SVMs and gradient descent.

We start with the introduction of the notation used throughout the manuscript. Let $X \in \mathcal{R}^{N \times D}$ be the matrix containing the D features of the N examples in the learning set, $Y \in \mathcal{R}^{N \times T}$ be the matrix containing the T targets associated with the N examples in the learning set, and $Z \in \mathcal{R}^{N \times K}$ be the matrix containing the C clustering attributes associated with the N examples in the learning set. This formulation encapsulates several predictive modeling tasks. First, single-target regression ($T = 1$) and multi-target regression ($T > 1$) fit this description easily. Second, for classification problems, the Y matrix contains binary values: The value $y_{ij} = 1$ if the i-th example has j-th label, otherwise $y_{ij} = 0$. Third, for binary classification problems, the number of targets is $T = 1$. Next, for multi-class classification with k possible classes, the class information can be encoded via one-hot encoding ($T = k$). Each row in Y has exactly one element set to 1, and the rest are set to 0. Note that similarly as for the targets, nominal features across all tasks are supported via one-hot encoding. Finally, label sets in (hierarchical) multi-label classification with k possible labels can be encoded with binary vectors ($T = k$), but here each row can have multiple elements set to 1.

Below, $M_{i.}$ will refer to the i-th row and $M_{.i}$ to the i-th column of the matrix M. Let `colmean`(M) be a vector of column means of the matrix M. We wish to learn a vector of weights $w \in \mathcal{R}^D$ and a bias term $b \in \mathcal{R}$ that define a hyperplane, which splits the learning examples into two subsets. We will refer to the subset where $X_{i.} \cdot w + b \geq 0$ as the positive subset, and the subset where $X_{i.} \cdot w + b < 0$ as the negative subset. We want the examples in the same subset to have similar values of clustering attributes in Z. Prior to learning each split, features and clustering attributes are standardized to mean 0 and standard deviation 1. After learning the split, the tree construction continues recursively on the positive and negative subsets. As with standard PCTs, the learning stops when the learned split is not acceptable, at which point a leaf node is made where the prototype of the targets is stored. As prototypes, we use target means over the examples in the leaf, i.e., `colmean`(Y). This prototype can be used as-is to make predictions (regression problems, pseudo-probabilities for classification problems), or it can be used to predict the majority class, or all labels with frequencies above a specified threshold (multi-label classification).

When making a prediction for a new example with features $x \in \mathcal{R}^D$, we calculate the value $x \cdot w + b$ in the root node. If it is non-negative, we repeat the process in the positive subtree, otherwise in the negative subtree. This is done until a leaf node is reached, where a prediction is made.

3.1. SVM-based split learning

In the SVM-based split learning, we perform hyperplane optimization in two steps. First, we group the examples into two subsets in such a way that the similarity of clustering attributes in each subset is maximized, regardless of the values of the features. This is calculated using only the Z matrix, while X is ignored. For example, in binary classification the subset partitioning is clear: each subset should contain examples from one class. For other tasks, the subsets are not obvious. To obtain them we use the k-means clustering [18] to cluster rows in Z into two clusters. Let vector $c \in \{0,1\}^N$ be the result of the clustering.

Next, we look for a hyperplane in the input space that approximates this partitioning. In essence, we convert the split hyperplane optimization problem into a binary classification problem. We solve the following problem:

$$\min_{w,b} \; ||w||_1 + C \sum_{i=1}^{N} max(0, 1 - c_i(X_{i.} \cdot w + b))^2,$$

where parameter $C \in \mathcal{R}$ determines the strength of regularization. To solve this problem we use the LIBLINEAR library [19], specifically their $L1$-regularized $L2$-loss support vector classification.

3.2. Gradient-descent-based split learning

In the gradient-descent-based split learning, we directly search for a hyperplane split that minimizes the impurity on both sides of the hyperplane, unlike in the SVM-based split learning, where the split hyperplane is optimized indirectly, by first determining an ideal split and then trying to approximate it with a hyperplane. First, we calculate the vector $s = \sigma(Xw + b) \in [0,1]^N$, where the sigmoid function σ is applied component-wise. The vector s contains values from the $[0,1]$ interval, and we treat it as a fuzzy membership indicator. Specifically, the value s_i tells us how much the i-th example belongs to the positive subset, whereas the value $1 - s_i$ tells us how much it belongs to the negative subset.

To measure the impurity of the positive subset, we calculate the weighted variance of each column (attribute) in Z, and we weigh each row (example) with its corresponding weight in s. To measure the impurity of the negative subset, we calculate the weighted variances with weights from $1 - s$. Weighted variance of a vector $v \in \mathcal{R}^n$ with weights $a \in \mathcal{R}^n$ is defined as

$$\texttt{var}(v,a) = \frac{\sum_{i=1}^{n} a_i (v_i - \texttt{mean}(v,a))^2}{A} = \texttt{mean}(v^2, a) - \texttt{mean}(v,a)^2,$$

where $A = \sum_{i=1}^{n} a_i$ is the sum of weights and $\texttt{mean}(v,a) = \frac{1}{A}\sum_{i=1}^{n} a_i v_i$ is the weighted mean of v.

The impurity of a subset is the weighted sum of weighted variances over all the clustering attributes. The impurity of the positive subset is $\texttt{imp}(Z,p,s) =$

$\sum_{j=1}^{L} p_j \mathtt{var}(Z_{.j}, s)$, and similarly $\mathtt{imp}(Z, p, 1-s)$ is the impurity of the negative subset. Weights $p \in \mathcal{R}^K$ enable us to give different priorities to different clustering attributes. The final split fitness function is

$$f(w, b) = S * \mathtt{imp}(Z, p, s) + (N - S) * \mathtt{imp}(Z, p, 1 - s),$$

where $s = \sigma(Xw + b)$ and $S = \sum_{i=1}^{N} s_i$. The terms S and $N - S$ represent the sizes of positive and negative subsets, and are added to guide the split-search procedure towards balanced splits. Finally, to obtain the split hyperplane, the following optimization problem is solved:

$$\min_{w,b} \ ||w||_{\frac{1}{2}} + Cf(w, b),$$

where C again controls the strength of regularization and $||w||_{\frac{1}{2}} = (\sum_{i=1}^{D} \sqrt{|w_i|})^2$ is the L$\frac{1}{2}$ norm. We selected L$\frac{1}{2}$ regularization because it induces weight sparsity more aggressively than L1, which helps reducing the model size.

For examples that are not close to the hyperplane, s_i is close to 0 or 1. Weighted variances are therefore approximations of the variances of the subsets, that the hyperplane produces. This formulation makes the objective function differentiable and enables us to use the efficient Adam [20] gradient descent optimization method. The weight vector w is initialized randomly, then b is set such that the examples are initially split in half.

3.3. Time complexity analysis

In this section, we analyze the time complexity of learning oblique predictive clustering trees and compare it to the time complexity of learning standard PCTs. We focus on the cost of learning a split since this is the only significant difference between the two methods.

Let us start with the SVM variant. The first step is to perform 2-means clustering on the clustering data $Z \in \mathcal{R}^{N \times K}$, with time complexity $O(NKI_c)$, where I_c is the number of clustering iterations we perform. Learning the linear SVM to differentiate the clusters costs $O(NDI_o)$, where I_o is the number of optimization iterations. The total cost is then $O(N(KI_c + DI_o))$.

For the gradient descent variant, the main operation is gradient calculation. The most expensive part of it is the multiplications of matrices X and Z with vectors, which costs $O(ND)$ and $O(NK)$, respectively. The total cost of learning a split is then $O(NI_o(D + K))$, where I_o is again the number of optimization iterations.

The time complexity of learning a split in standard PCTs is $O(DN \log N + NDK)$ [8]. The important difference we can notice is that standard PCTs scale with DK, whereas both proposed variants of oblique PCTs scale with $D + K$. This difference is very noticeable when solving problems with many clustering variables, e.g., (hierarchical) multi-label classification and semi-supervised learning. An additional benefit of our approach can be obtained when dealing with sparse data. If the features and/or clustering data is sparse, both variants can exploit the sparsity by performing operations with sparse matrices. This reduces the time complexity to $O(N(\hat{K}I_c + \hat{D}I_o))$ for the SVM variant and $O(NI_o(\hat{D}+\hat{K})$ for the gradient descent variant, where $\hat{D} \ll D$ and $\hat{K} \ll K$ are average numbers of non-zero elements in each row of matrices X and Z.

3.4. Ensembles of oblique predictive clustering trees

Single decision trees are mainly used when we wish to visually inspect and interpret the model. To achieve state-of-the-art performance, trees have to be used in ensembles. Oblique trees are inherently more difficult to visually interpret, because of the linear combinations in the split nodes. Also, because the splits are more complex, they can very easily overfit the data in the deeper nodes. For these reasons, we believe the most natural use of oblique PCTs is by combining them in ensembles.

To construct bagging ensembles [3], we simply construct each oblique PCT on a different bootstrapped sample of the learning set. When making predictions, we average the prototypes predicted by each tree in the ensemble to get the final prediction. We can also build random forest ensembles [4] of oblique PCTs. In addition to bootstrapping, each split hyperplane is only learned on a random subset of features. After the hyperplane weights are learned, the w_i for features that were not in the selected subset are set to 0.

3.5. Feature importance

Even though ensembles of oblique PCTs are hard to interpret directly, we can still gain insight into how the models make their decision by calculating feature importances, much like it is done with ensembles of axis-parallel trees. The feature importance scores of a single oblique PCT are calculated as follows:

$$imp(T) = \sum_{s \in T} \frac{s_n}{N} \frac{s_w}{\|s_w\|_1},$$

where s iterates over split nodes in tree T, s_w is the weight vector defining the split hyperplane, s_n is the number of learning examples that were present in the node and N is the total number of learning examples. The contributions of each node to the final feature importance scores are weighted according to the number of examples that were used to learn the split. This puts more emphasis on weights higher in the tree, which affect more examples. To get feature importance scores of an ensemble, we simply average feature importances of individual trees in the ensemble.

3.6. Implementation

We implemented the proposed oblique predictive clustering trees in a python package *spyct*. It is freely licensed and available for use and download at `https://gitlab.com/TStepi/spyct`. The package includes both SVM and gradient descent variant, single trees, bagging, and random forest ensembles. In ensembles, trees can be built in parallel. We also provide a number of pre-pruning options: maximum tree depth, minimum number of examples required to perform a split, and the minimum reduction of impurity required for a split to be accepted. By default, tree depth is not limited, a split is always attempted if more than 1 example is still present in a node, and it is accepted if the impurity is reduced by at least 5% in at least one of the subsets. Additionally, the splitting stops if one of the subsets is empty (the hyperplane does not split the data). We can control the maximum number of clustering iterations (SVM variant, 10 by default), learning rate (gradient variant, default 0.1), and the maximum number of hyperplane optimization iterations (both variants, default 100). We also make use of early stopping in both clustering and hyperplane optimization, if the process converges. Strength of regularization (both variants, $C = 10$ by default) and other parameters of the Adam optimizer (gradient variant, default values from PyTorch [1] library) are also configurable.

4. Experimental setting

In this section, we present a comprehensive experimental study designed to evaluate the proposed oblique PCTs and compare them to standard PCTs and other baseline methods. We evaluated the methods on benchmark datasets from the classification spectrum: binary (BIN), multi-class (MCC), multi-label (MLC), and hierarchical multi-label classification (HMLC), as well as from the regression spectrum: single-target (STR) and multi-target regression (MTR). We first present the competing methods, and then the benchmark data and evaluation strategy.

4.1. Competing methods

The main baselines for comparison are standard PCTs, which can be used for all 6 predictive modeling tasks considered. We trained single trees, bagging ensembles of PCTs, and random forests of PCTs. Next, we compared to existing oblique tree methods in single tree settings, as they were proposed by the authors. This includes CART-LC [2], OC1 [13] and WODT [14]. Both CART-LC and OC1 are only applicable to BIN problems, whereas WODT can also be used for MCC.

For ensembles of oblique trees, we considered oblique random forests (ORF) [15] and the FastXML method [16] as competing methods. The ORF method is only applicable to BIN problems. The FastXML method is specialized for the MLC task and optimized to work with sparse data. We will also use it as a baseline for the HMLC datasets, by discarding the hierarchy and treating the problem as a flat MLC task.

Additionally, the comparison includes LightGBM gradient boosted tree ensembles [21]. They can be used for BIN, MCC, and STR tasks. Finally, to include a non-tree-based baseline, we used linear SVMs. They can be directly applied to BIN and STR tasks. We will also use them as baselines for other tasks,

[1]`https://pytorch.org/docs/stable/optim.html`

Method	Tasks	Implementation
SPYCT-SVM	all	`gitlab.com/TStepi/spyct`
SPYCT-GRAD	all	`gitlab.com/TStepi/spyct`
PCT	all	`http://source.ijs.si/ktclus/clus-public/`
SVM	all	`https://scikit-learn.org/stable/`
CART-LC	BIN	`github.com/AndriyMulyar/sklearn-oblique-tree`
OC1	BIN	`github.com/AndriyMulyar/sklearn-oblique-tree`
WODT	BIN, MCC	`www.lamda.nju.edu.cn/yangbb`
ORF	BIN	`rdrr.io/cran/obliqueRF/man/obliqueRF.html`
FastXML	MLC, HMLC	`manikvarma.org/code/FastXML/download.html`
LightGBM	BIN, MCC, STR	`lightgbm.readthedocs.io`

Table 1: Methods included in the benchmarking comparison.

Dataset	N	D	Dataset	N	D
ailerons [22]	13750	40	*puma8NH* [22]	8192	8
cpmp-2015 [22]	2108	23	*qsar-234* [22]	2145	1024
cpu_small [22]	8192	12	*satellite_image* [22]	6435	36
elevators [22]	16599	19	*space_ga* [22]	3107	6
house_8L [22]	22784	8	*triazines* [22]	186	60

Table 2: Properties of the benchmark STR datasets. Columns show the number of examples (N) and the number features (D).

Dataset	N	D	T	Dataset	N	D	T
atp1d [23]	337	411	6	*oes97* [23]	334	263	16
edm [23]	154	16	2	*rf1* [23]	9125	64	8
enb [23]	768	8	2	*rf2* [23]	9125	576	8
jura [23]	359	15	3	*scm1d* [23]	9803	280	16
oes10 [23]	403	298	16	*slump* [23]	103	7	3

Table 3: Properties of the benchmark MTR datasets. Columns show the number of examples (N), the number of features (D) and the number of targets (T).

by learning a separate SVM for each label/target (one-vs-all approach in MCC, binary relevance in MLC and HMLC, local approach to MTR). We decided to use the linear kernel to keep the learning time manageable and because it is closest to the nature of the splits used in the oblique trees.

Table 1 sums up the baseline methods and provides links to the implementations we used in the experiments. We will refer to the proposed methods as SPYCT-SVM and SPYCT-GRAD, for the SVM and gradient descent variant, respectively. For standard and oblique PCTs, we will prepend BAG or RF to mark bagging and random forest ensembles, respectively (e.g., BAG-SPYCT-GRAD denotes bagging ensembles of gradient descent variant oblique PCTs).

Dataset	N	D	Dataset	N	D
banknote [22]	1372	4	*musk* [22]	6598	166
bioresponse [22]	3751	1776	*OVA_Breast* [22]	1545	10935
credit-approval [22]	690	15	*OVA_Lung* [22]	1545	10935
credit-g [22]	1000	20	*spambase* [22]	4601	57
diabetes [22]	768	8	*speeddating* [22]	8378	120

Table 4: Properties of the benchmark BIN datasets. Columns show the number of examples (N) and the number of features (D).

Dataset	N	D	T
amazon-reviews [22]	1500	10000	50
balance [22]	625	4	3
diabetes130us [22]	101766	47	3
gas-drift [22]	13910	128	6
hepatitisC [22]	283	54621	3
isolet [22]	7797	617	26
mfeat-pixel [22]	2000	240	10
micro-mass [22]	571	1300	20
vehicle [22]	846	18	4
wine-white [22]	4898	11	7

Table 5: Properties of the benchmark MCC datasets. Columns show the number of examples (N), the number of features (D) and the number of classes (T).

Dataset	N	D	T	Sparse
bibtex [23]	7395	1836	159	D, T
bookmarks [23]	87856	2150	208	D, T
CAL500 [23]	502	68	174	T
corel5k [23]	5000	499	374	D, T
emotions [23]	593	72	6	
eurlex-eurovoc [23]	19348	5000	3993	D, T
flags [23]	194	19	7	
rcv1subset1 [23]	6000	47236	101	D, T
scene [23]	2407	294	6	
tmc2007 [23]	28596	49060	22	D

Table 6: Properties of the benchmark MLC datasets. Columns show the number of examples (N), the number of features (D), the number of targets/labels (T), and whether the input or output space is sparse.

4.2. Evaluation

We evaluated the methods on 60 benchmarking datasets, 10 for each of the 6 tasks. Their details are presented in Tables 2-7. If the input or output data matrix had fewer than 10% of nonzero values, it was represented in a sparse format (marked in the tables, where applicable). To measure predictive

Dataset	N	D	T	Sparse
enron [24]	1648	1001	56	D, T
imclef07d [24]	11006	80	46	T
reuters [24]	6000	47235	102	D, T
wipo [24]	1710	74435	188	D, T
yeast_GO [25]	3465	5930	133	D, T
yeast_spo_FUN [25]	3711	80	594	T
yeast_expr_FUN [25]	3788	551	594	T
ara_exprindiv_FUN [25]	3496	1251	261	T
yeast_gasch1_FUN [25]	3773	173	594	T
ara_interpro_GO [25]	11763	2815	630	D, T

Table 7: Properties of the benchmark HMLC datasets. Columns show the number of examples (N), the number of features (D), the number of targets/labels (T), and whether the input or output space is sparse.

performance of the methods on STR and MTR datasets, we used the *coefficient of determination* as performance measure

$$R^2(y, \hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2},$$

where y is the vector of true target values, $\bar{y}$ is their mean, and $\hat{y}$ is the vector of predicted values. For MTR problems, the mean of R^2 scores per target was calculated. For BIN and MCC tasks, we used *F1 score*, macro averaged in the MCC case.

Methods solving MLC and HMLC tasks typically return a score for each label and each example, higher score meaning that example is more likely to have that label. Let $y \in \{0,1\}^{n \times l}$ be the matrix of label indicators and $\hat{y} \in \mathcal{R}^{n \times l}$ the matrix of label scores returned by a method. We measured the performance of methods with weighted label ranking average precision

$$LRAP(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j : y_{ij} = 1} \frac{w_j}{W_i} \frac{L_{ij}}{R_{ij}},$$

where $L_{ij} = |\{k : y_{ik} = 1 \wedge \hat{y}_{ik} \geq \hat{y}_{ij}\}|$ is the number of real labels assigned to example i that the method ranked higher than label j, $R_{ij} = |\{k : \hat{y}_{ik} \geq \hat{y}_{ij}\}|$ is the number of all labels ranked higher than label j, w_j is the weight we put to label j and W_i is the sum of weights of all labels assigned to example i. For MLC tasks we put equal weights to all labels, whereas for HMLC task we weighted each label with 0.75^d, where d is the depth of the label in the hierarchy [8].

For all three measures used (R2, F1, LRAP), a higher value indicates better performance, with a value of 1 indicating the best possible performance. We estimated the predictive performance using 10-fold cross-validation on each dataset. In addition to performance measures, we also recorded the learning time of each method. All experiments were performed on the same computer and the methods were allowed to use up to 10 processor cores. We set the number of trees in the ensembles to 50. Random forest ensembles (RF-PCT, RF-spyct-svm, RF-spyct-grad, ORF) used $\sqrt{D}$ features for each split, where D is the number of features. For SVM and LightGBM methods, parameter tuning is advised. In experiments with these methods, we set aside 20% of the training set for each fold to tune the parameters. For SVM we selected the regularization parameter C from values $\{0.1, 1, 10, 100, 1000\}$. For LightGBM we selected the maximum number of leaves in the trees among values $\{20, 50, 100\}$ and the minimum number of samples to perform a split from values $\{1, 10, 100, 1000\}$. The ORF method includes internal optimization of its regularization parameter, which we left at its default setting. In the HMLC experiments, standard and oblique PCTs also received the same label weights used to calculate LRAP.

5. Results and discussion

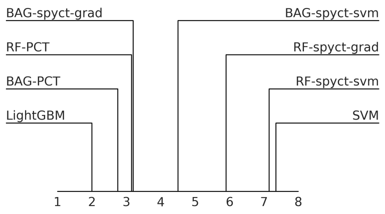
In this section, we present the results of our extensive benchmarking experiments. We first discuss the predictive performance of the proposed oblique PCTs in an ensemble setting. Next, we analyze the learning times, focusing on large and sparse datasets. Finally, we present a comparison of the proposed methods to other single-tree methods.

5.1. Predictive performance

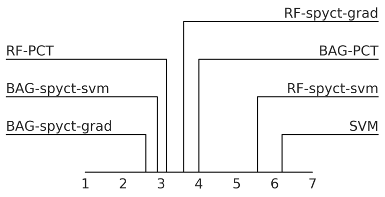
To keep the paper concise, we do not present the large number of raw performance results of our experiments here (they are available in the Appendix), but instead focus on the aggregated results presenting the overall findings. Figure 2 presents the average ranks of the methods based on predictive performance on the benchmark datasets, grouped by predictive modeling tasks. If a method did not finish the evaluation in 3 days, it was assigned the lowest rank. This only occurred for the MLC task, where BAG-PCT did not finish on *eurlex-eurovoc* and *tmc2007* datasets, and RF-PCT did not finish on *eurlex-eurovoc* dataset.

In a nutshell, the results show that in terms of predictive performance, the proposed bagging ensembles (BAG-SPYCT-SVM and BAG-SPYCT-GRAD) are on par with current state-of-the-art methods. Furthermore, BAG-SPYCT-GRAD achieves the best rank on 3 tasks (MTR, MCC, and MLC), and is close to the top on the other 3 tasks as well (close second on BIN and HMLC, where on the latter best performing is BAG-SPYCT-SVM).

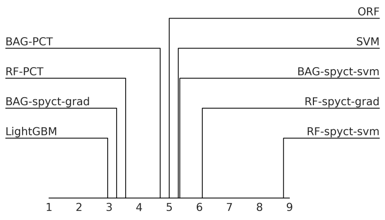
While the bagging ensembles of oblique PCTs have premium predictive performance, the proposed random forest ensembles of oblique PCTs (RF-SPYCT-SVM and RF-SPYCT-GRAD) did not perform well. Especially RF-SPYCT-SVM struggled to learn on several datasets. We believe this is the result of sparse input features (exacerbated by the one-hot encoding required for nominal features) combined with the stopping criterion used. A poor feature subset selection consisting of irrelevant features and features with most or all values of 0, makes split optimization difficult. If the learned split is not useful, splitting is immediately stopped on the current branch, hence the trees become heavily



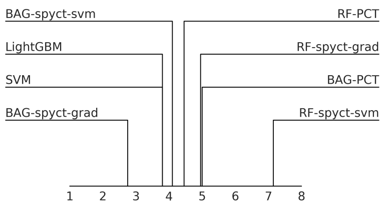
a) STR



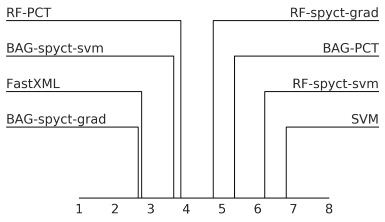
b) MTR



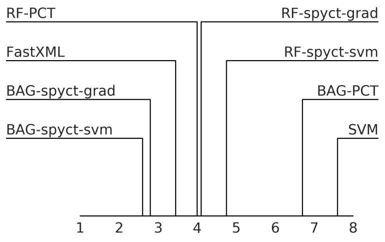
c) BIN



d) MCC



e) MLC



f) HMLC

Figure 2: Average ranks of predictive performances obtained by methods on single-target regression (a), multi-target regression (b), binary classification (c), multi-class classification (d), multi-label classification (e) and hierarchical multi-label classification (f) benchmark datasets.

over-pruned. It appears that random forest ensembles of oblique PCTs require a deeper redesign of the learning algorithm and straight-forward subspacing is not effective.

5.2. Learning time

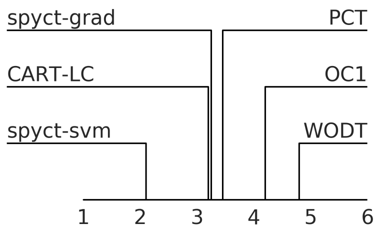
Experimental comparison of the learning times of different methods is a challenging task and difficult to do completely fairly. The implementations are in different programming languages (python, java, R, C++) and they are not equally optimized for efficient execution. Hence, we focus on the largest datasets, where time efficiency is more pronounced, and where the method's time complexity is more likely to dominate any implementation details.

Dataset	Task	BAG-SPYCT-SVM	BAG-SPYCT-GRAD	BAG-PCT	FASTXML	LIGHT-GBM
diabetes130us	MCC	95.7	65.4	749.0	NA	11.2
hepatitisC	MCC	5.4	1.8	12.4	NA	55.4
bookmarks	MLC	28.4	44.4	1747.7	16.8	NA
eurlex-eurovoc	MLC	200.8	8.0	DNF	8.1	NA
rcv1subset1	MLC	6.0	18.2	987.2	0.4	NA
tmc2007	MLC	22.5	129.2	DNF	4.8	NA
reuters	HMLC	4.0	11.8	883.2	0.4	NA
yeast_expr_FUN	HMLC	17.2	1.2	517.1	3.0	NA

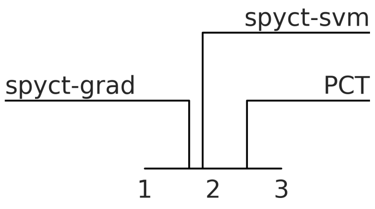
Table 8: Learning times of bagging ensembles on large datasets in minutes. NA means the method is not applicable to the task, DNF means the method did not finish in 3 days (4320 minutes). For dataset properties see Tables 5-7.

Table 8 presents the learning times on selected datasets with large numbers of examples, features, and/or targets. It is apparent, that both proposed variants SPYCT-SVM and SPYCT-GRAD are computationally much more efficient than standard PCTs. There are several reasons for the improvement of the learning time. To begin with, the main advantage comes from better scaling with the number of targets (as theoretically analyzed in Section 3.3), which is evident in the large differences on MLC and HMLC datasets. Next, another advantage is the exploitation of the sparse representation of the data. For example, if dense matrices are used instead of sparse matrices on the *reuters* dataset the learning time is dramatically increased: for BAG-SPYCT-SVM it increases from 4 minutes to 247.7 minutes, and for BAG-SPYCT-GRAD from 11.8 minutes to 54 minutes. Although learning from dense matrices is much slower than learning on sparse matrices, it is still faster than BAG-PCT. Furthermore, another source of learning time improvements is that with oblique splits, models can be much smaller (in terms of numbers of split nodes) compared to axis-parallel trees, because the splits are more expressive. An illustrative example of this is the *diabetes130us* dataset that does not even have multiple targets or sparse features: The BAG-SPYCT-SVM model consists of only 87 nodes and the BAG-SPYCT-GRAD of 2345 nodes, while the standard BAG-PCT model consists of 1803069 nodes. Lastly, we noticed faster learning on datasets with a large number of features, even when they are not sparse (e.g., the learning times of the *hepatitisC* dataset). We believe this is because the matrix operations are very well optimized in modern CPUs, which gives an advantage to the proposed optimization of oblique splits, compared to exhaustive search over the features in axis-parallel trees.

We can also note that LIGHTGBM is very efficient on datasets with many examples (*diabetes130us*), but less so on datasets with many features (*hepatitisC*). Furthermore, the implementation of the FASTXML method is extremely well optimized for (H)MLC datasets with a sparse structure. Therefore, even



a) BIN



b) MTR

Figure 3: Average ranks of predictive performances obtained by single-tree methods on binary classification and multi-target regression benchmark datasets.

though its theoretical computational complexity is very similar to our proposed methods, it has lower learning times on such datasets. However, when the features were not sparse, the observed learning times were in the same range as the proposed methods (e.g., for the *yeast_expr_FUN dataset*, FastXML has worse learning time than BAG-spyct-grad and better than BAG-spyct-svm).

5.3. Single tree methods

Even though we believe the oblique PCTs are mostly suited for use in ensembles, we briefly discuss our experiments for learning single trees. The results are illustrated in Figure 3 by presenting the average ranks on two tasks: BIN and MTR. Note that the same behavior can be observed for all the other tasks. As discussed in the introduction, most existing oblique trees are applicable only to binary classification problems. We see that our proposed methods achieve competitive performance also in the single tree setting. The results for the OC1 method on 3 datasets (*bioresponse*, *OVA_Breast* and *OVA_Lung*) are missing, because the algorithm exited without returning a tree when it was unable to find a split of the root node. It was assigned the worst ranking in those cases. We should note that even though WODT method performed poorly on BIN datasets, it was on par with other methods on MCC datasets. On SOP tasks, oblique PCTs generally outperformed standard PCTs, as illustrated in Figure 3b.

6. Feature importance scoring

We demonstrate the capability of extracting meaningful feature importance scores from the models using the approach described in Section 3.5. To evaluate the proposed approach, we added random noise features to the datasets and then learned BAG-spyct-svm and BAG-spyct-grad models on the expanded datasets. If the dataset originally had d features, we added another d features with random values, so the number of features has doubled. We took data

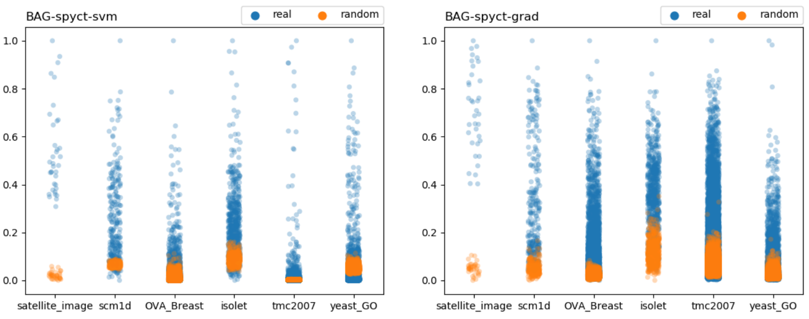


Figure 4: Feature importance scores obtained on different datasets by BAG-spyct-svm (left) and BAG-spyct-grad (right) methods. Results shows that the models do not rely on irrelevant features.

sparsity into account, so the ratio of non-zero values did not change (i.e., sparse datasets remained sparse).

Figure 4 presents the results on 6 datasets – we illustrate this on one dataset for each task. The importances of real features have diverse values which is expected: Datasets can contain features that are informative and features that are less informative or not informative at all. The main observation is that the added random features do not obtain high importance scores. This indicates that obtained importance scores are meaningful, and that methods are resilient to spurious features. This holds even on datasets with thousands of features and on datasets with sparse features.

7. Parameter sensitivity analysis

We performed parameter sensitivity analysis to investigate the influence of the different parameter values on the performance of the proposed methods. The analysis was performed on a different selection of datasets than the benchmarking experiments, to avoid any test set leakage. We used 3 datasets per task, yielding 18 datasets – their properties are presented in Table 9. For estimating the predictive performance, we used 5-fold cross validation.We experimented with different regularization strengths $C \in \{0.1, 1, 10, 100, 1000\}$ and different numbers of ensemble members (1, 10, 25, 50, 100). We also tried different thresholds for impurity reduction required to accept a split: $\{0\%, 5\%, 10\%, 20\%\}$. Note that at 0%, pre-pruning based on impurity reduction is turned off.

We present the most interesting observations in Figure 5. First, as expected, increasing the number of trees in the ensemble leads to better predictive performance. However, on several datasets (e.g., the *scm1d* dataset) the improvements become relatively small already from 10-25 trees onward. Second, the regularization strength C is difficult to get right. If the value is too low, the model does not fit the data enough. If it is too high, the model can overfit and the split weights are less sparse which can lead to large model sizes (memory-wise).

Dataset	Task	N	D	T	Sparse
cholesterol [22]	STR	303	13	1	
pol [22]	STR	15000	48	1	
qsar-30007 [22]	STR	534	1024	1	
andro [23]	MTR	49	30	6	
atp7d [23]	MTR	296	411	6	
wq [23]	MTR	1060	16	14	
arrhythmia [22]	BIN	452	279	1	
OVA_Endometrium [22]	BIN	1545	10935	1	
transfusion [22]	BIN	748	4	1	
energy_efficiency [22]	MCC	768	8	37	
gcm [22]	MCC	190	160063	14	
gesture [22]	MCC	9873	32	5	
birds [23]	MLC	645	260	19	T
delicious [23]	MLC	16105	500	983	D,T
mediamill [23]	MLC	43907	120	101	T
ara_scop_GO [25]	HMLC	9843	2003	572	T
imclef07a [24]	HMLC	11006	80	96	T
yeast_seq_FUN [25]	HMLC	3932	478	594	T

Table 9: Properties of datasets for parameter sensitivity study. Columns show predictive modelling task, number of examples (N), number of features (D), number of targets (T) and whether the input or output is sparse.

Gradient descent variant appeared to be more sensitive to this parameter than the SVM variant. The performance-wise results show that it is less damaging if C is too large, compared to if it is too small. Third, the impurity reduction threshold determines how good a split must be to be accepted (otherwise the current tree branch stops growing). Similar to the regularization strength, if the threshold is set too high, the model does not fit the data enough. If it is set too low, we risk overfitting the data (e.g., the *cholesterol* dataset) and/or inflating the model for no benefit (increases learning time and memory requirements, e.g., the *birds* dataset). Finally, our benchmarking experiments have shown that the selected default parameter values work well overall, however, they are unlikely to be optimal for any particular dataset. The best performance on a given dataset will be obtained by tuning the parameters for that dataset.

8. Conclusions

In this paper, we propose two methods for learning oblique predictive clustering trees. The nodes in oblique trees contain oblique splits that have linear combinations of features in the tests, hence the splits correspond to an arbitrary hyperplane in the input space. The first method starts by clustering the examples based on the target values and then learns a hyperplane in the input/feature space that approximates this clustering (the spyct-svm variant). The second method uses fuzzy membership indicators and weighted variance

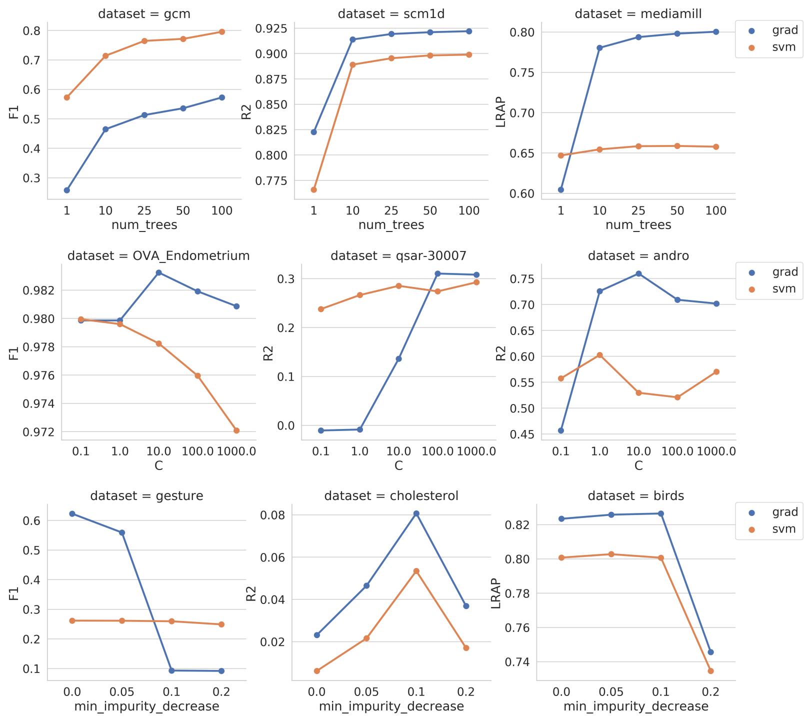


Figure 5: Predictive performance obtained with different numbers of ensemble members (top), regularization strengths (middle) and impurity reduction thresholds (bottom).

to directly optimize the hyperplane to minimize the impurity of examples on both sides (the spyct-grad variant). The proposed methods are designed to have drastically lower learning time on datasets with many targets compared to axis-parallel predictive clustering trees, while being capable of addressing many predictive modeling tasks, including structured output prediction. They can also exploit the sparsity present in the input/output data to more efficiently learn the predictive models.

We evaluated the proposed methods in a single tree and ensemble settings on 60 benchmark datasets for 6 predictive modeling tasks: binary classification, multi-class classification, multi-label classification, hierarchical multi-label classification, single-target regression, and multi-target regression. The results from the empirical study show that the predictive performance of ensembles of SPYCT-SVM and SPCYT-GRAD trees is on-par with state-of-the-art methods on all considered predictive modeling tasks, often exceeding it (especially SPYCT-GRAD). The learning times on datasets with high dimensional input and output spaces are orders of magnitude lower than learning times of standard PCTs, especially when the data is sparse. We also demonstrate the potential for extracting meaningful feature importance scores from the models, additionally indicating that the models are resilient to irrelevant features.

In future work, we plan to extend our work in several directions. To begin with, random forest ensembles of proposed oblique PCTs were not very successful. Hence, we plan to research the reasons behind these results and develop an improved version of the algorithm. Next, we will extend the developed methods towards the task of semi-supervised learning, where axis-parallel PCTs scale especially poorly in terms of computational cost due to a large number of clustering attributes considered there. Last but not least, we will evaluate the use of oblique trees for efficient learning of feature rankings in all of the above tasks as well as in the context of semi-supervised learning.

Acknowledgements

This work was supported by projects from the Slovenian Research Agency through the research program grant number P2-0103 (Young Researcher grant to TS) and the project grant number J2-9230.

References

[1] T. Stepišnik, D. Kocev, Multivariate predictive clustering trees for classification, in: D. Helic, G. Leitner, M. Stettinger, A. Felfernig, Z. W. Raś (Eds.), Foundations of Intelligent Systems, Springer International Publishing, Cham, 2020, pp. 331–341.

[2] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and Regression Trees, CRC press, 1984.

[3] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[4] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[5] M. Petković, D. Kocev, S. Džeroski, Feature ranking for multi-target regression, Machine Learning (Aug. 2019). `doi:10.1007/s10994-019-05829-8`.

[6] H. Blockeel, L. D. Raedt, J. Ramon, Top-Down Induction of Clustering Trees, in: Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98, Morgan Kaufmann Publishers Inc., 1998, pp. 55–63.

[7] H. Blockeel, M. Bruynooghe, S. Džeroski, J. Ramon, J. Struyf, Hierarchical multi-classification, in: Workshop Notes of the KDD'02 Workshop on Multi-Relational Data Mining, De Raedt, Luc, 2002, pp. 21–35.

[8] D. Kocev, C. Vens, J. Struyf, S. Džeroski, Tree ensembles for predicting structured outputs, Pattern Recognition 46 (3) (2013) 817–833. `doi:10.1016/j.patcog.2012.09.023`.

[9] J. Levatić, M. Ceci, D. Kocev, S. Džeroski, Semi-supervised classification trees, Journal of Intelligent Information Systems 49 (3) (2017) 461–486. doi:10.1007/s10844-017-0457-4.

[10] J. Levatić, D. Kocev, M. Ceci, S. Džeroski, Semi-supervised trees for multi-target regression, Information Sciences 450 (2018) 109–127. doi:10.1016/j.ins.2018.03.033.

[11] T. Stepišnik, D. Kocev, Hyperbolic embeddings for hierarchical multi-label classification, in: D. Helic, G. Leitner, M. Stettinger, A. Felfernig, Z. W. Raś (Eds.), Foundations of Intelligent Systems, Springer International Publishing, Cham, 2020, pp. 66–76.

[12] D. G. Heath, A geometric framework for machine learning, Ph.D. thesis, Johns Hopkins University, USA (1993).

[13] S. K. Murthy, S. Kasif, S. Salzberg, A system for induction of oblique decision trees, Journal of artificial intelligence research 2 (1994) 1–32.

[14] B.-B. Yang, S.-Q. Shen, W. Gao, Weighted Oblique Decision Trees, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 5621–5627.

[15] B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, F. A. Hamprecht, On oblique random forests, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2011, pp. 453–469.

[16] Y. Prabhu, M. Varma, FastXML: A Fast, Accurate and Stable Tree-Classifier for Extreme Multi-Label Learning, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, Association for Computing Machinery, 2014, pp. 263–272. doi:10.1145/2623330.2623651.

[17] H. Jain, Y. Prabhu, M. Varma, Extreme Multi-Label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, 2016, pp. 935–944. doi:10.1145/2939672.2939756.

[18] J. A. Hartigan, M. A. Wong, Algorithm AS 136: A K-Means Clustering Algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28 (1) (1979) 100–108. doi:10.2307/2346830.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, LIBLINEAR: A library for large linear classification, Journal of machine learning research 9 (Aug) (2008) 1871–1874.

[20] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

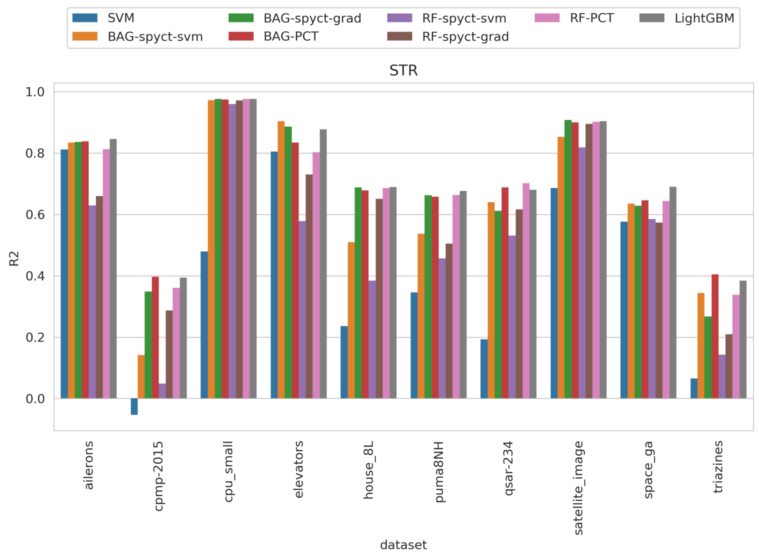


Figure 6: Predictive performance of ensemble methods and SVMs on regression datasets

[21] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.

[22] Openml, `https://www.openml.org`, accessed: 2020-04-15.

[23] Mulan: A java library for multi-label learning, `http://mulan.sourceforge.net/datasets.html`, accessed: 2020-04-15.

[24] `http://kt.ijs.si/DragiKocev/PhD/resources/doku.php?id=hmc_classification`, accessed: 2020-04-15.

[25] `https://dtai.cs.kuleuven.be/clus/hmc-ens/`, accessed: 2020-04-15.

Appendix

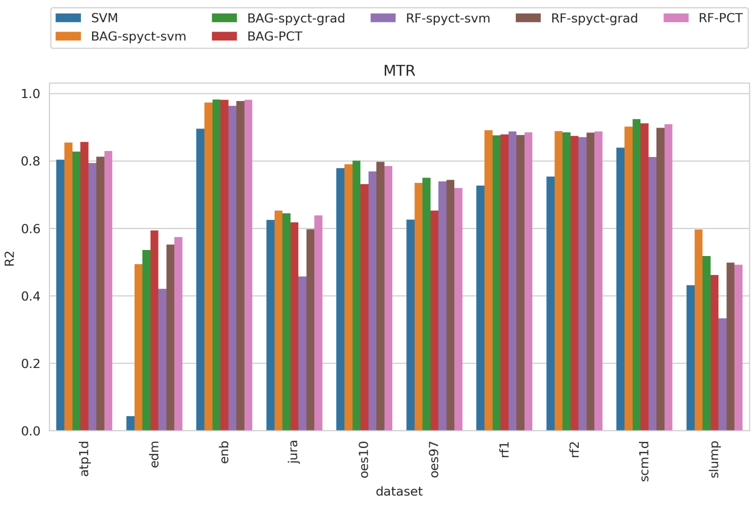


Figure 7: Predictive performance of ensemble methods and SVMs on multi-target regression datasets

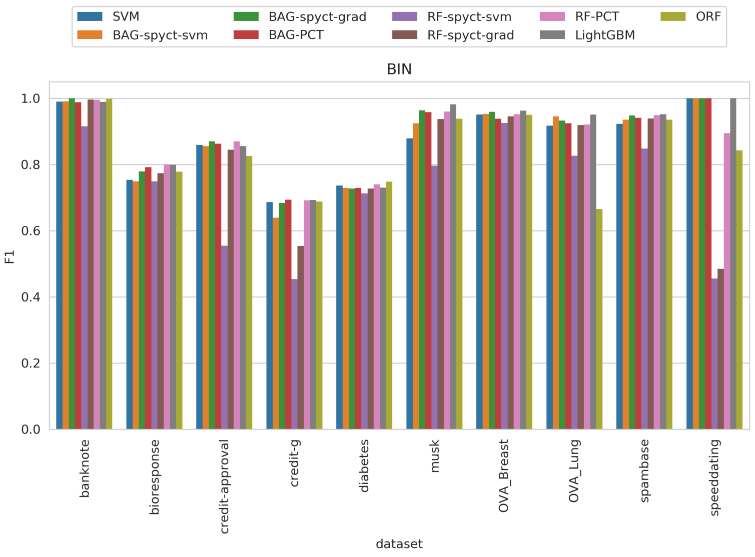


Figure 8: Predictive performance of ensemble methods and SVMs on binary classification datasets

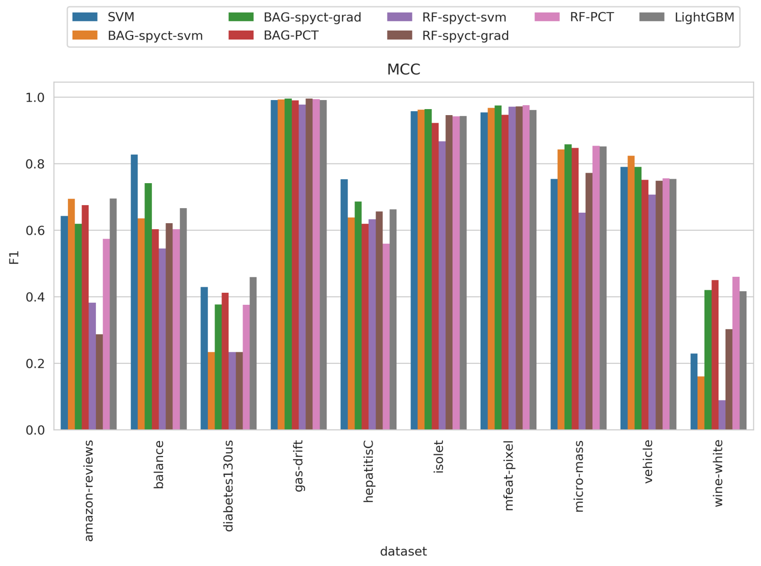


Figure 9: Predictive performance of ensemble methods and SVMs on multi-class classification datasets

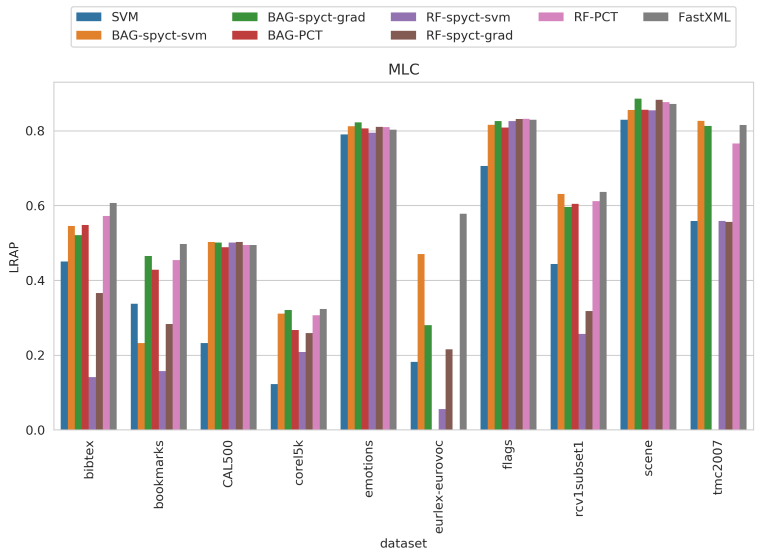


Figure 10: Predictive performance of ensemble methods and SVMs on multi-label classification datasets

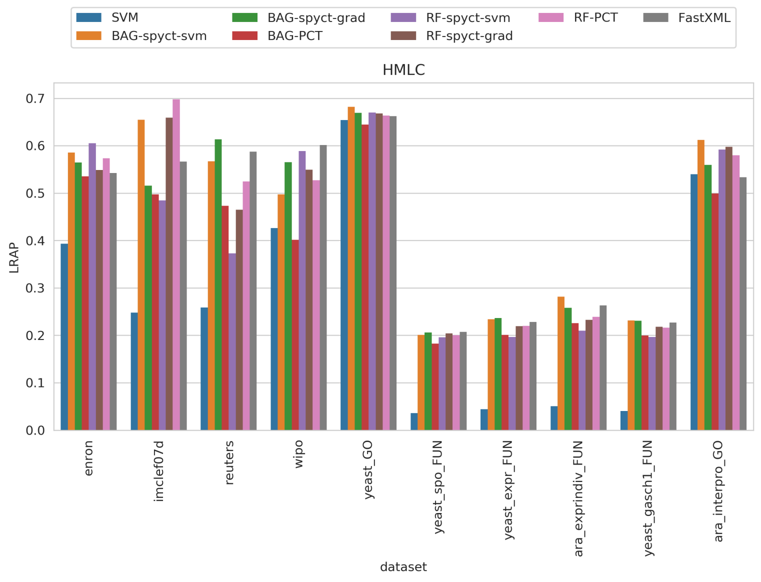


Figure 11: Predictive performance of ensemble methods and SVMs on hierarchical multi-label classification datasets

Chapter 8

Semi-Supervised Oblique Predictive Clustering Trees

In this chapter, we present the final set of contributions in this thesis – those related to semi-supervised learning of oblique predictive clustering trees. In the introductory sections, we described the principles of semi-supervised learning (Section 2.1.3) and provided a general overview of SPYCTs (Section 3.2) which included semi-supervised learning of SPYCTs (SSL-SPYCTs) as a special case. This section presents all the relevant details. Specifically, the contributions to this topic include:

1. Extension of the SVM variant of oblique predictive clustering trees towards supporting semi-supervised learning.

2. Extension of the gradient variant of oblique predictive clustering trees towards supporting semi-supervised learning.

3. Experimental evaluation of the proposed methods on benchmark datasets for 6 predictive modeling tasks: binary classification (BC), multi-class classification (MCC), multi-label classification (MLC), hierarchical multi-label classification (HMLC), single-target regression (STR), and multi-target regression (MTR).

The work on this topic was organized in a paper that was submitted to PeerJ Computer Science journal (Stepišnik & Kocev, 2020d) and is included in this chapter. It presents how both variants of SPYCTs can be learned in a semi-supervised manner and experimentally evaluates the proposed methods on 30 datasets for 6 predictive modeling tasks, both primitive and structured. We try different numbers of labeled examples for each dataset and focus on the comparison to supervised SPYCTs and semi-supervised standard PCTs (SSL-PCTs).

The main motivation for SSL-SPYCTs was the computational benefit they offer compared to SSL-PCTs, due to the better scaling with the number of clustering attributes, which in a semi-supervised setting include all the features. This advantage was confirmed with the experiments where SSL-SPYCTs had drastically lower learning times than SSL-PCTs. Additionally, we compared predictive performances of both variants of SSL-SPYCTs to their supervised counterparts and SSL-PCTs, both in single tree settings and in bagging ensembles. The results were again favorable. SSL-SPYCTs often achieved better predictive performance than both supervised SPYCTs and SSL-PCTs. The performance edge was often observed even in ensemble settings, where SSL-PCTs typically do not outperform supervised PCTs. Finally, we compared the feature importance scores obtained with SSL-SPYCTs and supervised SPYCTs and found that the semi-supervised approach can be more successful at identifying important features from few learning examples. This points to the potential for the development of a method for feature ranking in the context of semi-supervised learning.

From the hypotheses we defined in Section 1.2, this chapter addresses the following:

H5 Oblique predictive clustering trees are learned significantly faster than standard PCTs in a semi-supervised learning setting.

H3 Oblique predictive clustering trees reduce learning time on high dimensional data compared to standard PCTs without deteriorating predictive performance.

H6 Meaningful feature importances are calculated by using oblique PCTs.

The main focus in this chapter was on H5, which was confirmed with the theoretical analysis and experimental evaluation. Technically, we also addressed H3 and H6 and confirmed them for the semi-supervised context.

The paper included in this Chapter is:

- Stepišnik, T., Kocev, D. (2020d). Semi-supervised oblique predictive clustering trees. *PeerJ Computer Science [Under Review].*

The contributions of Tomaž Stepišnik to this paper are as follows. He designed and implemented the extensions of both SPYCT variants towards semi-supervised learning. He participated in the design of the study, carried out the experiments, and analyzed the results. He drafted the paper and revised it following the feedback from the co-authors and the reviewers.

Semi-supervised oblique predictive clustering trees

Tomaž Stepišnik[1,2] and Dragi Kocev[1,2,3]

[1]Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2]Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[3]Bias Variance Labs, d.o.o., Ljubljana, Slovenia

Corresponding author:
Tomaž Stepišnik[1]

Email address: tomaz.stepisnik@ijs.si

ABSTRACT

Semi-supervised learning combines supervised and unsupervised learning approaches to learn predictive models from both labeled and unlabeled data. It is most appropriate for problems where labeled examples are difficult to obtain but unlabeled examples are readily available (e.g., drug repurposing). Semi-supervised predictive clustering trees (SSL-PCTs) are a prominent method for semi-supervised learning that achieves good performance on various predictive modeling tasks, including structured output prediction tasks. The main issue, however, is that the learning time scales quadratically with the number of features. In contrast to axis-parallel trees, which only use individual features to split the data, oblique predictive clustering trees (SPYCTs) use linear combinations of features. This makes the splits more flexible and expressive and often leads to better predictive performance. With a carefully designed criterion function, we can use efficient optimization techniques to learn oblique splits. In this paper, we propose semi-supervised oblique predictive clustering trees (SSL-SPYCTs). We adjust the split learning to take unlabeled examples into account while remaining efficient. The main advantage over SSL-PCTs is that the proposed method scales linearly with the number of features. The experimental evaluation confirms the theoretical computational advantage and shows that SSL-SPYCTs often outperform SSL-PCTs and supervised PCTs both in single-tree setting and ensemble settings. We also show that SSL-SPYCTs are better at producing meaningful feature importance scores than supervised SPYCTs when the amount of labeled data is limited.

INTRODUCTION

The most common tasks in machine learning are supervised and unsupervised learning. In supervised learning, we are presented with a set of examples described with their properties (i.e., descriptive variables or features) as well as with a target property (i.e., output variables, target variables, or labels). The goal of a supervised learning method is to learn a mapping from the descriptive values to the output values that generalizes well to examples that were not used for learning. In unsupervised learning, on the other hand, no output values are provided for the examples. Instead, unsupervised methods aim to extract some underlying structure of the examples (e.g., discover clusters of similar examples, learn low dimensional representations, etc.).

Semi-supervised learning combines these two approaches (Chapelle et al., 2006). We are presented with a set of examples, where a (smaller) part of them are associated with output values (labeled examples), and a (larger) part of them are not (unlabeled examples). Semi-supervised methods learn a mapping from examples to the output values (like supervised methods), but also include unlabeled examples in the learning process (like unsupervised methods). The semi-supervised approach is typically used when learning examples are too scarce for supervised methods to learn a model that generalizes well, and, at the same time, unlabeled examples are relatively easy to obtain. This often happens in problems from life sciences, where the process of labeling the examples requires wet-lab experiments that are time-consuming and expensive. For example, consider the problem of discovering a new drug for a certain disease. Testing the effects of compounds on the progression of the disease requires screening

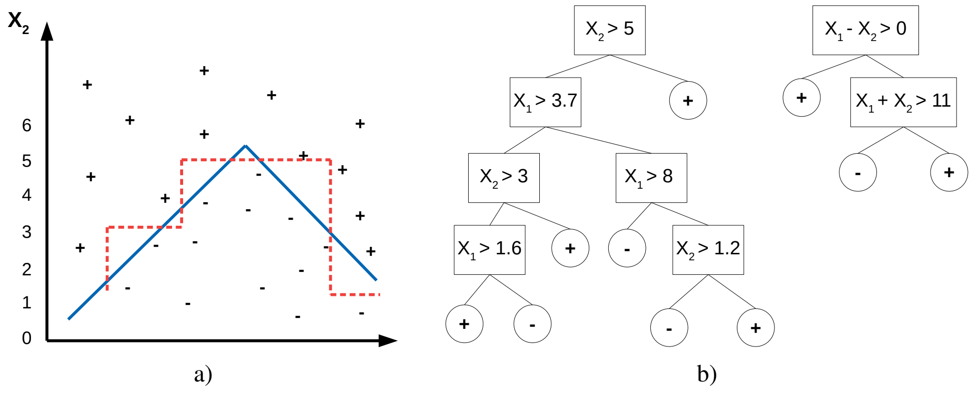


Figure 1. A toy dataset (a) with drawn decision boundaries learned by axis-parallel (red, dashed) and oblique (blue, solid) decision trees (b).

experiments, so labeling the examples (compounds) is expensive. On the other hand, millions of unlabeled compounds are present and described in online databases. Ideally, a semi-supervised method can use a handful of labeled compounds, combine them with the unlabeled compounds, and learn a model that can predict the effect of a compound on the disease progression, to facilitate the discovery of a novel drug.

The most common approaches to semi-supervised learning are wrapper methods (Van Engelen and Hoos, 2020), such as self-training (Kang et al., 2016), where a model iteratively labels the unlabeled examples and includes these pseudo-labels in the learning set in the next iteration. Alternatively, in co-training (Zhou and Li, 2007) there are two models that iteratively label the data for each other. Typically, these two models are different or at least learn on different views of the data. Among the intrinsically semi-supervised methods (Van Engelen and Hoos, 2020), semi-supervised predictive clustering trees (Levatić, 2017) are a prominent method. They can be used to solve a variety of predictive tasks, including multi-target regression and (hierarchical) multi-label classification (Levatić, 2017; Levatić et al., 2017, 2018; Levatić et al., 2020). They achieve good predictive performance and, as a bonus, the learned models can be interpreted, either by inspecting the learned trees or calculating feature importances from ensembles of trees (Petković et al., 2020). However, the method scales poorly with data dimensionality – the model learning can take a very long time on datasets with many features or targets.

Standard decision/regression trees (Breiman et al., 1984) split data based on the features in a way that minimizes the impurity of the target in the resulting clusters (e.g., variance for regression, entropy for classification). In the end nodes (leaves), predictions for the target are made. Predictive clustering trees (Blockeel et al., 1998, 2002) (PCTs) generalize standard trees by differentiating between three types of attributes: features, clustering attributes, and targets. Features are used to divide the examples; these are the attributes encountered in the split nodes. Clustering attributes are used to calculate the heuristic that guides the search of the best split at a given node, and targets are predicted in the leaves. The role of the targets in standard trees is therefore split between the clustering attributes and targets in PCTs. In theory, the clustering attributes can be selected independently of the features and the targets. However, the learned tree should make accurate predictions for the targets, so minimizing the impurity of the clustering attributes should help minimize the impurity of the targets. This attribute differentiation gives PCTs a lot of flexibility. They have been used for predicting various structured outputs (Kocev et al., 2013), including multi-target regression, multi-label classification, and hierarchical multi-label classification. Embeddings of the targets have been used as clustering attributes in order to reduce the time complexity of tree learning (Stepišnik and Kocev, 2020a). Semi-supervised PCTs use both targets and features as clustering attributes. This makes leaves homogeneous in both the input and the output space, which allows unlabeled examples to influence the learning process.

PCTs use individual features to split the data, which means the split hyperplanes in the input spaces are axis-parallel. SPYCTs (Stepišnik and Kocev, 2020b; Stepišnik and Kocev, 2020) are a redesign of standard PCTs and use linear combinations of features to achieve oblique splits of the data – the split hyperplanes are arbitrary. The potential advantage of oblique splits compared to axis-parallel splits is

Algorithm 1 Learning a SSL-SPYCT: The inputs are features $X_l \in \mathbb{R}^{L\times D}$ and $X_u \in \mathbb{R}^{U\times D}$ of labeled and unlabeled examples, targets $Y \in \mathbb{R}^{L\times T}$ of the labeled examples, and a vector $c \in \mathbb{R}^{D+T}$ of clustering weights.

```
1: procedure GROW_TREE($X_l$, $X_u$, $Y$, $c$)
2:     $w, b$ = get_split_hyperplane($X_l$, $X_u$, $Y$, $c$)
3:     score = $Xw + b$
4:     rows1 = $\{i \mid \text{score}_i > 0\}$
5:     rows2 = $\{i \mid \text{score}_i \leq 0\}$
6:     if acceptable_split(rows1, rows2) then
7:         left_subtree = grow_tree($X_l$[rows1], $X_u$[rows1], $Y$[rows1], $c$)
8:         right_subtree = grow_tree($X_l$[rows2], $X_u$[rows2], $Y$[rows2], $c$)
9:         return Node($w$, $b$, left_subtree, right_subtree)
10:    else
11:        return Leaf(prototype($Y$))
```

presented in Figure 1. SPYCTs offer state-of-the-art predictive performance, scale better with the number of clustering attributes, and can exploit sparse data to speed up computation.

In this paper, we propose SPYCTs for semi-supervised learning. We follow the same semi-supervised approach that regular PCTs do, which includes features in the heuristic function for evaluating the quality of a split. This makes the improved scaling of SPYCTs over PCTs especially beneficial, which is the main motivation for our proposal. We modify the oblique split learning objective functions of SPYCTs to account for missing target values. We evaluate the proposed approach on multiple benchmark datasets for different predictive modeling tasks.

In the remainder of the paper, we first describe the proposed semi-supervised methods and present the experimental setting for their evaluation. Next, we present and discuss the results of our experiments and, finally, conclude the paper by providing several take-home messages.

METHOD DESCRIPTION

In this section, we present our proposal for semi-supervised learning of SPYCTs (SSL-SPYCTs). We start by introducing the notation used in the manuscript. Let $X^l \in \mathbb{R}^{L\times D}$ and $X^u \in \mathbb{R}^{U\times D}$ be the matrices containing the D features of the L labeled and U unlabeled examples, respectively. Let $Y \in \mathbb{R}^{L\times T}$ be the matrix containing the T targets associated with the L labeled examples. And let $X = [(X^l)^T \ (X^u)^T]^T \in \mathbb{R}^{(L+U)\times D}$ be the matrix combining the features of both labeled and unlabeled examples. Finally, let $p \in \mathbb{R}^{D+T}$ be the vector of clustering weights, used to put different priorities to different clustering attributes (features and targets) when learning a split.

There are two variants of SPYCTs that learn the split hyperplanes in different ways.

1. The *SVM variant* first groups the examples into two clusters based on the clustering attributes using k-means clustering, then learns a linear SVM on the features with cluster indicators as targets to approximate this split.

2. The *gradient variant* uses a fuzzy membership indicator to define a differentiable objective function which measures the impurity on both sides of the split hyperplane. The hyperplane is then optimized with gradient descent to minimize the impurity.

The basis of the semi-supervised approach is to use both features and targets as clustering attributes, so that unlabeled examples influence the learning process through the heuristic score calculation, despite the missing target values. For the SVM variant, this means that examples are clustered based on both target and feature values. For the gradient variant, the split is optimized to minimize the impurity of both features and targets on each side of the hyperplane. The overview of the SSL-SPYCT learning algorithm is presented in Algorithm 1. The weights $w \in \mathbb{R}^D$ and bias $b \in \mathbb{R}$ define the split hyperplane, and they are obtained differently for each SPYCT variant as follows.

SVM variant The first step is to cluster examples into two groups using k-means clustering. The initial centroids are selected randomly from the labeled examples. Since the clustering is performed based on both features and targets, the cluster centroids consist of feature and target parts, i.e.,

$$c^0 = \begin{bmatrix} X^l_{i,:} & Y_{i,:} \end{bmatrix} \in \mathbb{R}^{D+T}, \; c^1 = \begin{bmatrix} X^l_{j,:} & Y_{j,:} \end{bmatrix} \in \mathbb{R}^{D+T}.$$

Next, we calculate the Euclidean distance to the two centroids for each of the examples. For unlabeled examples, we only calculate the distance to the feature part of the centroids (c^x_0 and c^x_1):

$$d(i,j) = \sum_{k=1}^{D} p_k (X_{j,k} - c^i_k)^2 + \alpha \sum_{k=1}^{T} p_{D+k} (Y_{j,k} - c^i_{D+k})^2,$$

where $i \in \{0,1\}$ is the cluster indicator, $1 \leq j \leq L+U$ is the example index, and $\alpha = 1$ if the example is labeled (i.e., $j \leq L$) and $\alpha = 0$ if it is unlabeled. The examples are split into two clusters according to the closer centroid. In the case of ties in the distance, the examples are assigned (uniformly) randomly to a cluster.

Let $s \in \{0,1\}^{L+U}$ be the vector indicating the cluster membership. The new centroids are then the means of the examples assigned to each cluster. The means of the target parts of the centroids are calculated only using the labeled examples, i.e.,

$$c^i_j = \frac{\sum_{k=1}^{L+U} \mathbb{1}[s_k = i] X_{k,j}}{\sum_{k=1}^{L+U} \mathbb{1}[s_k = i]}, \qquad \text{if } 1 \leq j \leq D,$$

$$c^i_j = \frac{\sum_{k=1}^{L} \mathbb{1}[s_k = i] Y_{k,j-D}}{\sum_{k=1}^{L} \mathbb{1}[s_k = i]}, \qquad \text{if } D < j \leq D+T.$$

This procedure is repeated for a specified number of iterations. After the final clusters are determined, a linear SVM is used to approximate this split based on the features. Specifically, the following optimization problem is solved:

$$\min_{w,b} \; ||w||_1 + C \sum_{k=1}^{L+U} max(0, 1 - s_k(X_{k,:} \cdot w + b))^2,$$

where parameter $C \in \mathbb{R}$ determines the strength of regularization.

Gradient variant We start with randomly initialized weights (w) and bias (b) and calculate the fuzzy membership vector $s = \sigma(Xw + b) \in [0,1]^{L+U}$. The value s_i tells us how much the corresponding example belongs to the "positive" group, whereas the value $1 - s_i$ tells us how much it belongs to the "negative" group. To calculate the impurity of a group, we calculate the weighted variance for every feature and every target. For the targets, only labeled examples are used in the calculation. Weighted variance of a vector $v \in \mathbb{R}^n$ with weights $a \in \mathbb{R}^n$ is defined as

$$\mathtt{var}(v,a) = \frac{\sum_i^n a_i (v_i - \mathtt{mean}(v,a))^2}{A} = \mathtt{mean}(v^2,a) - \mathtt{mean}(v,a)^2,$$

where $A = \sum_i^n a_i$ is the sum of weights and $\mathtt{mean}(v,a) = \frac{1}{A} \sum_i^n a_i v_i$ is the weighted mean of v. The impurity of the positive group is then calculated as

$$\mathtt{imp}(s,p) = \sum_{k=1}^{D} p_k \mathtt{var}(X_{:,k}, s) + \sum_{k=1}^{T} p_{D+k} \mathtt{var}(Y_{:,k}, s).$$

To get the impurity of the negative group $\mathtt{imp}(1-s,p)$, we simply swap the fuzzy membership weights with $1-s$. The split fitness function we wish to optimize is then

$$f(w,b) = S \cdot \mathtt{imp}(s,p) + (L + U - S) \cdot \mathtt{imp}(1-s,p),$$

where $s = \sigma(Xw + b)$ and $S = \sum_i s_i$. The terms S and $L+U-S$ represent the sizes of the positive and negative subsets and are added to guide the split search towards balanced splits. The final optimization problem for learning the split hyperplane is

$$\min_{w,b} \; ||w||_{\frac{1}{2}} + Cf(w,b),$$

where C again controls the strength of regularization. The objective function is differentiable, and we can efficiently solve the problem using the Adam(Kingma and Ba, 2014) gradient descent optimization method.

The clustering weights are uniform for the targets for tasks of binary classification, multi-class classification, multi-label classification, regression, and multi-target regression. For hierarchical multi-label classification, the weights for target labels positioned lower in the hierarchy are smaller. This gives more importance to labels higher in the hierarchy when splitting the examples.

We also implement a parameter ω that determines the degree of supervision. The clustering weights, corresponding to features (p_i for $1 \leq i \leq D$), are scaled so that their sum is $1 - w$, and clustering weights, corresponding to targets (p_i for $D < i \leq D + T$), are scaled so that their sum is ω. This enables us to determine the relative importance of features and targets when splitting the data. With the borderline values selected for ω (0 or 1), we get the extreme behavior in terms of the amount of supervision. Setting the value of ω to 0 means that the target impurity is ignored and tree construction is effectively unsupervised, i.e., without supervision. Alternatively, setting the value of ω to 1 means that feature impurity is ignored when learning splits, hence, the unlabeled examples do not affect the split selection. The tree construction in this case is fully supervised.

The splitting of the examples (i.e., the tree construction) stops when at least one of the following stopping criteria is reached. We can specify the minimum number of examples required in leaf nodes (at least one labeled example is always required otherwise predictions cannot be made). We can also require a split to reduce the impurity by a specified amount or specify the maximum depth of the tree.

After the splitting stops, a leaf node is created. The prototype of the targets of the remaining examples is calculated and it is stored for use as the prediction for the examples reaching that leaf. Since the targets in SOP are represented as tuples/vectors, the prototypes are calculated as column-wise mean values of the targets (Y). They can be used directly as predictions (in regression problems) or used to calculate the majority class (in binary and multi-class classification), or used to predict all labels with the mean above a certain threshold (in hierarchical and flat multi-label classification).

The time complexity of learning a split in standard PCTs is $O(DN \log N + NDK)$ (Kocev et al., 2013), where K is the number of clustering attributes. For the SVM and gradient variant of SPYCTs, the time complexities are $O(N(I_c K + I_o D))$ and $O(N I_o (D + K))$, respectively (Stepišnik and Kocev, 2020), where I_o is the number of w, b optimization iterations and I_c is the number of clustering iterations (SVM variant). When learning SSL variants (SSL-PCTs and SSL-SPYCTs), clustering attributes consist of both features and targets, therefore $K = D + T$. This means that SSL-PCTs scale quadratically with the number of features, whereas both variants of SSL-SPYCTs scale linearly. SSL-SPYCTs are therefore much more computationally efficient, and can additionally take advantage of sparse data by performing calculations with sparse matrices. Our implementation of the proposed method is freely licensed and available for use and download at `https://gitlab.com/TStepi/spyct`.

EXPERIMENTAL DESIGN

We evaluated our approach on multiple benchmark dataset for different predictive modeling tasks: binary classification (BC), multiclass classification (MCC), multi-label classification (MLC), and hierarchical multi-label classification (HMLC), single-target regression (STR) and multi-target regression (MTR). The datasets are freely available and were obtained from the following repositories: `openml`[1], `mulan`[2], `dtai-cs`[3] and `kt-ijs`[4]. The selected datasets have diverse properties in terms of application domains, number of examples, number of features, and number of targets. Their properties and sources are presented in Table 1.

We focus on the comparison of our proposed SSL-SPYCT method with the original supervised method SPYCT and the semi-supervised learning of axis-parallel PCTs: the SSL-PCT Levatić (2017). These two baselines are the most related supervised and semi-supervised methods of the proposed approach, respectively. For completeness, we also include supervised PCTs in the comparison. Note that SPYCTs and PCTs are the only available methods able to address all of the structured output prediction tasks in a uniform manner. We evaluate the methods in single tree setting and in bagging ensembles Breiman (1996) of 50 trees.

We follow the evaluation setup used for the evaluation of SSL-PCTs Levatić et al. (2020). For semi-supervised methods, we select the ω parameter with 3-fold internal cross-validation on the training

[1] `https://www.openml.org`
[2] `http://mulan.sourceforge.net/datasets.html`
[3] `https://dtai.cs.kuleuven.be/clus/hmc-ens/`
[4] `http://kt.ijs.si/DragiKocev/PhD/resources/doku.php?id=hmc_classification`

Table 1. Details of the benchmark datasets used for the evaluation. The task column shows the predictive modeling task applicable to the datasets (BC is binary classification, MCC is multi-class classification, MLC is multi-label classification, HMLC is hierarchical multi-label classification, STR is single-target regression, MTR is multi-target regression), N is the number of examples, D is the number of features, and T is the number of targets (for MCC, it is the number of classes).

dataset	source	task	N	D	T
bioresponse	openml	BC	3751	1776	1
mushroom	openml	BC	8124	22	1
phoneme	openml	BC	5404	5	1
spambase	openml	BC	4601	57	1
speeddating	openml	BC	8378	120	1
cardiotocography	openml	MCC	2126	35	10
gesture	openml	MCC	9873	32	5
isolet	openml	MCC	7797	617	26
mfeat-pixel	openml	MCC	2000	240	10
plants-texture	openml	MCC	1599	64	100
bibtex	mulan	MLC	7395	1836	159
birds	mulan	MLC	645	260	19
bookmarks	mulan	MLC	87856	2150	208
delicious	mulan	MLC	16105	500	983
scene	mulan	MLC	2407	294	6
ara_interpro_GO	dtai-cs	HMLC	11763	2815	630
diatoms	kt-ijs	HMLC	3119	371	398
enron	kt-ijs	HMLC	1648	1001	56
imclef07d	kt-ijs	HMLC	11006	80	46
yeast_seq_FUN	dtai-cs	HMLC	3932	478	594
cpmp-2015	openml	STR	2108	23	1
pol	openml	STR	15000	48	1
qsar-197	openml	STR	1243	1024	1
qsar-122261	openml	STR	1842	1024	1
satellite_image	openml	STR	6435	36	1
atp1d	mulan	MTR	337	411	6
enb	mulan	MTR	768	8	2
oes97	mulan	MTR	334	263	16
rf2	mulan	MTR	9125	576	8
scm1d	mulan	MTR	9803	280	16

set. We select the best value from the set $\{0, 0.25, 0.5, 0.75, 1\}$. We investigate the influence of the number of labeled examples L on the performance of the semi-supervised methods. We set L to the following numbers of available labeled examples: $\{25, 50, 100, 250, 500\}$. For each L, we randomly sample L examples from the dataset as labeled examples. The remaining examples are used both as unlabeled examples (by discarding their labels) and as a testing set. This follows the transductive learning setting Malerba et al. (2009), where the learning algorithms see the features of the testing examples, but not their targets. The supervised algorithms do not use unlabeled examples, so they are only learned on the L labeled examples, and evaluate on the same test set. We repeat this procedure 10 times for each selection of L. On the two MTR datasets that have fewer than 500 examples (*atp1d* and *oes97*) experiments with $L = 500$ are not performed.

To measure the predictive performance of the methods on STR and MTR datasets, we use the *coefficient of determination*

$$R^2(y, \hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2},$$

where y is the vector of true target values, $\bar{y}$ is their mean, and $\hat{y}$ is the vector of predicted values. For MTR problems, we calculate the mean of R^2 scores per target. For BIN and MCC tasks, we use *F1 score*,

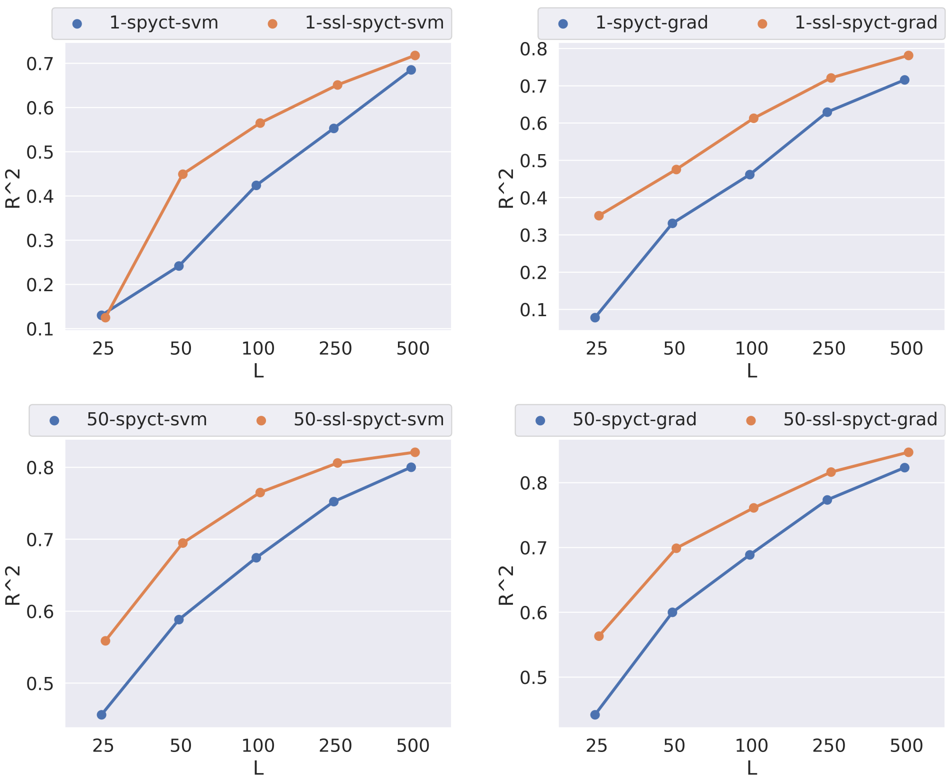


Figure 2. Comparison of supervised and semi-supervised variants of SPYCT-SVM and SPYCT-GRAD methods (columns) in both single tree and ensemble settings (rows) on the *rf2* dataset with different numbers of labeled examples (L).

macro averaged in the MCC case.

Methods solving MLC and HMLC tasks typically return a score for each label and each example, higher score meaning that example is more likely to have that label. Let $y \in \{0,1\}^{n\times l}$ be the matrix of label indicators and $\hat{y} \in \mathscr{R}^{n\times l}$ the matrix of label scores returned by a method. We measured the performance of methods with weighted label ranking average precision

$$LRAP(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j:y_{ij}=1}\frac{w_j}{W_i}\frac{L_{ij}}{R_{ij}},$$

where $L_{ij} = |\{k : y_{ik} = 1 \wedge \hat{y}_{ik} \geq \hat{y}_{ij}\}|$ is the number of real labels assigned to example i that the method ranked higher than label j, $R_{ij} = |\{k : \hat{y}_{ik} \geq \hat{y}_{ij}\}|$ is the number of all labels ranked higher than label j, w_j is the weight we put to label j and W_i is the sum of weights of all labels assigned to example i. For the MLC datasets, we put equal weights to all labels, whereas, for the HMLC datasets, we weighted each label with 0.75^d, with d being the depth of the label in the hierarchy Kocev et al. (2013). The same weights are also used as the clustering weights for the targets for all methods.

RESULTS AND DISCUSSION

Predictive performance comparison

We first present the results obtained on the *rf2* dataset in Figure 2. Here, the semi-supervised approach outperforms supervised learning for both SPYCT variants, in both single-tree and ensemble settings, and all considered numbers of labeled examples. The only exception is for a single SPYCT-SVM tree with 25 labeled examples, where the semi-supervised variant selected $\omega = 1$ and the performance is therefore equivalent to the supervised variant. These results demonstrate the potential of the proposed SSL methods.

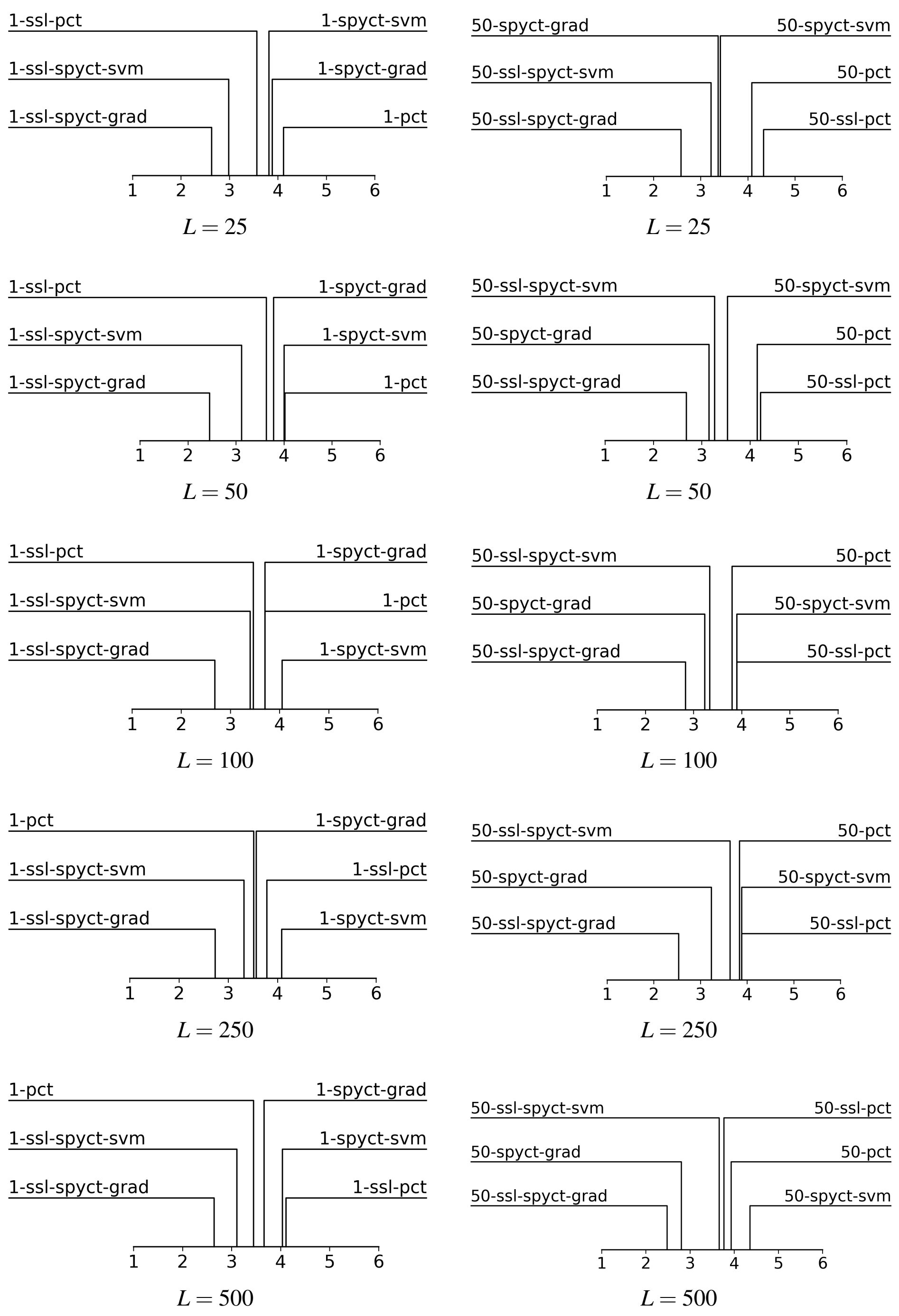


Figure 3. Average ranking diagrams comparing the predictive performance of the proposed SPYCT-SVM-SSL and SPYCT-GRAD-SSL methods and the baselines with different numbers of labeled examples.

For a high-level comparison of the predictive performance of the proposed SSL methods and the baselines, we use average ranking diagrams Demsar (2006). The results are presented in Figure 3. The

Table 2. Comparison of the proposed SSL-SPYCT methods to their supervised counterparts and SSL-PCTs in single tree setting in terms of number of wins. Bolded values indicate that the sign test Demsar (2006) showed significant difference in performance at $\alpha = 0.05$.

#wins	$L = 25$	$L = 50$	$L = 100$	$L = 250$	$L = 500$
1-SPYCT-GRAD-SSL vs. 1-SPYCT-GRAD	**23**	**25**	**24**	20	**22**
1-SPYCT-SVM-SSL vs. 1-SPYCT-SVM	18	19	20	19	**20**
1-SPYCT-GRAD-SSL vs. 1-PCT-SSL	19	**21**	18	**21**	**21**
1-SPYCT-SVM-SSL vs. 1-PCT-SSL	18	17	15	18	**20**

Table 3. Comparison of the proposed SSL-SPYCT methods to their supervised counterparts and SSL-PCTs in ensemble setting in terms of number of wins. Bolded values indicate that the sign test Demsar (2006) showed significant difference in performance at $\alpha = 0.05$.

#wins	$L = 25$	$L = 50$	$L = 100$	$L = 250$	$L = 500$
50-SSL-SPYCT-GRAD vs. 50-SPYCT-GRAD	20	17	17	17	14
50-SSL-SPYCT-SVM vs. 50-SPYCT-SVM	15	**21**	17	15	16
50-SSL-SPYCT-GRAD vs. 50-SSL-PCT	**24**	**24**	20	**21**	**20**
50-SSL-SPYCT-SVM vs. 50-SSL-PCT	19	**21**	18	17	17

first observation is that SSL-SPYCT-GRAD achieves the best rank for all numbers of labeled examples in both single tree and ensemble settings. Additionally, SSL-SPYCT-SVM also ranks better than both its supervised variant and SSL-PCT for all values of L and both single tree and ensemble settings. For standard PCTs, the semi-supervised version performed better than the supervised version in a single tree setting with very few labeled examples ($L = 25, 50, 100$), otherwise, their performances were very similar. This is consistent with the previous studies Levatić et al. (2017, 2018); Levatić et al. (2020).

Next, we dig deeper into the comparison of SSL-SPYCT variants to the supervised SPYCTs and SSL-PCTs. We performed pairwise comparisons among the competing pairs with sign tests Demsar (2006) on the number of wins. Table 2 and Table 3 present the results for single tree and ensemble settings, respectively. The results show that in a single tree setting, SSL-SPYCT-GRAD outperforms its supervised version for all different numbers of labeled examples. The difference is statistically significant, the only exception is for $L = 250$ where it falls slightly short. On the other hand, while SSL-SPYCT-SVM also outperforms its supervised version more often than not, the difference was only statistically significant for $L = 500$. Both SSL-SPYCT-GRAD and SSL-SPYCT-SVM compare favorably to SSL-PCTs, winning on more than half datasets for any value of L. Statistically significant difference is more often achieved by SSL-SPYCT-GRAD variant. Another important observation is that supervised variants never significantly outperform the SSL variants. This confirms that dynamically selecting the ω parameter prevents scenarios where unlabeled examples are detrimental to the predictive performance and supervised learning works better. We can also see that SSL variants outperform supervised counterparts more often in single tree settings compared to ensembles. This is again consistent with the studies of SSL-PCTs Levatić et al. (2017, 2018); Levatić et al. (2020).

Learning time comparison

To compare the learning times of the proposed SSL methods and SSL-PCTs, we selected one large dataset for each predictive task. We focused on the large datasets where the differences highlight the scalability of the methods with respect to the numbers of features and targets. We compare learning times of tree ensembles, as they also serve as a (more reliable) comparison for learning times of single trees. Figure 4 shows the learning times on the selected datasets. The results confirm our theoretical analysis

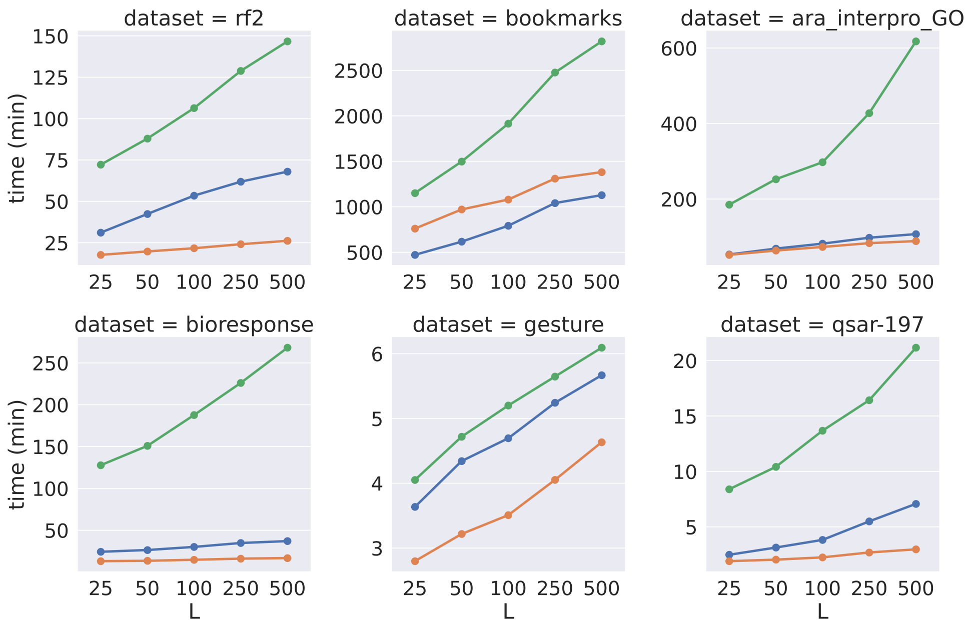


Figure 4. Comparison of learning times of the SSL algorithms on a selection of large benchmark datasets.

and show that the proposed SSL-SPYCTs are learned significantly faster than SSL-PCTs. The differences are especially large on datasets with many features and/or targets (e.g., *ara_interpro_GO*). The learning times are most similar on the *gesture* dataset, which has only 32 features, so the theoretical advantage of SSL-SPYCTs is less accentuated. Notwithstanding, the proposed methods are faster also on this dataset.

Investigating the ω parameter

The ω parameter controls the amount of influence of the unlabeled examples on the learning process. Figure 5 shows the distributions of the ω values selected with the internal 3-fold cross-validation. We can see that the selected values varied greatly, sometimes different values were chosen even for different folds of the same dataset. This confirms the need to determine ω with internal cross-validation for each dataset separately. Additionally, we notice that larger of ω values tend to be selected with more labeled examples and by ensembles compared to single trees. With larger numbers of labeled examples, it makes sense that the model can rely more heavily on the labeled part of the data and unlabeled examples are not as beneficial. For ensembles, this indicates that they can extract more useful information from few labeled examples compared to single trees. Whereas this seems clear for larger datasets, bootstrapping on few examples is not obviously beneficial. The fact that ensembles tend to select larger ω values also explains why the differences in predictive performance between supervised and semi-supervised variants are smaller in ensembles compared to single trees. We also investigated whether the selected ω values were influenced by the predictive modeling task (regression vs. classification, single target vs. multiple targets), but we found no noticeable differences between the ω distributions.

Investigating feature importances

We can extract feature importance scores from learned SPYCT trees Stepišnik and Kocev (2020). The importances are calculated based on absolute values of weights assigned to individual features in all the split nodes in a tree (or ensemble of trees). These scores tell us how much the model relies on individual features and can also be used to identify important features for a given task. We investigated if SSL-SPYCTs are more successful at identifying important features compared to supervised SPYCTs in problems with limited labeled data. To do this, we followed the setup from Stepišnik and Kocev (2020) and added random features (noise) to the datasets. For each original feature, we added a random one so that the total number of features was doubled. Then we learned SPYCTs and SSL-SPYCTs and compared

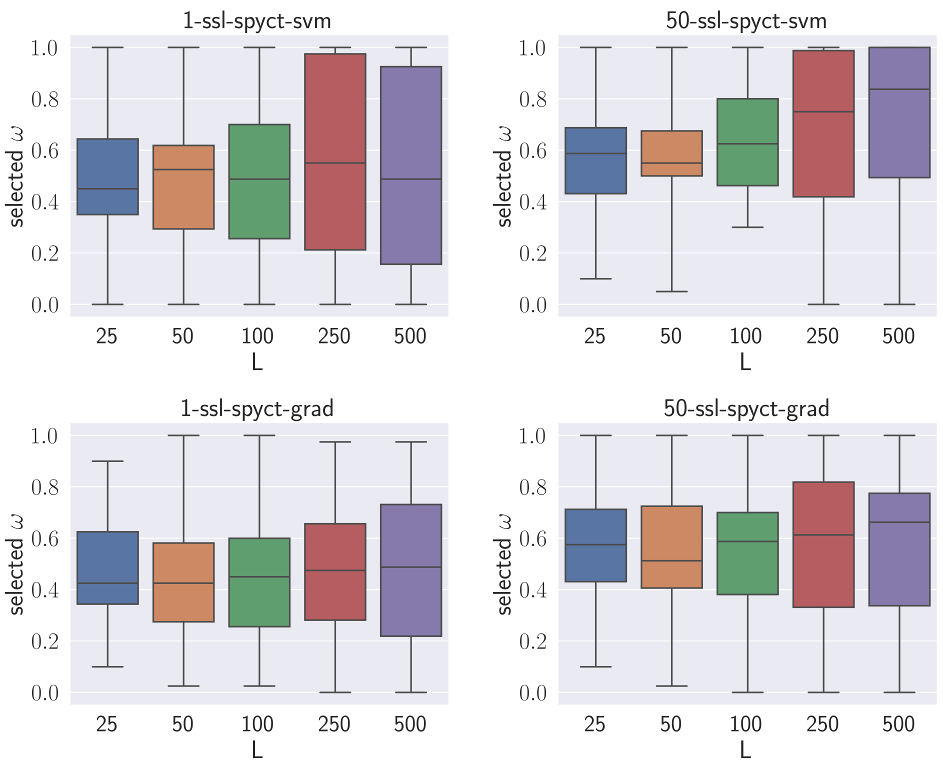


Figure 5. Distributions of the selected ω parameters for different number of labeled examples.

the extracted feature importances.

Figure 6 presents the results on the qsar-197 dataset. For convenience, we also show the predictive performances of SPYCT and SSL-SPYCT methods. A good feature importance scoring would put the scores of random features (orange) to zero, whereas some real features (blue) would have noticeably higher scores. Low scores of many real features are not concerning, as datasets often include features that are not very useful for predicting the target. This example shows that SSL-SPYCTs can be better at identifying useful features than supervised SPYCTs. The difference here is greater with the gradient variant, especially with 50-250 labeled examples. This is also reflected in the predictive performance of the methods.

In general, the quality of feature importance scores obtained from a model was correlated with the model's predictive performance. This is expected and means that the conclusions here are similar. In terms of feature importance scores, SSL-SPYCTs are often similar to supervised SPYCTs, but there are several examples (e.g., Figure 6) where they are significantly better and worth the extra effort.

CONCLUSION

In this paper, we propose semi-supervised learning of oblique predictive clustering trees. We follow the approach of standard semi-supervised predictive clustering trees and adapt both SVM and gradient variants of SPYCTs and make them capable of learning from unlabeled examples. The main motivation for the proposed methods was the improved computational scaling of SPYCTs compared to PCTs which is highlighted in the proposed SSL approach, where features are also taken into account when evaluating the splits.

We experimentally evaluated the proposed methods on 30 benchmark datasets for various predictive modeling tasks in both single tree and ensemble settings. The experiments confirmed the substantial theoretical computational advantage the proposed SSL-SPYCT methods have over standard SSL-PCTs. The results also showed that the proposed methods often achieve better predictive performance than

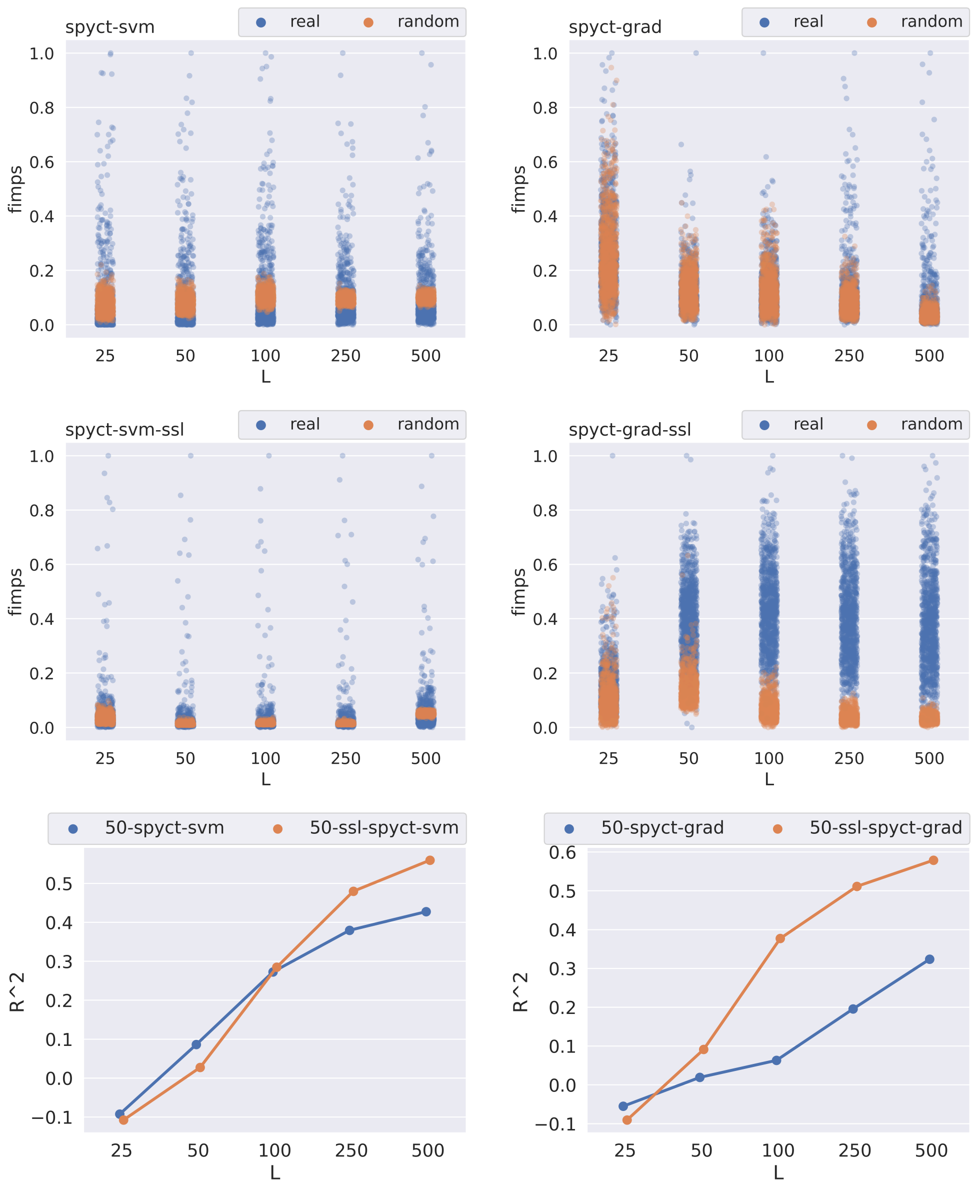


Figure 6. Comparison of feature importance scores of real and random features (scaled to $[0,1]$ interval) for the qsar-197 dataset with different numbers of labeled examples. The first column shows the SPYCT-SVM method, the second column the SPYCT-GRAD method. The first row shows importance scores obtained with the supervised method, the second row shows the importance scores obtained with the unsupervised method, and the third row presents the predictive performance of both supervised and semi-supervised methods.

both supervised SPYCTs and SSL-PCTs. The performance edge was often preserved even in ensemble settings, where SSL-PCTs typically did not outperform supervised PCTs. Finally, we demonstrated that SSL-SPYCTs can be significantly better at obtaining meaningful feature importance scores.

The main drawback of SSL-SPYCTs (which is shared with SSL-PCTs) is the requirement to determine the ω parameter dynamically with internal cross-validation. This increases the learning time compared to supervised learning but prevents occasions where introducing unlabeled examples into the learning process hurts the predictive performance. We investigated the selected values for ω and found that higher values tend to be selected when there is more labeled data available, and by ensembles compared to single trees. But the selected values were still very varied, which confirms the need for dynamic selection of ω.

For future work, we plan to investigate SPYCTs in boosting ensembles for both supervised and semi-supervised learning. Variants of gradient boosting Friedman (2001) have proven especially successful in many applications recently. We will also try improving the interpretability of the learned models with Shapley additive explanations (SHAP Lundberg et al. (2020)). Because our method is tree-based we might be able to calculate the Shapley values efficiently, similarly to how they are calculated for axis-parallel tree methods.

REFERENCES

Blockeel, H., Bruynooghe, M., Džeroski, S., Ramon, J., and Struyf, J. (2002). Hierarchical multi-classification. In *Workshop Notes of the KDD'02 Workshop on Multi-Relational Data Mining*, pages 21–35. De Raedt, Luc.

Blockeel, H., Raedt, L. D., and Ramon, J. (1998). Top-Down Induction of Clustering Trees. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 55–63. Morgan Kaufmann Publishers Inc.

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

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. CRC press.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-supervised Learning*. MIT Press.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232.

Kang, P., Kim, D., and Cho, S. (2016). Semi-supervised support vector regression based on self-training with label uncertainty: An application to virtual metrology in semiconductor manufacturing. *Expert Systems with Applications*, 51:85–106.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kocev, D., Vens, C., Struyf, J., and Džeroski, S. (2013). Tree ensembles for predicting structured outputs. *Pattern Recognition*, 46(3):817–833.

Levatić, J. (2017). *Semi-Supervised Learning for Structured Output Prediction*. PhD thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia.

Levatić, J., Ceci, M., Kocev, D., and Džeroski, S. (2017). Semi-supervised classification trees. *Journal of Intelligent Information Systems*, 49(3):461–486.

Levatić, J., Kocev, D., Ceci, M., and Džeroski, S. (2018). Semi-supervised trees for multi-target regression. *Information Sciences*, 450:109–127.

Levatić, J., Ceci, M., Stepišnik, T., Džeroski, S., and Kocev, D. (2020). Semi-supervised regression trees with application to qsar modelling. *Expert Systems with Applications*, 158:113569.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67.

Malerba, D., Ceci, M., and Appice, A. (2009). A relational approach to probabilistic classification in a transductive setting. *Engineering Applications of Artificial Intelligence*, 22(1):109–116.

Petković, M., Džeroski, S., and Kocev, D. (2020). Feature ranking for semi-supervised learning.

Stepišnik, T. and Kocev, D. (2020a). Hyperbolic embeddings for hierarchical multi-label classification. In Helic, D., Leitner, G., Stettinger, M., Felfernig, A., and Raś, Z. W., editors, *Foundations of Intelligent Systems*, pages 66–76, Cham. Springer International Publishing.

Stepišnik, T. and Kocev, D. (2020b). Multivariate predictive clustering trees for classification. In Helic, D., Leitner, G., Stettinger, M., Felfernig, A., and Raś, Z. W., editors, *Foundations of Intelligent Systems*, pages 331–341, Cham. Springer International Publishing.

Stepišnik, T. and Kocev, D. (2020). Oblique predictive clustering trees.

Van Engelen, J. E. and Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440.

Zhou, Z.-H. and Li, M. (2007). Semi-supervised regression with co-training style algorithms. *IEEE Transaction on Knowledge and Data Engineering*, 19(11):1479–1493.

Chapter 9

Conclusions and Further Work

In this thesis, we presented two approaches to learning predictive clustering trees with complex nodes. PCTs are well-established machine learning models that can be used for a variety of tasks and achieve great predictive performance in ensembles. However, like standard decision trees, they are induced with a greedy top-down induction algorithm, which makes myopic decisions that can lead to sub-optimal trees. Additionally, while this algorithm is efficient for primitive data, it scales poorly with the number of clustering attributes. This becomes a noticeable problem with multiple outputs encountered in structured output prediction problems and, especially, in the semi-supervised setting where clustering attributes include both targets and features. We proposed option nodes and oblique split nodes to tackle these weaknesses while retaining the flexibility of original PCTs in terms of machine learning tasks they can address, i.e., primitive and structured outputs in supervised and semi-supervised settings.

In the remainder of this chapter, we first summarize the scientific contributions of the thesis. We then provide directions for further work, including approaches to address the limitations of the proposed methods.

9.1 Summary of the Contributions

A method for learning option predictive clustering trees that are capable of addressing structured output prediction (SOP) tasks

Option decision trees were proposed to reduce the myopia of the tree-learning algorithm by taking into account multiple alternative splits when the best split is not clearly distinguishable (i.e., when multiple splits achieve heuristic scores close to the maximum). We extended PCTs with option nodes to create option predictive clustering trees (OPCTs). Compared to standard decision trees, PCTs modify the split evaluation and the prototypes in the leaves. However, the split evaluation is still performed with a heuristic scoring function that returns a real value, and the prototypes are not involved in option nodes. Therefore, we were able to naturally extend PCTs with option nodes using the same mechanisms that were proposed for option decision trees while keeping the flexibility of standard PCTs intact.

An empirical evaluation of option PCTs on benchmark datasets for SOP and their comparison to standard PCTs and ensembles thereof

We evaluated the proposed OPCTs for multi-target regression, multi-label classification, and hierarchical multi-label classification tasks. We investigated how the number of option nodes introduced in a tree affects the trade-off between the predictive performance and the interpretability (size) of the learned models. We found that introducing many option nodes into OPCTs makes them similar to bagging ensembles of PCTs. They achieve great predictive performance, but models are large and take a relatively long time to learn. On the other hand, by strictly limiting the number of option nodes, OPCTs remain more similar to single PCTs. Like ensembles with very few trees, predictive performance typically improves quickly even with just a few option nodes. However, OPCTs offer better interpretability compared to ensembles of similar sizes because the different options still share common ancestor splits, whereas the trees in an ensemble are completely independent.

SVM-based and gradient-descent-based methods for efficient learning of oblique PCTs capable of addressing SOP tasks and handling high dimensional and sparse data.

To improve the computational scaling of PCTs, we proposed oblique predictive clustering trees (SPYCTs). They replace axis-parallel split nodes with oblique split nodes where tests use linear combinations of features. Because it is computationally infeasible to iterate over all possible oblique splits and select the one with the highest heuristic score, we needed to thoroughly redesign the split selection part of the learning algorithm. We proposed two methods for optimizing split hyperplanes. The SVM variant first finds an ideal split based on the clustering attributes and then optimizes a hyperplane in the input space that approximates it. The gradient variant uses fuzzy membership indicators to define a differentiable approximation of the heuristic score used by standard PCTs. The hyperplane is then optimized to minimize it using gradient-based methods.

While the learning time of standard PCTs scales with the product of the numbers of features and clustering attributes, the learning times for both proposed variants for learning SPYCTs scale with their sum. Additionally, both variants can exploit sparsity in the data to further reduce the computational costs by skipping the operations where 0-valued features/targets are involved.

An empirical evaluation of oblique PCTs and ensembles thereof on benchmark datasets for standard classification and regression tasks as well as for SOP tasks

We evaluated the proposed methods for learning SPYCTs on benchmark datasets for MTR, MLC, and HMLC tasks. Because the approach is also novel for primitive outputs, we evaluated the methods on binary classification, multi-class classification, and single-target regression problems as well. The experiments confirmed the significant theoretical computational advantage of SPYCTs over standard PCTs. Furthermore, the predictive performance of SPYCTs, especially the gradient variant, was at least on par with other similar and state-of-the-art methods. Most notably, they compare favorably to other oblique tree methods and standard PCTs, both in ensembles and single-tree settings. The computational gains were achieved without sacrificing performance. Finally, we demonstrated the possibility of extracting meaningful feature importances from the learned SPYCTs and ensembles thereof.

An empirical comparison of oblique PCTs and standard PCTs on semi-supervised learning tasks

The scaling problem of PCTs is most noticeable when they are learned in the semi-supervised setting. For this reason, we also compared semi-supervised SPYCTs and semi-supervised PCTs. We used the same 6 predictive modeling tasks as for the supervised setting for the comparison. The experiments again confirmed the computational advantage of SPYCTs. Additionally, SSL-SPYCTs often achieved better predictive performance than both SSL-PCTs and supervised SPYCTs. This was the case in single-tree settings as well as in ensembles, where SSL-PCTs typically do not outperform supervised PCTs. We also showed that SSL-SPYCTs can be better than supervised SPYCTs at extracting feature importances in problems with few learning examples.

9.2 Further Work

The presented work addressed the problems presented in the introduction, and the proposed methods were evaluated in a wide variety of settings. However, there are still some avenues left to explore and new directions that opened up as a result of our research.

First, while OPCTs address the myopia of the learning algorithm, they still exhibit the scaling problems of original PCTs. One obvious idea is to combine option nodes and oblique split nodes. Unfortunately, the proposed methods for learning oblique splits optimize a single split, and looking for alternative splits to create an option node would come with a great computational cost. Alternative splits can be obtained by learning multiple splits with the proposed methods and different initializations, or by replacing gradient-based optimization with population-based optimization methods (e.g., genetic algorithms). On the other hand, one can use feature sampling in the manner of random forests to only evaluate tests for a subset of features at each node and improve the scaling with the number of features. It would be interesting to see how such a method would perform and how the number of option nodes would be affected by the subsampling.

On a related note, our first attempt at feature sampling in SPYCTs was not completely successful because trees sometimes stopped growing prematurely. Multiple options can be explored to fix this behavior. First, using encodings of nominal features that are less sparse than the one-hot encoding we used can help reduce the probability of selecting a bad subset of features. Second, a different strategy for selecting the feature subset might be more appropriate than a uniformly random approach. A smart sampling strategy could help us avoid particularly bad feature subsets. Finally, we could give the algorithm multiple chances to learn a split. If the process was unsuccessful on the first subset of features, we try again on a different sample.

In terms of the interpretability of OPCTs, we mostly focused on manual inspection of the learned trees. The feasibility of such an approach depends mostly on the size of a tree, which is why we investigated the numbers of option nodes introduced with different parameters. On the other hand, we could also use OPCTs to calculate feature importances, similarly to how they are calculated from tree ensembles (Petković et al., 2020). While less informative, the feature importances still provide insight into which features the model relies upon. This approach for extracting feature importances from OPCTs, and the method for extracting feature importances from SPYCTs that we proposed, can also be used to do feature ranking. We can then explore how the feature rankings obtained from OPCTs and SPYCTs compare to feature rankings obtained from PCT ensembles.

While feature rankings and importances tell us which features a model relies upon and how much, they do not tell us in what way they influence the predictions. For example, when predicting if a patient has breast cancer, the sex of the patient would have a very high feature importance score. However, this would not tell us whether the model predicts higher probabilities of breast cancer for men or for women. For standard trees and ensembles thereof, the TreeSHAP method (Lundberg et al., 2020) can be used to obtain this level of insight into the model. We plan on investigating how this approach can be used with OPCTs and SPYCTs to improve their interpretability as well.

References

Anwar, S. M., Majid, M., Qayyum, A., Awais, M., Alnowami, M., & Khan, M. K. (2018). Medical image analysis using convolutional neural networks: A review. *Journal of medical systems*, *42*(11), 226.

Araújo, T., Aresta, G., Castro, E., Rouco, J., Aguiar, P., Eloy, C., Polónia, A., & Campilho, A. (2017). Classification of breast cancer histology images using convolutional neural networks. *PloS one*, *12*(6), e0177544.

Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixão, T. M., Mutz, F., de Paula Veronese, L., Oliveira-Santos, T., & De Souza, A. F. (2021). Self-driving cars: A survey. *Expert Systems with Applications*, *165*, 113816.

Bakır, G., Hofmann, T., Schölkopf, B., Smola, A. J., & Taskar, B. (2007). *Predicting structured data*. MIT press.

Bennett, K. P., & Demiriz, A. (1998). Semi-supervised support vector machines. *Proceedings of the 11th International Conference on Neural Information Processing Systems*, 368–374.

Blockeel, H., Bruynooghe, M., Džeroski, S., Ramon, J., & Struyf, J. (2002). Hierarchical multi-classification. *Workshop Notes of the KDD'02 Workshop on Multi-Relational Data Mining*, 21–35.

Blockeel, H., Raedt, L. D., & Ramon, J. (1998). Top-Down Induction of Clustering Trees. *Proceedings of the Fifteenth International Conference on Machine Learning*, 55–63.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bratko, I. (2001). *Prolog programming for artificial intelligence*. Addison Wesley. https://books.google.si/books?id=-15su78YRj8C

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

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

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Buntine, W. (1992). Learning classification trees. *Statistics and computing*, *2*(2), 63–73.

Chen, Z.-M., Wei, X.-S., Wang, P., & Guo, Y. (2019). Multi-label image recognition with graph convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5177–5186.

Consortium, G. O. (2019). The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, *47*(D1), D330–D338.

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

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, *7*(Jan), 1–30.

Dimitrovski, I., Kocev, D., Loskovska, S., & Dzeroski, S. (2012). Hierarchical classification of diatom images using ensembles of predictive clustering trees. *Ecological Informatics*, *7*(1), 19–29.

Dimitrovski, I., Kocev, D., Loskovska, S., & Džeroski, S. (2016). Improving bag-of-visual-words image retrieval with predictive clustering trees. *Information Sciences*, *329*, 851–865.

Elisseeff, A., & Weston, J. (2002). A kernel method for multi-labelled classification. *Advances in neural information processing systems*, 681–687.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, *9*(Aug), 1871–1874.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, *4*(1), 1–58.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *28*(1), 100–108.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Heath, D. G. (1993). *A geometric framework for machine learning* (Doctoral dissertation). Johns Hopkins University. USA.

Hodos, R. A., Kidd, B. A., Shameer, K., Readhead, B. P., & Dudley, J. T. (2016). In silico methods for drug repurposing and pharmacology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, *8*(3), 186–210.

International Organization for Standardization. (2007). *ISO/IEC 11404:2007 Information technology – General-Purpose Datatypes (GPD)* (tech. rep.). https://www.iso.org/standard/39479.html

Jain, H., Prabhu, Y., & Varma, M. (2016). Extreme Multi-Label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 935–944.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kocev, D., Ceci, M., & Stepišnik, T. (2020). Ensembles of extremely randomized predictive clustering trees for predicting structured outputs. *Machine Learning*, *109*, 2213–2241.

Kocev, D., Vens, C., Struyf, J., & Džeroski, S. (2013). Tree ensembles for predicting structured outputs. *Pattern Recognition*, *46*(3), 817–833.

Kohavi, R., & Kunz, C. (1997). Option decision trees with majority votes. *ICML*, *97*, 161–169.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Langley, P., & Morgan, M. (1996). *Elements of machine learning*. Morgan Kaufmann. https://books.google.si/books?id=TNg5qVoqRtUC

Latitude. (2020). *AI dungeon*. Retrieved November 2, 2020, from https://play.aidungeon.io

Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, *5*(1), 15–17.

Levatić, J., Ceci, M., Kocev, D., & Džeroski, S. (2017). Semi-supervised classification trees. *Journal of Intelligent Information Systems*, *49*(3), 461–486.

Levatić, J., Ceci, M., Stepišnik, T., Džeroski, S., & Kocev, D. (2020). Semi-supervised regression trees with application to qsar modelling. *Expert Systems with Applications*, 113569.

Levatić, J., Kocev, D., Ceci, M., & Džeroski, S. (2018). Semi-supervised trees for multi-target regression. *Information Sciencies*, *450*(100), 109–127.

Liu, P., Qiu, X., & Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101.*

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, *2*(1), 2522–5839.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems 30* (pp. 4765–4774). Curran Associates, Inc.

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. *Proceedings of the European conference on computer vision (ECCV)*, 116–131.

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, *45*, 3084–3104.

Menze, B. H., Kelm, B. M., Splitthoff, D. N., Koethe, U., & Hamprecht, F. A. (2011). On oblique random forests. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 453–469.

Minguet, F., Salgado, T. M., Santopadre, C., & Fernandez-Llimos, F. (2017). Redefining the pharmacology and pharmacy subject category in the journal citation reports using medical subject headings (mesh). *International Journal of Clinical Pharmacy*, *39*(5), 989–997.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of artificial intelligence research*, *2*, 1–32.

Osojnik, A., Džeroski, S., & Kocev, D. (2016). Option predictive clustering trees for multi-target regression. *International Conference on Discovery Science*, 118–133.

Panov, P., Soldatova, L., & Džeroski, S. (2016). Generic ontology of datatypes. *Information Sciences*, *329*, 900–920.

Panov, P., Soldatova, L. N., & Džeroski, S. (2016). Generic ontology of datatypes [Special issue on Discovery Science]. *Information Sciences*, *329*, 900–920. https://doi.org/https://doi.org/10.1016/j.ins.2015.08.006

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Petković, M., Džeroski, S., & Kocev, D. (2019a). Ensemble-based feature ranking for semi-supervised classification. *International Conference on Discovery Science*, 290–305.

Petković, M., Kocev, D., & Džeroski, S. (2020). Feature ranking for multi-target regression. *Machine Learning*, *109*(6), 1179–1204.

Petković, M., Lucas, L., Kocev, D., Džeroski, S., Boumghar, R., & Simidjievski, N. (2019b). Quantifying the effects of gyroless flying of the mars express spacecraft with machine learning. *Proceedings of the 2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 9–16.

Prabhu, Y., & Varma, M. (2014). FastXML: A Fast, Accurate and Stable Tree-Classifier for Extreme Multi-Label Learning. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 263–272.

Rosenberg, C., Hebert, M., & Schneiderman, H. (2005). Semi-supervised self-training of object detection models. *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*, *1*, 29–36.

Rousu, J., Saunders, C., Szedmak, S., & Shawe-Taylor, J. (2006). Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, *7*(Jul), 1601–1626.

Silla, C. N., & Freitas, A. A. (2011). A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, *22*(1-2), 31–72.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Sliwoski, G., Kothiwale, S., Meiler, J., & Lowe, E. W. (2014). Computational methods in drug discovery. *Pharmacological reviews*, *66*(1), 334–395.

Stepišnik, T., & Kocev, D. (2020a). Hyperbolic embeddings for hierarchical multi-label classification. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, & Z. W. Raś (Eds.), *Foundations of intelligent systems* (pp. 66–76). Springer International Publishing.

Stepišnik, T., & Kocev, D. (2020b). Multivariate predictive clustering trees for classification. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, & Z. W. Raś (Eds.), *Foundations of intelligent systems* (pp. 331–341). Springer International Publishing.

Stepišnik, T., & Kocev, D. (2020c). Oblique predictive clustering trees. *Knowledge-Based Systems [Under review]*.

Stepišnik, T., & Kocev, D. (2020d). Semi-supervised oblique predictive clustering trees. *PeerJ Computer Science [Under review]*.

Stepišnik, T., Kocev, D., & Džeroski, S. (2020). Option predictive clustering trees for multi-label classification. *Acta Polytechnica Hungarica – Journal of Applied Sciences*, *17*(10), 109–128.

Stepišnik, T., Osojnik, A., Džeroski, S., & Kocev, D. (2020). Option predictive clustering trees for multi-target regression. *Computer Science and Information Systems*, *17*(2), 459–486.

Stepišnik Perdih, T., Kocev, D., & Džeroski, S. (2018). Option predictive clustering trees for multi-label classification. *ETAI: Proc. Abstracts of ETAI–2018, XIV Intl. Conf. on Electronics, Telecommunications, Automatics and Informatics.*

Stepišnik Perdih, T., Osojnik, A., Džeroski, S., & Kocev, D. (2017). Option predictive clustering trees for hierarchical multi-label classification. In A. Yamamoto, T. Kida, T. Uno, & T. Kuboyama (Eds.), *Discovery science* (pp. 116–123). Springer International Publishing.

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

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2009). Mining multi-label data. *Data mining and knowledge discovery handbook* (pp. 667–685). Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998–6008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., & Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine learning*, *73*(2), 185.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Wehrmann, J., Cerri, R., & Barros, R. (2018). Hierarchical multi-label classification networks. *International Conference on Machine Learning*, 5075–5084.

Winter, R., Montanari, F., Noé, F., & Clevert, D.-A. (2019). Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. *Chemical science*, *10*(6), 1692–1701.

Witten, I. H., & Frank, E. (2002). Data mining: Practical machine learning tools and techniques with java implementations. *SIGMOD Rec.*, *31*(1), 76–77. https://doi.org/10.1145/507338.507355

Yang, B.-B., Shen, S.-Q., & Gao, W. (2019). Weighted Oblique Decision Trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 5621–5627.

Zhou, Z.-H., & Li, M. (2007). Semi-supervised regression with co-training style algorithms. *IEEE Transaction on Knowledge and Data Engineering*, *19*(11), 1479–1493.

Bibliography

Publications Included in the Thesis

Journal Articles

Stepišnik, T., & Kocev, D. (2020c). Oblique predictive clustering trees. *Knowledge-Based Systems [Under review].*

Stepišnik, T., & Kocev, D. (2020d). Semi-supervised oblique predictive clustering trees. *PeerJ Computer Science [Under review].*

Stepišnik, T., Kocev, D., & Džeroski, S. (2020). Option predictive clustering trees for multi-label classification. *Acta Polytechnica Hungarica – Journal of Applied Sciences*, *17*(10), 109–128.

Stepišnik, T., Osojnik, A., Džeroski, S., & Kocev, D. (2020). Option predictive clustering trees for multi-target regression. *Computer Science and Information Systems*, *17*(2), 459–486.

Conference Papers

Stepišnik Perdih, T., Osojnik, A., Džeroski, S., & Kocev, D. (2017). Option predictive clustering trees for hierarchical multi-label classification. In A. Yamamoto, T. Kida, T. Uno, & T. Kuboyama (Eds.), *Discovery science* (pp. 116–123). Springer International Publishing.

Publications Related to the Thesis

Conference Papers

Stepišnik, T., & Kocev, D. (2020b). Multivariate predictive clustering trees for classification. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, & Z. W. Raś (Eds.), *Foundations of intelligent systems* (pp. 331–341). Springer International Publishing.

Stepišnik Perdih, T., Kocev, D., & Džeroski, S. (2018). Option predictive clustering trees for multi-label classification. *ETAI: Proc. Abstracts of ETAI–2018, XIV Intl. Conf. on Electronics, Telecommunications, Automatics and Informatics.*

Other Publications

Journal Articles

Kocev, D., Ceci, M., & Stepišnik, T. (2020). Ensembles of extremely randomized predictive clustering trees for predicting structured outputs. *Machine Learning*, *109*, 2213–2241.

Levatić, J., Ceci, M., Stepišnik, T., Džeroski, S., & Kocev, D. (2020). Semi-supervised regression trees with application to qsar modelling. *Expert Systems with Applications*, 113569.

Ljoncheva, M., Stepišnik, T., Džeroski, S., & Kosjek, T. (2020). Cheminformatics in ms-based environmental exposomics: Current achievements and future directions. *Trends in Environmental Analytical Chemistry*, e00099.

Conference Papers

Levatić, J., Brbić, M., Perdih, T. S., Kocev, D., Vidulin, V., Šmuc, T., Supek, F., & Džeroski, S. (2018a). Phenotype prediction with semi-supervised classification trees. In A. Appice, C. Loglisci, G. Manco, E. Masciari, & Z. W. Ras (Eds.), *New frontiers in mining complex patterns* (pp. 138–150). Springer International Publishing.

Stepišnik, T., & Kocev, D. (2020a). Hyperbolic embeddings for hierarchical multi-label classification. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, & Z. W. Raś (Eds.), *Foundations of intelligent systems* (pp. 66–76). Springer International Publishing.

Biography

Tomaž Stepišnik was born on 18 April 1991 in Postojna, Slovenia. He finished primary school in Šmarje pri Jelšah and secondary school in Celje. In 2010, he started his bachelor's studies of Mathematics at the Faculty of Mathematics and Physics, University of Ljubljana. He graduated in 2013 (GPA 9.81) and started his master's studies at the same faculty. He finished the master's programme in 2016 (GPA 9.69) with the defence of the master's thesis entitled Distances on Structured Data (slo. Razdalje na strukturiranih podatkih) under the mentorship of Prof. Dr. Sašo Džeroski. In 2016, he received a Young Researcher grant from the Slovenian Research Agency and started working as a young researcher at the Department of Knowledge Technologies, Jožef Stefan Institute. He enrolled in the PhD study program Information and Communication Technologies at the Jožef Stefan International Postgraduate School.

His research interest is in the area of machine learning, focusing on structured output prediction and tree-based methods as well as their applications in a variety of domains such as medicine, chemoinformatics and life sciences. He has published multiple scientific papers and presented his work at several international conferences and workshops. During his studies he took part in several projects, including two funded by the European Union (HBP – The Human Brain project, MAESTRA – Learning from Massive, Incompletely annotated, and Structured Data) and two funded by the European Space Agency (GalaxAI – Machine Learning for Space Operations, AiTLAS – Artificial Intelligence Toolbox for Earth Observation). In 2017/18, he was a teaching assistant at the Faculty of Mathematics and Physics. He also received a Recognition for best study grades (slo. Svečana listina za najboljši študijski uspeh) in 2014 and the golden award at the Mathematical Kangaroo competition for students in 2019.