Machine Learning Methods for Fuzzy Pattern Tree Induction

Robin Senge

aus Bad Brückenau

Fachbereich Mathematik und Informatik

Philipps-Universität Marburg

Dissertation

zur Erlangung eines

Doktorgrades der Naturwissenschaften (Dr. rer. nat.)

Marburg/Lahn, 2014

Vom Fachbereich Mathematik und Informatik der Philipps-Universiät Marburg
(Hochschulkennziffer: 1180) als Dissertation angenommen am: 15.04.2014

Erstgutachter: Prof. Dr. Eyke Hüllermeier
Zweitgutachter: Prof. Dr. Christophe Marsala

weitere Mitglieder der Prüfungskommission:
Prof. Dr. Manfred Sommer
Prof. Dr. Bernhard Seeger

Tag der mündlichen Prüfung: 20.06.2014

Abstract

This thesis elaborates on a novel approach to fuzzy machine learning, that is, the combination of machine learning methods with mathematical tools for modeling and information processing based on fuzzy logic. More specifically, the thesis is devoted to so-called fuzzy pattern trees, a model class that has recently been introduced for representing dependencies between input and output variables in supervised learning tasks, such as classification and regression. Due to its hierarchical, modular structure and the use of different types of (nonlinear) aggregation operators, a fuzzy pattern tree has the ability to represent such dependencies in a very flexible and compact way, thereby offering a reasonable balance between accuracy and model transparency.

The focus of the thesis is on novel algorithms for pattern tree induction, i.e., for learning fuzzy pattern trees from observed data. In total, three new algorithms are introduced and compared to an existing method for the data-driven construction of pattern trees. While the first two algorithms are mainly geared toward an improvement of predictive accuracy, the last one focuses on efficiency aspects and seeks to make the learning process faster. The description and discussion of each algorithm is complemented with theoretical analyses and empirical studies in order to show the effectiveness of the proposed solutions.

Zusammenfassung

Diese Arbeit befasst sich mit einer neuen Methode im Bereich des "unscharfen maschinellen Lernens". Dieser Bereich umfasst Methoden, die eine Kombination von maschinellen Lernverfahren mit Konzepten aus der

Theorie unscharfer Mengen (Fuzzy Set Theory) darstellen. Im Speziellen beschäftigt sich die Arbeit mit der Modellklasse der sogenannten Fuzzy Pattern Trees. Diese Modellklasse wurde kürzlich vorgestellt und dient der Modellierung von Abhängigkeiten zwischen Ein- und Ausgabevariablen in Problemstellungen des überwachten Lernens (supervised learning) wie beispielsweise Regression und Klassifikation. Aufgrund ihrer hierarchischen und modularen Struktur sowie der Verwendung von unterschiedlichen, zum Teil nicht-linearen Aggregationsoperatoren ist die Modellklasse in der Lage, diese Abhängigkeiten flexibel und kompakt darzustellen. Dabei erreicht sie eine gute Balance zwischen prädiktiver Genauigkeit und Transparenz.

Schwerpunkt dieser Arbeit ist die Entwicklung neuer Algorithmen zum Lernen von Fuzzy Pattern Trees aus Daten. Insgesamt werden in dieser Arbeit neben einer Reihe von Heuristiken drei neue Algorithmen vorgestellt. Während die ersten beiden nahezu ausschließlich auf die Verbesserung der Prädiktionsgenauigkeit abzielen, wird mit dem dritten die Verbesserung der Laufzeit während der Trainingsphase erreicht. Die Beschreibung der Algorithmen ist immer sowohl von einer theoretischen Diskussion als auch einer empirischen Evaluierung begleitet, die die Effektivität und Effizienz der Algorithmen demonstriert und statistisch belegt.

To myself.

Acknowledgements

During my PhD study, I was supported by the help of many people. Most importantly, I would like to thank my advisor Prof. Eyke Hüllermeier. His advice and support were always very helpful and directed me to the right path for my studies. In particular, the discussions with him and his whole research group of Computational Intelligence in the Department of Mathematics and Computer Science at the University of Marburg, were always challenging and productive. It was always fun to be part of a group, in which an environment of high-level scientific research culture exists and flourishes.

During my studies, I had the opportunity to work together with many great people. Besides Prof. Hüllermeier, I have the privilege to co-author papers with Dr. Stefan Bösner, Dr. Weiwei Cheng, Prof. Juan José del Coz, Dr. Krzysztof Dembczyński, Prof. Norbert Donner-Banzhoff, Dr. Thomas Fober, Jörg Haasenritter, Dr. Dominik Heider, Sascha Henzgen, Dr. Oliver Hirsch, Dr. Edwin Lughofer, Maryam Nasiri, Prof. Maria Rifqi and Ammar Shaker. In addition, I would like to thank Amira Abdel-Aziz for their help in revising the first versions of this thesis.

Last but not least, I would like to thank my wife Jennifer and my kids Jamila and Rodney. They not only have been a constant motivation to me to finish my studies on time, but they also reminded me that work is just a part of it all.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

1.1 Machine Learning

Machine learning is one branch of the broad research field of *Artificial Intelligence* (AI). The primary goal of machine learning is to develop strategies, i.e., computer programs, which make use of data to improve their behavior. To be a bit more formal, I want to recall a more prominent definition by Tom M. Mitchell:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [67]

Several components are involved in this brief definition: To start with, we are talking about computer programs, the core of which mostly is an *algorithm*. Informally, an algorithm takes input values, processes these in a predefined sequence of instructions and returns a result as an output [26]. In general, algorithms are designed to solve well-defined problems. In the early days of computer science, several frameworks have been developed to describe the type of tasks (also called problems), which can be solved by certain kinds of algorithms. Some famous ones are: the *Turing machine* [103] introduced by Alan Turing and the *lambda calculus* [23] by Alonzo Church, just to name two.

Independent of the concrete definition, all these frameworks share one property: the algorithms have to be developed and implemented by humans. This circumstance introduces at least two difficulties: first, the programmer would need to know a solution to the problem at hand, and second, available human resources in terms of programming

craft are limited in number. These factors hinder the development of algorithms for difficult problems, because neither an exact nor an approximate solution is known or the implementation might take too long or costs too much to be economically efficient.

Examples of such tasks can be drawn from many domains like biology, medicine, multi-media, finance and many more. Some examples are:

- determining the most effective medication for a given patient (medicine)
- recognizing and determining concepts like faces or other objects in digital images (multi-media)
- determining the risk of a loan default (finance)
- determining the quality of a good, produced in a factory (production)

These are just a few examples of typical tasks for which machine learning is used. The wide spectrum of industrial sectors involved already foreshadows the huge amount of potential applications of machine learning. However, there is also a potential drawback in using a learning algorithm compared to a classical algorithm (if existing). The latter usually provides properties like completeness and correctness. In this case, a correct solution is provably existing for every instantiation of the problem at hand. In contrast, a learning algorithm usually does not guarantee a perfect solution. Rather, it produces a model which mimics the unknown mapping between a problem, taken as input, and its solution, provided as output with the help of examples provided as data.

Coming back to the definition by Mitchell, a machine learning approach to the above problems uses data – referred to as experience – in order to enable the computer to learn and improve according to some predefined performance measure.

In the remainder of this thesis, we will focus on a specific but still prevalent problem of machine learning i.e. *supervised learning*. It is also referred to by *learning from examples*. In the following, we will briefly introduce the basic setting and notation. For a more comprehensive introductions see [34, 46].

Supervised Learning

Data usually comprises a set of examples. Emanating from a supervised learning [46] setting, an example is a tuple $(\boldsymbol{x}, y)$, where $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{X}$ denotes the input –

also called instance – and $y \in \mathbb{Y}$ denotes the output. $\mathbb{X} = \mathbb{X}_1 \times \ldots \times \mathbb{X}_m$ denotes the instance space, from which each instance is drawn. It often conforms to the Cartesian product of domains $\mathbb{X}_j$, also referred to as attributes. Depending on $\mathbb{Y}$, several different learning tasks can be distinguished. If $\mathbb{Y}$ is a set of categorical values, the task is about *classification*. If $\mathbb{Y}$ equals the real numbers $\mathbb{R}$, it is a *regression* task. Some more types of learning problems will be introduced later.

Given a *training dataset* $\mathcal{T} \in (\mathbb{X} \times \mathbb{Y})^n$, the aim of a supervised learning algorithm $\mathcal{A}$ is to find a mapping $M : \mathbb{X} \to \mathbb{Y}$. Thus $\mathcal{A}$ itself implements a mapping

$$\mathcal{A} : \bigcup_{n \in \mathbb{N}} (\mathbb{X} \times \mathbb{Y})^n \to \mathcal{H}.$$

$\mathcal{H}$ denotes the so-called *hypothesis space*. M is called *hypothesis* or *model*. Applying the model to an instance produces a *prediction* $M(\boldsymbol{x}) = \hat{y}$.

In accordance with Mitchell's definition, it is not enough to find an arbitrary hypothesis M. Instead, M must be evaluated in terms of a performance measure, which we seek to maximize. Or, equivalently, a loss function $\mathcal{L} : (\mathbb{Y} \times \mathbb{Y}) \to \mathbb{R}$ to be minimized. To be more precise, the ultimate goal is to find a model M^*, which minimizes the *risk of error* $\mathcal{R}$:

$$M^* = \operatorname*{argmin}_{M \in \mathcal{H}} \mathcal{R}(M) = \operatorname*{argmin}_{M \in \mathcal{H}} \int_{(\boldsymbol{x},y) \in (\mathbb{X} \times \mathbb{Y})} \mathbf{P}((\boldsymbol{x}, y)) \cdot \mathcal{L}(y, M(x)) \ \mathrm{d}(\boldsymbol{x}, y) \tag{1.1}$$

$\mathbf{P}$ denotes the joint probability distribution over $\mathbb{X} \times \mathbb{Y}$. Since $\mathbf{P}$ is usually unknown, in practice, we have to rely on an estimate of $\mathcal{R}$. One common possibility is to use the *empirical risk* $\mathcal{R}^{emp}$:

$$M^* = \operatorname*{argmin}_{M \in \mathcal{H}} \mathcal{R}^{emp}(M) = \operatorname*{argmin}_{M \in \mathcal{H}} \frac{1}{|\mathcal{V}|} \sum_{(\boldsymbol{x},y) \in \mathcal{V}} \mathcal{L}(y, M(\boldsymbol{x})) \tag{1.2}$$

The validation set $\mathcal{V}$ is a set of examples, which has not been seen by the learning algorithm before, and which we assume is drawn from the same distribution as the training dataset $\mathcal{T}$. Since $\mathcal{T}$ and $\mathcal{V}$ are just samples, one seeks to be more sure about the risk estimate by repeating the steps of training and validation several times. A popular procedure in this regard is *cross-validation*. Given a dataset $\mathcal{D}$, cross-validation randomly splits $\mathcal{D}$ into k folds. Then the following steps are repeated for every fold:

1. Put the selected i^{th} fold aside and run the learning algorithm on the remaining $k-1$ folds.

2. Calculate an error estimation on the i^{th} fold.

In the end, all k error estimations are averaged. In practice, this procedure is used for *model selection*, i.e. many potential hypothesis classes are evaluated and the one which performs best usually is chosen for production. Since, this way the validation set become an intrinsic part of learning, the estimated error on the validation data is most often to optimistic. Therefore, it is recommended that another dataset, the *test set*, is used to estimate the most realistic performance for a productive system.

Capacity Control

The kind of error we are interested in is the so-called *generalization error*. We are explicitly interested in a small error on unseen data. Or, the other way around, we are not primarily interested in a low training error (the error on the training set); because a low training error does not necessarily imply a small generalization error. A model with a low training error and a high generalization error is likely to *overfit* the data it has seen. This is especially dangerous if a model class is very flexible like it is the case for FPTs.

In order to explain these phenomena, we assume some underlying (still unknown) data generating process, specified by $\mathbf{P}$. This might for example be an underlaying functional relationship between the input and output variables. Since we are ignorant about the true relationship, we have to make some assumptions about it. This we do by selecting a hopefully well fitting model class (e.g., polynomials of degree 2). These assumptions strongly influence the learning process. Their implication on the resulting model is also called *inductive bias*. Since we only make a guess about the correct model class, it is possible, that we either under- or overestimate their true complexity. For example, if the correct model class would be a polynomial of degree 3, then we have underestimated their complexity and we might encounter underfitting, whereas if the correct model class is a polynomial of degree 1 (linear function), then we are prone to overfitting.

There are many so-called *regularization* techniques to avoid overfitting [4, 15, 34]. Generally, they are based on the idea to restrict die complexity of a model class. The

specific technique to use of course strongly depends on the model class at hand. Sticking to the above example of polynomials, one way of restricting complexity would be to upper-bound the maximum degree of the polynomials considered during learning. Furthermore, another common approach is to constraint the weight vector, making sure the weights are small in magnitude. This is referred to as *shrinkage*. Other model classes like *decision trees* [85] and also, as will be seen in later sections, fuzzy pattern trees can be regularized by restricting the *size* of a model. In this regard, the notion "size" will be defined as the number of subcomponents of the model and we will see, that this is a reasonable measure for its complexity.

As a side remark, the different goals of minimizing generalization vs. minimizing training error is also one of the main differences between the fields of machine learning and *optimization* [95].

1.2 Fuzzy Set Theory

Lotfi A. Zadeh first introduced his concept of a *fuzzy set* in [118]. Fuzzy sets extend the mathematical concept of a (regular or ordinary) set by allowing elements to be contained in the fuzzy set to a certain degree. To be more precise, a fuzzy set A is identified by its so-called *membership function* μ_A. A membership function is a mapping $\mu : X \to [0, 1]$, where X denotes the *domain* or the *universe* of discourse. Elements $x \in X$ belong to A to the degree of membership (or simply degree) $\mu_A(x)$. Note that $\mu_A(x)$ can take any value in the unit interval. As a special case, when μ_A can only take two values (0 and 1), it is reduced to the *characteristic function* of an ordinary set.

Common choices for membership functions are piecewise linear functions like bounded linear functions, triangular or trapezoidal functions. Furthermore, also Gaussian membership functions are used in many applications.

Zadeh's intention was to enable a human user to precisely express his subjective perception of classes of objects, commonly refereed to in natural language. Examples are "short people", "much older than 5 years", or "high temperature". In the following, we will refer to such classes as *fuzzy concepts* or *fuzzy terms*. A membership function for the second fuzzy concept, which refers to numbers, might be the one illustrated in Figure 1.1.

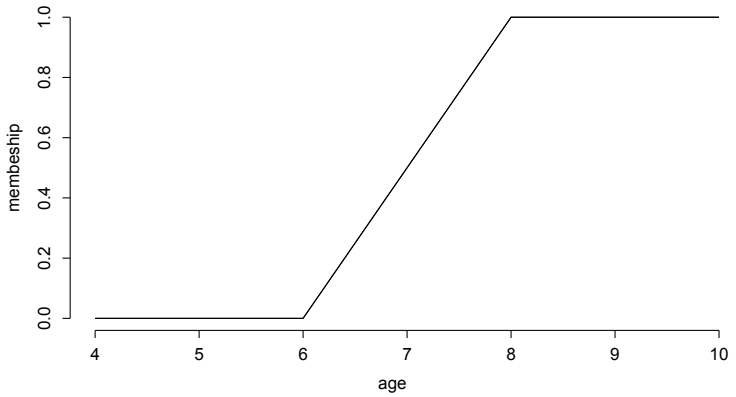


Figure 1.1: One possible formulization of a subjective perception of the fuzzy concept "much older than 5 years".

Working with fuzzy sets requires the ability to apply set operations. Basically, operations on fuzzy sets are defined in terms of their membership functions: The *complement* of A as denoted by $\bar{A}$ is defined by $\mu_{\bar{A}} = 1 - \mu_A$. Three more notions play a central role for ordinary sets as well as for fuzzy sets. These are *containment* ($A \subset B$), *union* ($A \cup B$) and *intersection* ($A \cap B$), which can be defined as:

- $\forall x \in X: \quad A \subset B \Leftrightarrow \mu_A(x) \leq \mu_B(x)$
- $\forall x \in X: \quad \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- $\forall x \in X: \quad \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$

The operators (min and max) above, are the ones originally used by Zadeh. Meanwhile, many more operators have been found, which can replace min and max, while keeping the same semantics of a conjunctive, respectively disjunctive, aggregation. These classes of operators are called *t-norms* and *t-conorms* [59].

A t-norm $\top(\cdot, \cdot)$ is a generalized conjunction, namely a monotone, associative and commutative $[0, 1]^2 \to [0, 1]$ mapping with neutral element 1 and absorbing element 0. Likewise, a t-conorm $\perp(\cdot, \cdot)$ is a generalized disjunction, namely a monotone, associative and commutative $[0, 1]^2 \to [0, 1]$ mapping with neutral element 0 and absorbing element 1.

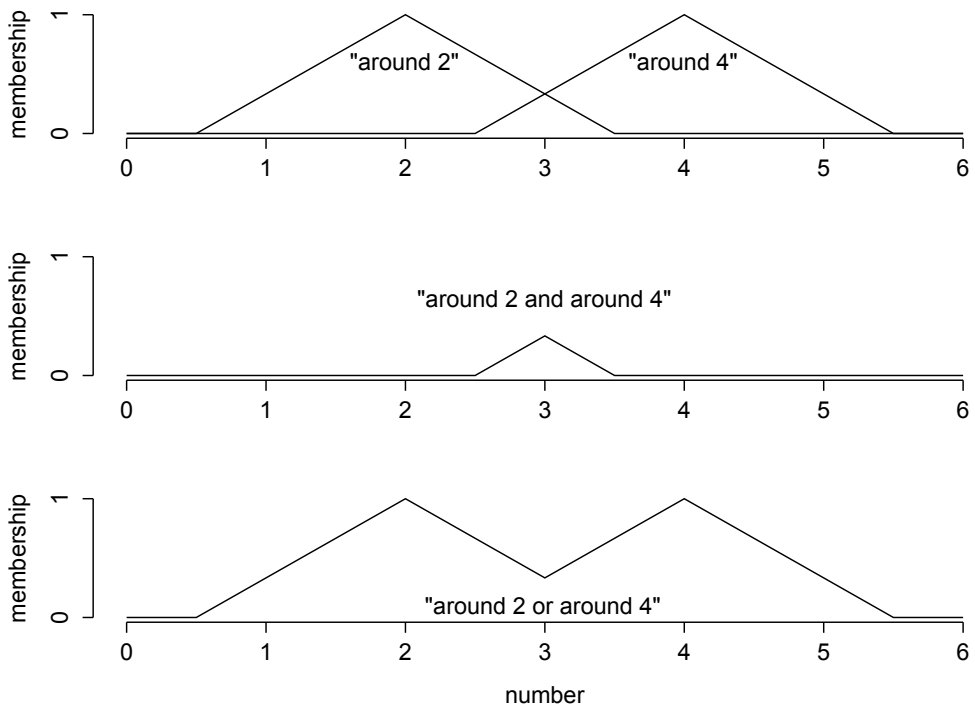


Figure 1.2: Example of two fuzzy numbers (top), which are combined with the minimum t-norm (middle) and with the maximum t-conorm (bottom).

Using t-norms and t-conorms, more complex fuzzy sets can be constructed by combining basic fuzzy sets. One simple example is shown in Figure 1.2. Starting with the fuzzy sets of two fuzzy numbers, namely *around 2* and *around 4* (top), one is able to derive the membership function of numbers, which belong to both sets simultaneously. In this example, the minimum operator was used as corresponding t-norm. The name of the derived fuzzy set might be *around 2 and around 4*. In the same way, the membership function of numbers, which are *around 2 or around 4* (bottom), is derived using the maximum operator.

Semantics of Fuzzy Sets

So far, we have set the theoretical foundations of fuzzy sets and their membership functions without caring about their semantics. Given a fuzzy set F and an element x, what does it actually mean when we say: "$\mu_F(x) = 0.7$"?

Like for probability theory, where probability is given a semantic meaning by either the "frequentist", the "subjectivist" or even some other less prevalent views [33], for fuzzy set theory there exist at least three different meanings referring to a degree of membership. Dubois and Prade gave a comprehensive overview of possible semantics in their work [31].

- *Similarity*: This view relates membership degrees to distances. This view is prevalent in many data analysis applications, where the distance to prototypical elements defines the membership i.e. the proximity of other elements to a class or cluster. It is supposable the oldest view and was suggested in [10].

  This view will also be the basis for this work and the methods which will be introduces in the following chapters.

- *Utility*: F in this case embodies a set of more or less preferred elements; $\mu_F(x)$ then determines the *utility* of each element. That is, $\mu_F(x)$ refers to the usefulness of x. This way fuzzy sets can act as soft constraints to optimization problems. The view was put forward in [11].

- *Uncertainty*: Zadeh introduced this interpretation in his works about *possibility theory* [120] and *approximate reasoning* [119]. A membership value $\mu_F(x)$ in these frameworks means the degree of possibility that a parameter x has a value u.

Towards Classification

Being able to dynamically construct more complex fuzzy sets with the help of fuzzy logical operators is especially interesting from a machine learning perspective. Fuzzy sets can be used as *gradual / fuzzy selectors* of instances, which e.g. shall belong to the black class in Figure 1.3. There, a simple two-dimensional example is presented.

The instances are represented by a vector (x_1, x_2). As can be seen in the figure, they are obviously arranged in three major groups, which are to some extent overlapping and may not be clearly separated from each other easily. Two of the groups belong to the white class (bottom-left and up-right), the other one belongs to the black class (top-left). At the top and the right axis of the scatter plot, trapezoidal fuzzy sets are shown (A, B and C, D), which are defined on their respective domain. Of course,

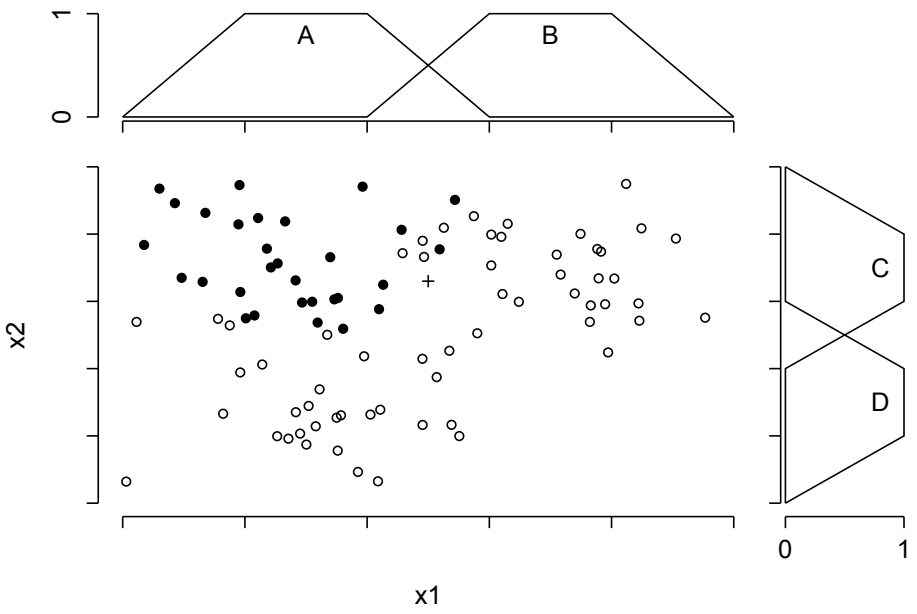


Figure 1.3: Example of elements existing in a 2-dimensional space, i.e. described by two attributes (x_1, x_2).

this example is idealized; in reality, class distributions seldom appear in such a strict arrangement.

Considering the task to select the instances of the black class, a reasonable choice of a fuzzy selector would be:

$$\mu_{S_{black}}(x_1, x_2) = \top(\mu_A(x_1), \mu_C(x_2)). \tag{1.3}$$

S_{black} as defined by its membership function in (1.3) conjunctively combines the two fuzzy sets A and C. Roughly speaking, in order to belong to the fuzzy set S_{black}, an instance has to belong to both fuzzy sets A and C, simultaneously. Here, the term "belonging to" refers to a high membership value, i.e., the instance has to produce a high output for both membership functions.

In the same way, we are able to create a fuzzy set for the white class:

$$\mu_{S_{white}}(x_1, x_2) = \bot(\top(\mu_B(x_1), \mu_C(x_2)), \top(\mu_A(x_1), \mu_D(x_2))) \tag{1.4}$$

First we select each of the two white groups individually as we did before for S_{black}, and then we aggregate them using a t-conorm. In words: An instance belongs to the white class, if it either belongs to *B and C or* it belongs to *A and D*.

Membership in this case is interpreted as similarity. Taking fuzzy set S_{black} as an example, it is constructed in a way, that it relates the membership of an element to its proximity to elements of the black class. The same applies for S_{white}, respectively.

This example already reveals an important benefit of using a fuzzy set instead of using an ordinary set in order to select instances of one class. The boundaries of the two classes are blurred, which is a common phenomenon in machine learning. Assuming sharp class boundaries when using ordinary sets (i.e. intervals) would not properly reflect the reality of the class distributions. Instead, for an instance, for which we do not know the actual class label and which is close to the class boundaries, both classes appear to be valid options to some degree.

Let us look at the instance x_{cross} like the one indicated as a cross in Figure 1.3. Concerning the white class, x_{cross} is located at the margin of the group of white elements, still within reach but also not directly covered. Hence, it seems reasonable to not give a full membership of 1 nor a membership of 0. The same reasoning also applies to the black class, independently.

After constructing S_{black} and S_{white} we not only have a fuzzy logical description of the two classes but we are also able to predict the class of an unknown instance $\boldsymbol{x}'$. To this end, we simply compare the membership values $S_{black}(\boldsymbol{x}')$ and $S_{white}(\boldsymbol{x}')$ and decide for the class with the higher membership.

As we already started to introduce fuzzy set theory in the realm of machine learning, in the following section we will further focus on this symbiosis.

1.3 Fuzzy Sets in Machine Learning

In the past, fuzzy set theory has already been used in the realm of machine learning in several regards. These include applications like *fuzzy clustering* [48], *fuzzy rule-based systems* [99, 109], *fuzzy decision trees* [54, 110] and *fuzzy association analysis* [21, 29], just to name a few.

In [51], Hüllermeier points out some potential contributions of fuzzy set theory to the field of machine learning. These include the ability to express fuzzy (or gradual)

concepts, like the ones we have discussed in the last section. In traditional machine learning, acquiring a definition of a general (non-fuzzy) concept by a set of positive and negative examples is also called *concept learning* [6]. Formally, a concept usually is expressed in terms of *predicates*, which are conditions on the properties of the objects. For example, a *bird* is a *small to medium size, winged, feathered, usually able to fly animal*. Extending concept learning to the fuzzy case has the following potential advantages:

- Many real world concepts are fuzzy by nature and do not have sharp boundaries. Therefore, it is inappropriate to make use of sharp constraints where there is no sharp boundary in the real world. Allowing fuzzy predicates enriches the concept description and makes it more realistic.

- A fuzzy concept can be considered particularly robust in the following sense: Comparing a standard interval on the real numbers with a fuzzy interval (trapezoidal fuzzy set), the former is prone to a strange "boundary effect", whereas the latter is not. This effect refers to the fact that a small variation of the boundary points of the interval may have a strong influence on the model in the interval case. The effect is alleviated when using fuzzy sets instead of intervals.

- Fuzzy set theory provides an interface between an arbitrary (most often a numeric) scale and a symbolic scale, which usually consists of linguistic terms. This provides a first layer of abstraction by utilizing natural language to describe complex objects. This potentially improves the interpretability of the formal description of a concept.

- In line with their improved interpretability, many fuzzy methods enable us to combine modeling and learning. This is especially true for rule-based systems as well as fuzzy pattern trees. For rule-based systems, experts are able to formulate if-then rules, roughly describing the input-output relation of the system. Then, implementing the linguistic terms employed by the expert in terms of fuzzy sets, we are able to tune the parameters of these fuzzy sets in an optimal way using the data we have observed. This is just one example of incorporating expert knowledge into the learning process. Many more variants can be thought of.

2

Fuzzy Pattern Trees

Fuzzy Pattern Trees (FPT) have independently been introduced by Huang et al. [49] and Yi et al. [116], who called this type of model *Fuzzy Operator Trees*. The FPT model class is related to several other model classes including *fuzzy rule-based systems* (FRBS), *fuzzy decision trees* (FDT) and *genetic programming* (GP). These model classes and their respective differences in comparison to FPTs are discussed in Chapter 5.

In this chapter we first introduce the basic constituent parts of the model class, then these parts are assembled into a fuzzy pattern tree. The introduction is followed by a discussion about the interpretatability of FPTs and the potential capability of an expert to use the model class for typical modeling [71, 72, 91] and machine learning tasks. After this, we focus on some aspects of FPTs, which are adjuvant in many applications and also form the first contributions of this thesis.

2.1 Model Overview

A fuzzy pattern tree is a hierarchical, tree-like structure, whose inner nodes are marked with generalized (fuzzy) logical and arithmetic operators, and whose leaf nodes are associated with fuzzy predicates on input attributes. It propagates information from the bottom to the top: A node takes the values of its descendants as input, aggregates them using the respective operator, and submits the result to its predecessor. Thus, an FPT implements a recursive mapping producing outputs in the unit interval.

Figure 2.1 shows an example of an FPT, which was trained from a wine quality

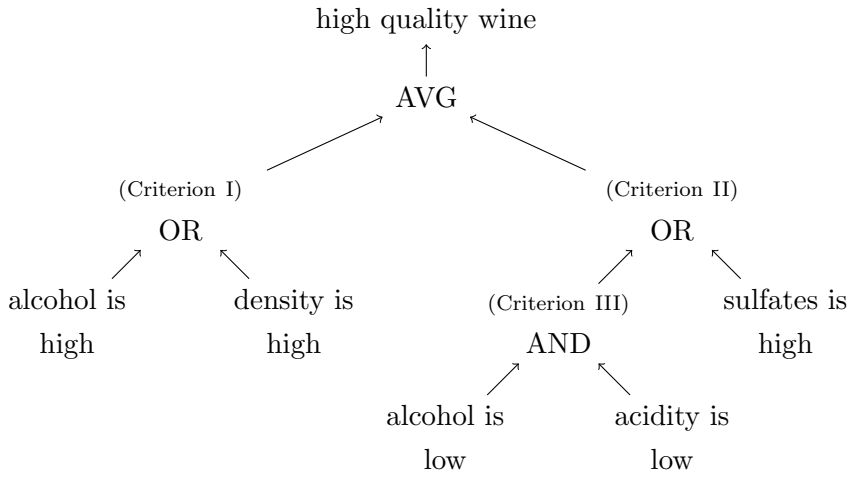


Figure 2.1: An interpreted example of a fuzzy pattern tree.

dataset[1]. It represents the fuzzy concept – a fuzzy criterion for – wine with a high quality. The node labels of the tree illustrate their interpretation and not yet their implementation.

In order to interpret the whole tree and grasp the *fuzzy pattern* it depicts, we start at the root node. It represents the final aggregation (a simple average in this case) and outputs the overall evaluation of the tree for a given instance (a wine). Then, we proceed to its children and so forth. The interpretation could be like this:

A high quality wine fulfills two criteria, which are able to compensate each other. We call these two criteria – the left and right subtree of the root node – criterion I and criterion II. Criterion I is fulfilled if the alcohol concentration of the wine is high or its density is high. Criterion II is fulfilled, if the wine has a high concentration of sulfates or a third criterion (III) is met. This is the case, if both alcohol concentration and the wines acidity is low.

Next, we will proceed with the implementation of the tree. Figure 3.17 shows the same model, but this time with more detailed information about the concrete operators, their parameters and the fuzzy sets involved. The average in the root node is realized by a weighted average operator (WA), which assigns the weight 0.65 to the left subcriterion, whereas the right one receives the remaining weight of 0.35. The AND and OR nodes are implemented by generalized logical operators (t-norms and t-conorms). Finally,

[1]The dataset is also part of the datasets used in the experiments in Chapter 4

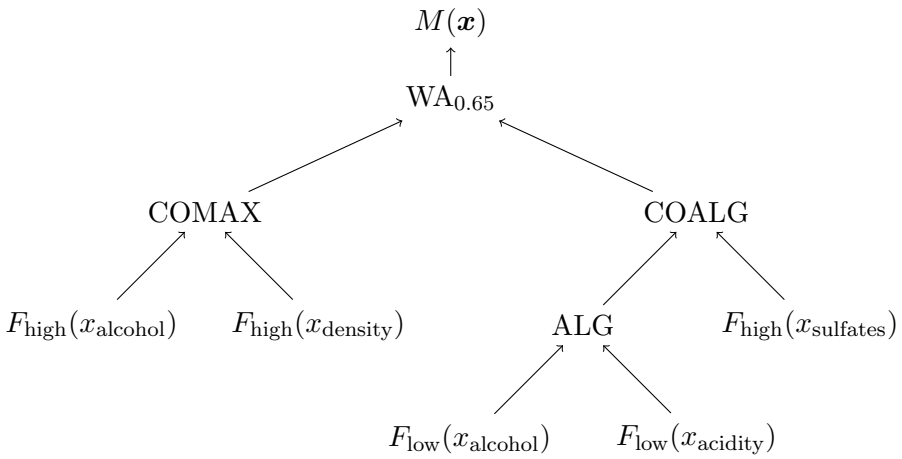


Figure 2.2: The same FPT as in the previous figure with additional information about the implementation of each node.

fuzzy sets – defined on their corresponding attribute domains – will be used in order to express the fuzzy linguistic terms contained in the leaf nodes.

2.2 Aggregation and Structure

As described above, internal nodes represent the aggregation of two membership values. The original set of aggregation operators used in [49] include three families of aggregation operators. These are *generalized conjunctions* (t-norms) and *generalized disjunctions* (t-conorms) as they where introduced in Section 1.2. Furthermore, two different averaging operators are used, which are the simple *weighted average* and the *ordered weighted average* [90, 114].

The operator set used in [49] is shown in Tables 2.1–2.2. The variables u and v denote the membership values, to be aggregated.

An ordered weighted average (OWA) combination of k numbers $v_1, v_2, \ldots, v_k$ is defined by

$$\mathrm{OWA}_w(v_1, v_2, \ldots, v_k) \stackrel{\mathrm{df}}{=} \sum_{i=1}^{k} w_i v_{\tau(i)}, \tag{2.1}$$

where τ is a permutation of $\{1, 2, \ldots, k\}$ such that $v_{\tau(1)} \leq v_{\tau(2)} \leq \ldots \leq v_{\tau(k)}$ and $w = (w_1, w_2, \ldots, w_k)$ is a weight vector satisfying $w_i \geq 0$ for $i = 1, 2, \ldots, k$ and $\sum_{i=1}^{k} w_i = 1$. Thus, just like the normal weighted average (WA), an OWA operator is parameterized

Name	Definition	Code
Minimum	$\min(u, v)$	MIN
Algebraic	uv	ALG
Lukasiewicz	$\max(u + v - 1, 0)$	LUK
Einstein	$\frac{uv}{2-(u+v-uv)}$	EIN

Table 2.1: Fuzzy operators: t-norms

Name	Definition	Code
Maximum	$\max(u, v)$	MAX
Algebraic	$u + v - uv$	COALG
Lukasiewicz	$\min(u + v, 1)$	COLUK
Einstein	$\frac{u+v}{1+uv}$	COEIN

Table 2.2: Fuzzy operators: t-conorms

by a set of weights. However, a weight does not directly refer to an argument, like in WA, but instead to a rank: w_i is the weight of the i-th smallest value among $v_1, v_2, \ldots, v_k$.

Note that for $k = 2$, which is the case of FPT, (2.1) boils down to

$$\mathrm{OWA}_\gamma(u, v) = \gamma \cdot \min(u, v) + (1 - \gamma) \cdot \max(u, v), \tag{2.2}$$

which is simply a convex combination of the minimum and the maximum. In fact, the minimum and the maximum operator are obtained, respectively, as the two extreme cases $\gamma = 1$ and $\gamma = 0$.

When defining an order relation "$\preceq$" on aggregation functions in agreement with the standard (point-wise) order on functions, then conjunctive aggregations A are those with $A \preceq min$, compensatory (averaging) those with $min \preceq A \preceq max$, and disjunctive those for which $max \preceq A$ [41].

Therefore, the class of OWA operators nicely "fills the gap" between the largest conjunctive combination, namely the minimum t-norm, and the smallest disjunctive combination, namely the maximum t-conorm.

2.2.1 Extending the Set of Operators

The original selection of operators is to some extent arbitrary. In general, there exist many different t-norm and t-conorms. Especially, there are parameterized families like the *Dubois & Prade* t-norm [35],

$$\mathrm{DP}_\alpha(u,v) = \frac{u+v-uv-\min(u,v,1-\alpha)}{\max(1-u,1-v,\alpha)}, \text{ where } 0 \leq \alpha \leq 1 \tag{2.3}$$

the *Hamacher* t-norm [45]

$$\mathrm{H}_\alpha(u,v) = \frac{uv}{\alpha+(1-\alpha)(u+v-uv)}, \text{ where } 0 \leq \alpha \leq +\infty \tag{2.4}$$

and many more. Parametric operators like these have an important advantage in the realm of machine learning. Their parameters can be optimized to best fit the data at hand. This potentially allows a more accurate fit of the overall model. It should be noted, however, that allowing for more and more operators also yields a higher runtime of the learning algorithms, which will be introduced in Chapter 3. Therefore, it is desirable to find a set of operators, which on the one hand is small and can efficiently be optimized, and on the other hand is at least as expressive as the original operator set. So far, we are not aware of a single parameterized operator family, which could be used to substitute all originally used t-norms, respectively t-conorms. In the following, three parameterized operators are proposed to be used for the FPT model class, one for each class of aggregation operators: t-norms, t-conorms and averaging operators.

To start with, WA and OWA are replaced by the so-called *Choquet integral* (CI) [41]. In order to define this operator in a formally correct way, it should be written as an integral of a function with respect to a suitable non-additive measure. However, in the discrete case with only two input arguments, one can show that it reduces to the following simple expression:

$$\mathrm{CI}(u,v) \quad = \quad \begin{cases} (1-\beta)u+\beta v & \text{if } u \leq v \\ \alpha u+(1-\alpha)v & \text{if } u > v \end{cases}, \tag{2.5}$$

where $\alpha, \beta \in [0,1]$. Note the following special cases: CI = min and CI = max for $(\alpha,\beta) = (0,0)$ and $(\alpha,\beta) = (1,1)$, respectively; for $\beta = 1-\alpha$, one obtains the WA and for $\beta = \alpha$ the OWA operator.

Especially interesting from a computational point of view, is the existence of an efficient way to approximate the optimal parameters α and β. For this purpose, we first use a closed form solution in order to minimize the squared loss of the operator on our training data. This solution, however, does not ensure the parameters to reside in $[0, 1]$. Hence, we simply force them to: If α is smaller than 0, we set it to 0; if it is bigger than 1, we set it to 1. The same procedure is applied to β. This may yield a suboptimal solution, however, in the general case it is assumed to be close to optimal. This procedure is commonly used for constraint optimization problems [18] and was also implemented by Huang et al. to determine the parameters of the WA and OWA operators.

In order to further provide a real extension of the current t-norms, respectively t-conorms, we use two convex combinations:

$$\begin{aligned}\mathrm{CC}(u,v) :=& \gamma_1 \cdot \mathrm{MIN}(u,v) + \gamma_2 \cdot \mathrm{ALG}(u,v) + \\ & \gamma_3 \cdot \mathrm{LUK}(u,v) + \gamma_4 \cdot \mathrm{EIN}(u,v)\end{aligned}$$

$$\begin{aligned}\mathrm{COCC}(u,v) :=& \gamma_1 \cdot \mathrm{MAX}(u,v) + \gamma_2 \cdot \mathrm{COALG}(u,v) + \\ & \gamma_3 \cdot \mathrm{COLUK}(u,v) + \gamma_4 \cdot \mathrm{COEIN}(u,v)\end{aligned}$$

$$\text{where } \gamma_i \geq 0, \;\; i \in \{1,2,3,4\} \text{ and } \sum_{1}^{4} \gamma_i = 1.$$

Note that CC and COCC are actually no longer t-norms, respectively t-conorms, because they do not satisfy the associativity property. However, one can easily prove, that CC is a *weak t-norm* and COCC is a *weak t-conorm* as they were introduced by Yager in [115]. Although CC and COCC do not satisfy associativity, they at least satisfy *quasi-associativity*. Furthermore, associativity is never used in the realm of FPT. Although the interpretation of weak t-(co)norms is not exactly the same as of t-(co)norms, [115] describes them as being "and-like" and "or-like", respectively. This is obvious for some special cases of γ_i. Whenever there is one γ_i, which takes all the weight (e.g. $\gamma_1 = 1$), the CC and COCC operators actually coincide with the i-th t-(co)norm.

In order to find good parameters for these operators as well as parametric t-norms and t-conorms, there might be the need for optimization methods like *gradient descend* [13] or even *evolutionary strategies* like CMA-ES [14].

2.3 Important Properties of Fuzzy Pattern Trees

Apart from the properties of the FPT model class we already discussed, it exhibits two other very interesting properties.

Monotonicity

The first one is the ability to model *monotonicity constraints* [32, 79]. The type of monotonicity constraint which is meant here refers to the type of mapping a tree implements. All operators introduced so far are monotonically increasing in their arguments, i.e.

$$\mu_A(x) \leq \mu_A(x') \text{ and } \mu_B(x) \leq \mu_B(x') \Rightarrow \phi(\mu_A(x), \mu_B(x)) \leq \phi(\mu_A(x'), \mu_B(x')), \quad (2.6)$$

where $\phi(\cdot, \cdot)$ denotes an operator, $\mu_A(\cdot)$ and $\mu_B(\cdot)$ denote membership functions and x and x' denote attribute values. Since every single operator is monotonically increasing, every composition of these operators is also monotonically increasing. This means that the whole tree is monotonically increasing in the membership values given by the leaf nodes. Therefore, the question of how the output of the tree is influenced by an attribute depends on how the leaf nodes' membership depends on its attribute.

Note however, that the fuzzy sets in a tree are not necessarily monotonic. A regular triangular or trapezoidal fuzzy set, for example, is not a monotonic function. Nevertheless, there are of course fuzzy sets, that are monotonic (e.g. Figure 1.1). They can even be monotonically increasing or decreasing. This actually allows us to constrain the influence of each single attribute in a way, which is suggested by background knowledge about the problem domain.

Consider, for example, the problem of making a diagnosis for a given patient. To be able to decide whether or not the patient is suffering from a certain disease, the physician conducts several tests – e.g. measuring the temperature, blood pressure, gathering important information by asking the patient or even measuring the level of certain ingredients in the blood of the patient. Assuming that each single test returns a score (or probability), in many cases these scores monotonically increase or decrease together with the risk of actually suffering from the disease. The *Marburg heart score* [19] constitutes a prevailing example. Roughly speaking, it aggregates several risk factors for the purpose of predicting a severe heart disease. The presence compared to

the absence of a risk factor, in this case, shall only increase the probability of suffering from the disease.

In order to learn an aggregation function on data of that kind, monotonically constraining the influence of each single test by only allowing monotonically increasing fuzzy sets like 'high of age' (cf. Figure 1.1) can guide the learning process and improve the results.

This is especially true for problems with *sparse* data as was shown in [5]. Sparsity in this regard means that there are only relatively few observations available to learn from, compared to the dimensionality of the problem. This situation is also referred to as *curse of dimensionality* [9]. Roughly speaking, when the dimensionality increases, the volume of the instance space and in the same way the size of the hypothesis space increases so fast that the available data becomes sparse. In this case, each type of background knowledge reducing the model space, including monotonicity constraints, will be useful.

Handling Missing Values

A common obstacle in machine learning is the problem of dealing with *missing values*. A missing value refers to the absence of a value for a certain attribute of an instance. Missing values can be treated in many different ways, depending on the assumptions about the reason of missing the value. In the literature [87, 89], three main types are distinguished, namely *missing completely at random* (MCAR), *missing at random* (MAR) and *missing not at random* (MNAR).

MCAR denotes the situation where missing values of an attribute occur for a random subset of instances. The second type denotes missing values, which occur (statistically) depending on an observed attribute. In this regard, the name MAR is vexing, since the values do not occur completely at random considering the information of the dependent attribute. Nevertheless, the dependence between an observed attribute and the occurrence of a missing value in another attribute may be of any strength. Except for full dependence, the occurrence stays random. MNAR denotes the setting, when the reason for a missing value (potentially fully) depends on unknown or unobserved information.

It can be determined from data, whether data is MCAR, whereas it is impossible to determine the cases of MAR and MNAR [87, 89].

The simplest way to handle missing values is to entirely exclude instances with missing values from the dataset, which is called *complete case analysis*. When data is MCAR, complete case analysis leads to unbiased results. However, in many cases data is not MCAR, but MAR or MNAR. In these cases, omitting every incomplete instance leads to biased results [89]. Another drawback of the complete case analysis is a loss of information. Especially for small datasets with many missing values, this reduces the information to learn from dramatically. Therefore, it is desirable to deal with incomplete instances differently.

Depending on the machine learning technique one wants to apply, it is necessary to include a preprocessing step and apply *imputation* methods. These methods replace missing by estimated values. How the estimations are calculated depends on the imputation method. Van Buuren provides an overview on imputation techniques in [104]. However, it is more elegant, if a machine learning method is capable of directly dealing with missing values in a reasonable way. For FPT, this is the case, as will be explained in more detail in the remainder of this subsection. To this end, we will show how a tree is evaluated in case of missing values. This is sufficient for all of our current induction algorithms to work properly with missing values.

Assuming that we have to determine the membership degree of an instance $\tilde{\boldsymbol{x}} \in \mathbb{X}$ with missing values for the fuzzy pattern tree M, we will make use of the monotonicity of the tree in the following way. $\tilde{\boldsymbol{x}}$ shall exhibit missing values at an arbitrary but positive number of attributes, for example $\tilde{\boldsymbol{x}} = (x_1, x_2, \emptyset, x_4, \emptyset)$, where $\emptyset$ denotes a missing value. In this example, the values for the third and fifth attribute are missing. For simplicity and without loss of generality, we also assume that there is only one fuzzy set for each attribute $(\mu_1(\cdot), ..., \mu_5(\cdot))$. This means, that we are not able to evaluate the fuzzy sets $\mu_3(\cdot)$ and $\mu_5(\cdot)$, but we definitely know in which range the unidentified membership values must reside, namely $[0, 1]$.

Now, for the following, it is adjuvant to define $\mathcal{M}$ to be M after replacing all fuzzy sets in the leaf nodes by the identity function $id(\cdot)$. Then it yields that

$$M((\cdot, \cdot, \cdot, \cdot, \cdot)) \equiv \mathcal{M}((\mu_1(\cdot), \mu_2(\cdot), \mu_3(\cdot), \mu_4(\cdot), \mu_5(\cdot))). \tag{2.7}$$

Due to monotonicity, we additionally know that

$$\begin{aligned}
\underline{\nu} := \quad & \mathcal{M}((\mu_1(x_1), \mu_2(x_2), 0, \mu_4(x_4), 0)) \\
\leq \quad & \mathcal{M}((\mu_1(x_1), \mu_2(x_2), 1, \mu_4(x_4), 0)) \\
\leq \quad & \mathcal{M}((\mu_1(x_1), \mu_2(x_2), 1, \mu_4(x_4), 1)) \quad =: \overline{\nu}
\end{aligned}$$

Due to the monotonicity of the tree in the membership values, we can now conclude, that the proper membership degree for the incomplete instance $\tilde{\boldsymbol{x}}$ must reside in $\nu = [\underline{\nu}, \overline{\nu}]$. In principle, this interval could be directly returned to the user. Although, we were not able to identify the proper exact membership because of the missing information, at least we are able to confine the interval of possible values. If we are forced to make a point prediction, the simplest way is just predicting the mean of $\underline{\nu}, \overline{\nu}$.

Depending on the learning task, which we will discuss in the next chapter, it is sometimes already enough to know ν in order to make a prediction (e.g., in classification). Additionally, we will see that we can utilize the ability to determine ν efficiently, in order to reduce the costs of a prediction.

2.4 Universal Approximation Property

As we have seen in the Chapter 1, learning in our case involves the minimization of the empirical risk (1.2). Assuming our data comes from a probabilistic data generating process $\mathbf{P}$ and let

$$f(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{y}} E(\mathcal{L}(\hat{y}, y)|\boldsymbol{x}) \tag{2.8}$$

$$= \operatorname*{argmin}_{\hat{y}} \int \mathcal{L}(\hat{y}, y) \, \mathrm{d}\, \mathbf{P}(\boldsymbol{x}, y) \tag{2.9}$$

be the so-called *Bayes predictor*, which attains the lowest possible prediction error. Then the question arises, if we are able to closely approximate f by means of a fuzzy pattern tree model. This would be a hint towards the belief that we are able to succeed in the minimization of (1.2).

To be more precise, in this section, we will give a proof, that there always exists an FPT M to approximate an arbitrary function $g : [0,1]^m \to \mathbb{R}$ up to any degree of accuracy. This ability is called *universal approximation* property.

The proof is constructive in the sense that given a function g, we construct an FPT, which implements a function $\tilde{g}$, to fulfill the desired approximation.

Theorem. *(FPTs are Universal Approximators) Let $m \in \mathbb{N}$ and $g : [0,1]^m \to [0,1]$ be an arbitrary continuous function. Let $\epsilon \in \mathbb{R}$ with $\epsilon > 0$. Then, there exists a fuzzy pattern tree M, which implements a function $\tilde{g}$, with*

$$|g(\boldsymbol{x}) - \tilde{g}(\boldsymbol{x})| < \epsilon \ \ \forall \boldsymbol{x} \in [0,1]^m.$$

Proof. Because g is continuous, for every $\boldsymbol{x_1} \in [0,1]^m$ and every $\epsilon > 0$ there exists a $\delta_{\boldsymbol{x_1},\epsilon} > 0$, such that

$$\forall \boldsymbol{x_2} \in [0,1]^m : |\boldsymbol{x_1} - \boldsymbol{x_2}| < \delta_{\boldsymbol{x_1},\epsilon} \Rightarrow |g(\boldsymbol{x_1}) - g(\boldsymbol{x_2})| < \epsilon \tag{2.10}$$

(Weierstrass definition) [86]. Because a continuous function, defined on a compact set is *uniformly continuous*, it even holds that:

$$\exists \delta_\epsilon > 0 \ \ \forall \boldsymbol{x_1}, \boldsymbol{x_2} \in [0,1]^m : |\boldsymbol{x_1} - \boldsymbol{x_2}| < \delta_\epsilon \Rightarrow |g(\boldsymbol{x_1}) - g(\boldsymbol{x_2})| < \epsilon \tag{2.11}$$

Let us select δ_ϵ according to (2.11).

In the following, we want to span a grid in $[0,1]^m$, for which it holds, that:

$$\forall \boldsymbol{x} \in [0,1]^m \ \ \exists \boldsymbol{p}^{(j)} \in G : |\boldsymbol{x} - \boldsymbol{p}^{(j)}| < \delta_\epsilon$$

Such a finite grid exists, because $[0,1]^m$ is bounded and $\delta_\epsilon > 0$. In order to create the grid, let

$$\gamma = \sqrt{\frac{(2\delta_\epsilon)^2}{m}} \ . \tag{2.12}$$

γ is the maximum Euclidean distance, which two neighboring grid points may have when they get projected onto one dimension. Now, let

$$k = \left\lceil \frac{1}{\gamma} \right\rceil \ \text{ and } \ \gamma' = \frac{1}{k} \leq \gamma \tag{2.13}$$

Let $\{c_1, c_2, ..., c_k\} = \{\frac{\gamma'}{2}, \frac{3\gamma'}{2}, ..., \frac{(2k-1)\gamma'}{2}\}$ be the possible grid point values on each single dimension. Then we finally can define the grid points as:

$$G = \{(c_{i_1}, ..., c_{i_k}) | c_{i_j} \in \{c_1, ..., c_k\}\} \tag{2.14}$$

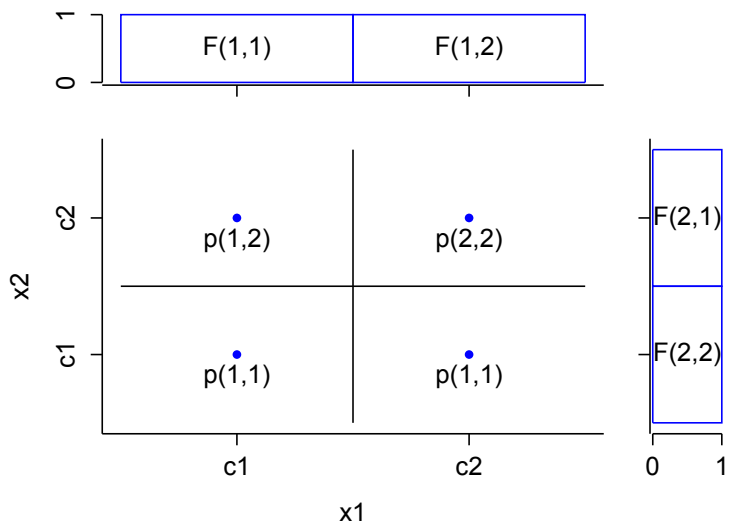


Figure 2.3: Example of a grid in two dimensions with two fuzzy sets for each dimension.

This yields $|G| = k^m$. Figure 2.3 illustrates the grid by an example.

For each point $\boldsymbol{p} \in G$ let us create a small subtree $S_{\boldsymbol{p}}$, which by itself consists of two parts: the *selector subtree* $S_{\boldsymbol{p}}^{Sel}$ and the *value subtree* $S_{\boldsymbol{p}}^{Val}$. Both are then aggregated by means of the conjunctive *min* operator. $S_{\boldsymbol{p}}^{Sel}$ "selects" exactly one grid cell, which means, it assigns full membership to those instances $\boldsymbol{x}$ that are close enough to $\boldsymbol{p}$ and zero membership to all other. This is accomplished with the help of a set of k fuzzy sets Fi, j for each of the m dimensions. The i-th fuzzy set of the j-th dimension $F_{i,j}$ is defined as:

$$F_{i,j}(\boldsymbol{x}) = \begin{cases} 1 & c_j - \frac{\gamma'}{2} \leq x_i < c_j + \frac{\gamma'}{2} \\ 0 & otherwise \end{cases} \tag{2.15}$$

Figure 2.4 shows a selector for the grid cell centered by $(c_1, ..., c_1) = (\frac{\gamma'}{2}, ..., \frac{\gamma'}{2})$.

$S_{\boldsymbol{p}}^{Val}$ will output a constant value of $g(\boldsymbol{p})$. We could either just use a constant membership function at the membership degree $g(\boldsymbol{p})$, but this fuzzy set would not be normalized, which is a common requirement. Therefore, instead we use an aggregation of the two fuzzy sets:

$$\text{low}(\boldsymbol{x}) = \begin{cases} 1 & x_1 < 0.5 \\ 0 & else \end{cases}$$

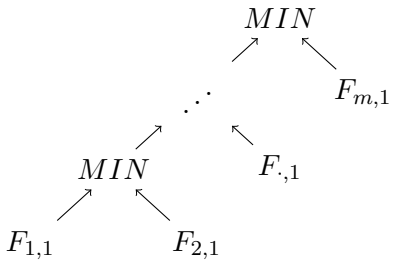


Figure 2.4: The implementation of the selector subtree $S_{\boldsymbol{p}}^{Sel}$ for the grid point $\boldsymbol{p} = (c_1, ..., c_1)$.

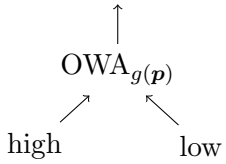


Figure 2.5: The implementation of a constant function $g(\boldsymbol{p})$.

$$\text{high}(\boldsymbol{x}) = \begin{cases} 1 & x_1 \geq 0.5 \\ 0 & else \end{cases}$$

Aggregating these two fuzzy sets with an OWA operator with a weight of $\alpha = g(\boldsymbol{p})$ produces the desired membership function. Figure 2.5 illustrates the value subtree.

Then, we construct a tree, which implements the following function:

$$\tilde{g}(\boldsymbol{x}) = \max_{p \in G} \{\mu_{S_{\boldsymbol{p}}}(\boldsymbol{x})\}$$

This is achieved by successive pairwise aggregation of all subtrees $S_{\boldsymbol{p}}$ for all $\boldsymbol{p} \in G$. Figure 2.6 demonstrates the tree structure.

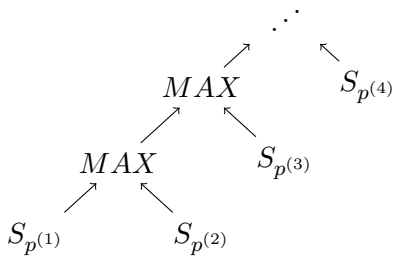


Figure 2.6: The implementation of the function $\tilde{g}(\cdot)$ using a fuzzy pattern tree.

Now, we have to reason about $|g(\boldsymbol{x}) - \tilde{g}(\boldsymbol{x})|$. Let $\hat{\boldsymbol{x}} \in [0,1]^m$ be arbitrary but fixed. Let $G_0 = \{\boldsymbol{p} \in G : |\hat{\boldsymbol{x}} - \boldsymbol{p}| \geq \delta_\epsilon\}$ and let $G_+ = \{\boldsymbol{p} \in G : |\hat{\boldsymbol{x}} - \boldsymbol{p}| < \delta_\epsilon\}$. Then, it holds:

1. $\forall \boldsymbol{p} \in G_0 : \mu_{S_p}(\hat{\boldsymbol{x}}) = 0$
2. $\forall \boldsymbol{p} \in G_+ : |\mu_{S_p}(\hat{\boldsymbol{x}}) - g(\hat{\boldsymbol{x}})| < \epsilon$
3. G_+ is not empty.

Therefore, we can conclude, that also $|\tilde{g}(\hat{\boldsymbol{x}}) - g(\hat{\boldsymbol{x}})| < \epsilon$.

□

This proof holds for all functions $g : [0,1]^m \to \mathbb{R}$. In Section 3.1, however, we will introduce a technique that enables us to easily extend the input space of g from $[0,1]^m$ to any compact set $\mathbb{X}$.

2.5 Vapnik-Chervonenkis Dimension

The well-known Vapnik-Chervonenkis (VC) dimension [17, 105, 106, 107] is a general measure of the potential capacity of model classes. This number is interesting for at least two reasons. First, it provides a rough idea of how powerful a model class is. "Powerful", in this regard, means the ability of dealing with complex (especially non-linear) problems. Second however, for a potentially very complex model class, it is easier to overfit (see Section 1.1) than for less complex classes. This has to be taken into account when using it and therefore, we will analyze the model class of fuzzy pattern trees in the following.

Assuming a number of d points $\mathbf{X} = \{x_i | i = 1, ..., d\}$ in the input space. Then, it is theoretically possible to assign a positive or negative class (setting of binary classification) to every point in 2^d many ways. This is equivalent to selecting a subset $C \subset \mathbf{X}$ of arbitrary size. Such a subset is also denoted as *concept*.

A model class $\mathcal{M}$ is said to *shatter* d points, if for every concept $C \subset \mathbf{X}$ there exists a model $M \in \mathcal{M}$, which is *consistent* with C. Consistency, in this regard, means

$$M(x) = 1 \Leftrightarrow x \in C.$$

The model class $\mathcal{M}$ exhibits a VC-dimension of d, if d is the largest number for which the above condition holds.

In order to prove, that the VC-dimension of a model class is d, one has to proceed in the following way. First prove that the VC-dimension is at least d and then, that it is not $d+1$. In our case however, we will easily prove that the model class of fuzzy pattern tree classifiers has an unlimited VC-dimension.

Theorem. *The model class of fuzzy pattern tree classifiers has a VC-dimension of* ∞.

Proof. Let $d \in \mathbb{N}$ be arbitrary but fixed. Let $\mathbf{X}$ be a set of d points $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(d)}\} \subset [0,1]^m$. Let the elements of $\mathbf{X}^+ \subset \mathbf{X}$ be labeled positive (1) and the elements of $\mathbf{X}^- = \mathbf{X} \setminus \mathbf{X}^+$ be labeled negative (0).

Like in the previous proof, we span a grid in $[0,1]^m$. This time, each cell of the grid shall contain at most one point $\boldsymbol{x} \in \mathbf{X}$. Such a grid exists, because $\mathbf{X}$ is finite. In order to create the grid, let

$$\gamma = \min\left\{|x_i^{(j)} - x_i^{(j')}| \;\middle|\; \forall i \in \{1,...,m\} \text{ and } j, j' \in \{1,...,d\}\right\}.$$

γ is the minimum gab between a pair of points projected onto one dimension. Still, we need to take the smallest distance between a point and the boarders of $\mathbb{X}$ into account. Therefore, let

$$\gamma' = \min\left\{\gamma, (1 - x_i^{(j)}), x_i^{(j)} \;\middle|\; \forall i \in \{1,...,m\} \text{ and } j \in \{1,...,d\}\right\}.$$

Just like before, let $\{c_1, c_2, ..., c_k\} = \{\frac{\gamma'}{2}, \frac{3\gamma'}{2}, ..., \frac{(2k-1)\gamma'}{2}\}$ be the possible grid point values on each single dimension. Then we can define the grid points as:

$$G = \{(c_{i_1}, ..., c_{i_k}) | c_{i_j} \in \{c_1, ..., c_k\}\} \tag{2.16}$$

See Figure 2.3 for an illustration of the grid.

The construction of the tree complies with the one in our previous proof with one exception. The value subtree $S_{\boldsymbol{p}}^{Val}$ will output a constant value of 1 only in case there exists a point $\boldsymbol{x}_i \in \mathbf{X}^+$ located within the cell centered by $\boldsymbol{p}$, else it outputs 0.

Concluding the construction, the resulting tree will output 1 for a point $\boldsymbol{x} \in [0,1]^m$ if and only if $\boldsymbol{x}$ belongs to a cell of the grid, which already contains an element of $\mathbf{X}^+$. Therefore, the output of the elements of $\mathbf{X}^+$ themselves is 1. Due to the construction of the grid (each cell at most contains one element of $\mathbf{X}$) all elements of $\mathbf{X}^-$ receive an output of 0.

Hence, the tree shatters $\mathbf{X}$ and because d was set arbitrarily we have proven that the VC-dimension of fuzzy pattern tree classifiers is ∞.

□

This result suggests, that for arbitrarily complex concepts, there is an FPT model, which is adequate in terms of capacity. Having said that, as already mentioned before, highly flexible model classes are prone to overfitting. This is especially true, when the VC-dimension is even infinite. Theoretical results in the realm of the *PAC Learning Theory* [57] have shown, that it might be even impossible at all to learn for such model classes. Therefore, it is important to be able to adjust the models' class complexity as needed.

One way of doing this is to limit the size of the trees. Smaller trees are less expressive and hence exhibit a lower VC-dimension. In later sections when we introduce algorithms, for building fuzzy pattern tree models from data, we will see, that these algorithms always start with small trees, which stepwise become larger until they terminate. This way, the *effective* VC-dimension stays finite and in fact, it dynamically increases to hopefully match the complexity of the learning task.

This approach is in line with other algorithms inducing models of different model classes like *rule-based systems* and *decision trees*. These will be discussed in Chapter 5. Just like for fuzzy pattern trees these algorithms start with small, hence simple, models that grow until some termination criterion is met.

In the realm of PAC learning, this basic strategy is justified by theoretical results [57]. The so-called *Occam algorithms* not only try to find a consistent hypothesis but additionally prefer small ones. The *size* of an hypothesis usually is defined by the length of its representation. The idea of Occam algorithms is that a simpler hypothesis better generalizes to unseen data than complexer ones. This is basically what *Occams razor* [16] suggests. Hence the name.

3

Learning Fuzzy Pattern Trees

In the last chapter, we have seen how an expert is able to model an FPT using his/her expert knowledge, hence this approach can be entitled *knowledge-driven*. In this section, we focus on another way of FPT construction, namely algorithmic or *data-driven* approaches to induce FPT models. As already seen in the introductory section, data usually comprises a set of training examples

$$\mathcal{T} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n} \subset \mathbb{X} \times \mathbb{Y} \ .$$

Being able to induce FPT models utilizing data is helpful in at least two regards. First, sometimes one is facing a new prediction problem for which there is no expert knowledge available. In this situation, there is no expert able to model an FPT. And second, even though there might be expert knowledge available, comparing an expert model with an algorithmically induced model can yield new insights and ignite an alternating development process.

To this end, we will introduce several algorithms starting with an already existing one by Huang, Gedeon & Nikravesh [49], followed by several new ones. For each algorithm, we first motivate its development by a discussion about drawbacks of existing approaches, then introduce the method itself as a potential solution to these drawbacks and in the end validate its success in an experimental study.

In order to provide comprehensive overall studies, the experimental setup is consistent throughout this work. When comparing learning algorithms[1] we always use a 3-times 10-fold cross validation procedure. Most of the time we will use 40 classification

[1] All upcoming algorithms are implemented in the WEKA Machine Learning Framework [112].

datasets listed in Table 3.1 for comparing different variants of FPT induction. However, in Section 4.4 we will evaluate the performance of FPT on 12 regression datasets. Both sets are assembled from and freely available at the UCI [7] and STATLIB [66] repositories. Tables 3.1 and 3.2 also summarize some of their properties: the number of instances (#instances), the number of numerical attributes (#num) and the number of nominal attributes (#nom). Additionally, for classification datasets the number of classes (#classes) and for regression datasets the mean and the standard derivation of the output variable are provided.

Before we actually concentrate on the learning algorithms we have to take care of two prerequisites. In the following section we start with the preparation of data. Since FPTs are designed to work with "fuzzy data" and most data is not of this type, we have to add a pre-processing and a post-processing step. These steps can be implemented with the help of an expert. However, because an expert might not be available, in the following section we propose a generic way to transform regular data into data applicable to FPTs. The second requirement concerns the optimization of parameters for the newly introduced CI operator. The optimization procedure will be an important component used by every algorithm discussed.

No.	Name	#instances	#num	#nom	#classes
1	analcatdata-braziltourism	411	4	4	6
2	analcatdata-cyyoung8092	97	7	3	2
3	analcatdata-germangss	400	1	4	4
4	analcatdata-homerun	163	13	14	2
5	analcatdata-lawsuit	264	3	1	2
6	australian	690	6	9	2
7	authorship	841	69	1	4
8	autos	205	15	10	6
9	balance-scale	625	4	0	3
10	biomed	209	7	1	2
11	blood	748	4	1	2
12	bupa	345	6	0	2
13	cancer	683	9	1	2
14	cars	406	6	1	3
15	cloud	108	6	1	4
16	cmc	1473	2	8	3
17	confidence	72	3	0	6
18	credit	690	6	10	2
19	fl2000	65	14	2	3
20	flag	194	17	11	8
21	flare2	1065	0	10	7
22	german	1000	7	14	2
23	glass	214	9	0	6
24	haberman	306	3	1	2
25	heart	270	7	7	2
26	ionosphere	351	34	1	2
27	iris	150	4	1	3
28	irish	500	2	3	2
29	lupus	87	3	0	2
30	lymphography	148	3	15	4
31	metStatRST	336	3	0	12
32	pima	768	8	0	2
33	primary-tumor	339	0	17	22
34	prnn-crabs	200	6	1	2
35	prnn-synth	250	2	0	2
36	schizo	340	12	2	2
37	sonar	208	60	0	2
38	vehilce	846	18	1	4
39	wine	178	13	1	3
40	zoo	101	1	15	7

Table 3.1: Properties of the datasets used in classification experiments.

No.	Name	#instances	#num	#nom	mean	stddev
44	auto-mpg	390	8	0	23.42	7.81
45	concrete	1030	9	0	35.82	16.71
46	flare1M	323	8	3	0.14	0.48
47	flare2C	1066	8	3	0.3	0.84
48	forestfires	517	11	2	12.85	63.66
49	housing	506	14	0	22.53	9.2
50	imports-85	205	16	10	13207.13	7868.77
51	machine	209	7	2	105.62	160.83
52	servo	167	3	2	1.39	1.56
53	slump	103	11	0	36.04	7.84
54	winequality-red	1599	12	0	8.32	1.74
55	winequality-white	4898	12	0	5.88	0.89

Table 3.2: Properties of the datasets used in regression experiments.

3.1 Fuzzification and Defuzzification

Fuzzification

We proceed from the common setting of supervised learning as already introduced in Section 1.1 and assume data to be a set of instances exhibiting an attribute-value representation, which means that an instance is a vector

$$\boldsymbol{x} \in \mathbb{X} = \mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_m \ ,$$

where $\mathbb{X}_i$ is the domain of the i-th attribute A_i. In addition to these input attributes, every instance $\boldsymbol{x}$ is assigned to an output value $y \in \mathbb{Y}$. For now, we only consider two cases: either $\mathbb{Y} = \mathbb{R}$ (regression) or $\mathbb{Y} = \{0, 1\}$ (binary classification), where 0 indicates the negative and 1 indicates the positive class.

Each domain $\mathbb{X}_i$ is discretized by means of a fuzzy partition, that is, a set of fuzzy subsets

$$F_{i,j} : \mathbb{X}_i \to [0, 1] \qquad (j = 1, \ldots, n_i) \tag{3.1}$$

such that $\sum_{j=1}^{n_j} F_{i,j}(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathbb{X}_i$. The $F_{i,j}$ are often associated with linguistic labels such as "small" or "large", in which case they are also referred to as *fuzzy terms* (the identifier of an underlaying fuzzy concept).

To make fuzzy pattern trees amenable to numerical, ordinal, nominal or binary attributes, these attributes have to be "fuzzified" and discretized beforehand. The fuzzy partitions are either provided by an expert, who uses his expert knowledge to define comprehensible and reasonable fuzzy sets for each attribute domain. If an expert is not available, there are generic methods to infer fuzzy partitions from data.

Discretization of data by means of an automatically generated fuzzy partition is far from trivial. First, since the fuzzy partition is the main building block for every kind of fuzzy logic-related machine learning method, the discretization should suite the method it is used in and enable it to capture the dependencies and structure in the data. Second, and equally important, the discretization must be comprehensible by an expert of the application field. Otherwise, one of the main benefits of fuzzy systems, namely their interpretability, would be lost.

In the following, one way of generating a fuzzy partition is presented, which will also be used in the experiments in the following sections. To accommodate our supervised setting, we will explicitly take the output

We discretize a domain $\mathbb{X}_i$ using three fuzzy sets $F_{i,1}, F_{i,2}, F_{i,3}$ associated, respectively, with the terms "low", "medium" and "high". The first and the third fuzzy set are defined as

$$F_{i,1}(x) = \begin{cases} 1 & x < min \\ 0 & x > max \\ 1 - \frac{x-min}{max-min} & otherwise \end{cases},$$

$$F_{i,3}(x) = \begin{cases} 1 & x > max \\ 0 & x < min \\ \frac{x-min}{max-min} & otherwise \end{cases},$$

with min and max being the minimum and the maximum value of the attribute domain. It is clear that these fuzzy sets can capture two types of influence of an attribute, namely a positive and a negative one: If the value of a numeric attribute increases, the membership of the "high"-term of that attribute also increases (positive influence), whereas the membership of the "low"-term decreases (negative influence). Due to the monotonicity of all aggregation operators (see Section 2.3), this furthermore implies the same kind of influence towards the output of the whole tree.

Apart from monotone dependencies, it is of course possible that a non-extreme attribute value is "preferred" by a class or coincides with the maximum numerical output value. The fuzzy set $F_{i,2}$ is meant to capture dependencies of this type. It is defined as a *triangular fuzzy set* with center c:

$$F_{i,2}(x) = \begin{cases} 0 & x \leq min \\ \frac{x-min}{c-min} & min < x \leq c \\ 1 - \frac{x-c}{max-c} & c < x < max \\ 0 & x \geq max \end{cases} \tag{3.2}$$

The parameter c is determined so as to maximize the absolute (Pearson) correlation between the membership degrees of the attribute values in $F_{i,2}$ and the corresponding binary class information on the training data. In case the correlation is negative, $F_{i,2}$ is replaced by its negation $1 - F_{i,2}$.

Nominal attributes are modeled as degenerated fuzzy sets: For each value v of the attribute, a fuzzy set with membership function

$$F_{i,v}(x) = \begin{cases} 1 & x = v \\ 0 & otherwise \end{cases}$$

is introduced. Full membership is only given to elements which equal v. Zero membership is assigned to all others.

Defuzzification

The defuzzification procedure for FPT models is more complex. It depends on the learning task at hand. Note that the range of output values of a tree is the unit interval, i.e., the output variable assumes values in the range $[0, 1]$. One can think of the output value as the degree of membership of a fuzzy subset G of the underlying domain $\mathbb{Y}$.

An FPT *model* consists of one or more trees, together with a *coding scheme* for the output variable. This scheme basically constitutes two steps, a *de-/composition* step and a *de-/fuzzification* step.

Before training, the original problem may be decomposed via decomposition schemes like one-vs-rest [3] or others, to either create several binary classification or regression tasks. For each of these tasks, then a fuzzification is performed and a tree is trained. After training, when a query instance has to be predicted, each tree provides a predicted membership. These memberships are defuzzified and composed to become a final prediction in $\mathbb{Y}$. In the following, three prominent examples are given.

Multi-class Classification

In multi-class classification, each instance is associated with a class label

$$y \in \mathbb{Y} = \{y_1, y_2, \ldots, y_k\} \ .$$

A pattern tree *classifier* is a collection of pattern trees

$$\{M_1, \ldots, M_k\} \ ,$$

where M_i is the tree associated with class $y_i \in \mathbb{Y}$ following a one-versus-rest scheme. Thus, the original k-class classification problem is decomposed into k binary problems, one for each class $y_j \in \mathbb{Y}$. In the j-th problem, y_j is considered as the positive class (for which the sought model prediction is 1) and $\mathbb{Y} \setminus \{y_j\}$ as the negative class (for which the target is 0). As a special case, for binary classification ($k = 2$) we only need one tree.

Given a new instance $\boldsymbol{x}$ to be classified, a prediction is made in favor of the class whose tree produces the highest score:

$$\widehat{y} = \underset{y_i \in \mathbb{Y}}{\operatorname{argmax}} \; M_i(\boldsymbol{x}) \tag{3.3}$$

A single tree M_i can also be seen as a "fuzzy" selector of its class, hence the name fuzzy pattern tree. Like a regular expression, a pattern tree selects instances belonging to its class, albeit in a fuzzy way. Eventually, the class is determined by the pattern tree that is most "confident" of being representative of the instance.

Regression

In the case of regression, i.e. $\mathbb{Y} = \mathbb{R}$, only one tree is needed. Therefore, no de-/composition step is necessary. However, the defuzzification step is necessary in cases where the output space $\mathbb{Y}$ does not coincide with the unit interval. For example, if $\mathbb{Y}$ is an arbitrary interval $[a, b]$, i.e., if the original output variable is lower-bounded by a and upper-bounded by b, then the membership function could be given by a simple linear scaling

$$G : y \mapsto \frac{y - a}{b - a} \ . \tag{3.4}$$

Thus, the corresponding fuzzy set could be interpreted as a model of the linguistic term "large". Likewise, if the original output is unbounded, a possible re-scaling is

$$G : y \mapsto \frac{1}{1 + \exp(-\alpha y)} \ . \tag{3.5}$$

More generally, G can be any fuzzy subset of $\mathbb{Y}$. In order to map the predicted membership values back into $\mathbb{Y}$, we need to be able to compute $G^{-1} : [0, 1] \to \mathbb{Y}$, which is the inverse of G.

Fuzzy Systems

Fuzzy pattern trees can also be considered as an interesting approach to fuzzy systems modeling [99] and, in this regard, as an alternative to conventional fuzzy rule models. Generalizing the simple regression scheme, we assume the domain $\mathbb{Y}$ to be discretized by means of a fuzzy partition consisting of fuzzy sets $G_1, G_2, \ldots, G_k$. Then, a single pattern tree model implementing a mapping

$$M_i : (x_1, x_2, \ldots, x_n) \mapsto G_i(y) \ ,$$

could be constructed for each of these fuzzy sets. Given an instance $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, a corresponding ensemble of pattern trees produces a fuzzy description of the output in the form of a vector

$$\begin{aligned} M(\boldsymbol{x}) &= (M_1(\boldsymbol{x}), \ldots, M_k(\boldsymbol{x})) \\ &= (z_1, \ldots, z_k) \in [0,1]^k \ . \end{aligned}$$

The simple decoding $(G^{-1}(M(\boldsymbol{x})))$ then has to be replaced by a defuzzification step, which could be accomplished, for example, by

$$\hat{y} = \operatorname*{argmin}_{y \in \mathbb{Y}} \|(z_1, \ldots, z_k) - (G_1(y), \ldots, G_k(y))\| \ .$$

The i-th pattern tree can be considered as a model describing conditions (on the input attributes) under which the output variable Y is in G_i, e.g., under which "Y is medium" or "Y is large". Interestingly, compared to rule-based fuzzy systems, the "direction" of modeling is thus reversed. In fact, in rule-based systems, one typically starts with the rule antecedents, i.e., one fixes conditions on the input attributes and then assigns a suitable (fuzzy) value for the output variable. In pattern trees, it is just the other way around: First, the (fuzzy) output values are fixed, and then conditions on the input attributes are specified.

Finally, it is worth mentioning that pattern tree-based fuzzy systems of the above kind are in a sense generalizations of (standard) rule-based systems, in which rule antecedents are combined by means of a t-norm and the rules themselves by means of a t-conorm. In fact, given a system of that kind, all rules with the same consequent part "Y is G_i" can be collected and represented as a three-level pattern tree: The highest level consists of a node labeled with a t-conorm, while the mid level consists of nodes labeled with a t-norm, one for each rule, aggregating the corresponding rule antecedents. The lowest level represents the fuzzy sets as usual. Strictly speaking, a tree of that kind is not a proper pattern tree, since it is not binary. However, noticing that both t-norms and t-conorms are associative operators, it can easily be "binarized".

3.2 Optimization of CI Parameters

As a prerequisite to the actual learning algorithm it is necessary to be able to optimize the parameters of all parameterized operators in an efficient way[1]. Huang et al. already

[1]Parts of this section are submitted to be published in [94].

provided a solution for the WA and OWA operators. Since we newly introduced the CI operator, in the following we give a solution that is similiar to the existing one for WA and OWA.

Optimizing the parameters of a CI operator constitues a constrained optimization problem. We try to find those α and β that minimize the squared error between the output of the node and a target vector $\boldsymbol{z}$, given the two input vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, denoting the outputs of the nodes' children. We propose an optimization of the parameters in two steps. First, find the optimum parameters for arbitrary values of α and β, hence not considering the constraints

$$0 \leq \alpha, \beta \leq 1 \ .$$

Second, trim the parameters to reside in $[0, 1]$.

Looking at (2.5), we have to take care about two cases: $u \leq v$ and $u > v$. To this end, we split the training instances into two groups $\mathcal{T}^{\leq}$ and $\mathcal{T}^{>}$ accordingly. Because each of the parameters only appears in one of the cases, they can be optimized independently of each other by using the instances of their respective group. In fact, the following calculations are very similar in both cases. Hence, we will only present the first one (α). For the sake of an easier notation, let $\boldsymbol{u}$ and $\boldsymbol{v}$ now be restricted to the instances in $\mathcal{T}^{>}$. Starting from (2.5), we first proceed with a simple transformation:

$$\begin{aligned} \alpha\boldsymbol{u} + (1 - \alpha')\boldsymbol{v} &= \boldsymbol{z} \\ \alpha(\boldsymbol{u} - \boldsymbol{v}) &= \boldsymbol{z} - \boldsymbol{v} \end{aligned}$$

Again to ease notation, let $\boldsymbol{p} = \boldsymbol{u} - \boldsymbol{v}$ and let $\boldsymbol{q} = \boldsymbol{z} - \boldsymbol{v}$. Then, minimizing the squared loss, we get:

$$
\begin{aligned}
\sum_{i=1}^{|\mathfrak{I}^{>}|}(\alpha' q_i - p_i)^2 &\rightarrow \min \\
\sum_{i=1}^{|\mathfrak{I}^{>}|} 2(\alpha' q_i - p_i) q_i &= 0 \\
\alpha' \sum_{i=1}^{|\mathfrak{I}^{>}|} q_i^2 - \sum_{i=1}^{|\mathfrak{I}^{>}|} p_i q_i &= 0 \\
\alpha' &= \frac{\sum_{i=1}^{|\mathfrak{I}^{>}|} p_i q_i}{\sum_{i=1}^{|\mathfrak{I}^{>}|} q_i^2}
\end{aligned}
$$

To ensure, that α is a valid parameter to the CI operator, we apply the following update:

$$\alpha = \min\{1, \max\{0, \alpha'\}\} \tag{3.6}$$

We are aware of the fact, that this update might yield a suboptimal solution for the unconstrained case. However, dealing with the squared error ensures the following. If the minimum does not reside inside the unit interval using (3.6) still preserves the constrained minimum.

In Section 4.1 we present an experimental study that evaluates the CI operator compared to WA and OWA.

3.3 Existing Learning Algorithms

3.3.1 Huang, Gedeon & Nikravesh

Following the original proposal of [49], pattern trees are built one by one, independently of each other. For each class, the induction method performs the following main steps:

1. initialize with primitive pattern trees
2. filter candidates by evaluation of their similarity to the target class
3. check stopping criterion
4. recombine candidates using fuzzy operators

5. loop at step 2

We can call this approach the *bottom-up approach* (PTBU) because new nodes are introduced at the top (the root) of the trees. This way, a tree grows from the bottom up to the top. Figure 3.1 contains the algorithm in detail.

During initialization, three sets are created. $\mathbf{P}$ denotes the set of all so-called *primitive pattern trees*. These are one-node trees, representing a single fuzzy set $F_{i,j}$. $\mathbf{S}$, the so-called *slave set* is also initialized, containing all elements of $\mathbf{P}$.

The first set of candidate trees $\mathbf{C}^0$ is a subset of $\mathbf{P}$. It contains the B best candidate trees. $\mathbf{C}^0$ contains the trees with the lowest average error (line 4). The average error of a tree M is measured in terms of the predefined loss function $\mathcal{L}$. It measures the error between the predicted membership $M(\boldsymbol{x}^{(i)})$ and the actual ones given by $G(y^{(i)})$ (cf. Section 3.1) for all training instances $(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{T}$.

Let Ψ be the set of available aggregation operators. Then, starting from line ten, all possible combinations of an aggregation operator $\psi \in \Psi$, a tree $M_1 \in \mathbf{C}^0$ and a tree $M_2 \in \mathbf{S}$ are used to create new candidates. Figure 3.2 illustrates the schema of a resulting tree. It is created by taking the operator ψ as root node which aggregates the two selected trees. In case the chosen operator is parameterized by a parameter θ, this parameter is optimized in line 20. The optimization method depends on the operator as it was described in Section 2.2.1.

In line 26, all candidate trees again are evaluated according to the loss function $\mathcal{L}$. Only the B best trees remain in $\mathbf{C}^t$. The algorithm stops if it has reached the maximum number of iterations t_{max} or the algorithm could not find model that improves the best model of the previous iteration (line 30).

Finally, the candidate tree with the lowest empirical error is returned.

3.3.2 Yu, Fober and Hüllermeier

As mentioned before, the model class of fuzzy pattern trees has been introduced simultaneously also by Yu et al. under the name fuzzy operator tree (FOT). Basically, the only difference refers to the way the trees are created. The approach of Yu et al. was mainly motivated by the general need to define *utility functions* for various applications. Especially in economics and *game theory*, utility functions play an important

Bottom-up Algorithm

1: {initializing sets}
2: $\mathbf{P} = \{..., F_{i,j}, ... | i = 1, ..., n; j = 1, ..., m\}$
3: $\mathbf{S} = \mathbf{P}$
4: $\mathbf{C}^0 = \underset{F \in \mathbf{P}}{\overset{B}{\operatorname{argmin}}}[\sum_{(\boldsymbol{x},y)\in\mathcal{T}} \mathcal{L}(y, F(\boldsymbol{x}))]$
5: {initializing termination criteria}
6: $t_{max} = 5$
7: $t = 0$
8: {starting iterative induction}
9: **repeat**
10: $t = t + 1$
11: $\mathbf{C}^t = \mathbf{C}^{t-1}$
12: {first loop on each candidate in $\mathbf{C}^{t-1}$}
13: **for all** $C_i^{t-1} \in \mathbf{C}^{t-1}$ **do**
14: {second loop on each candidate in $\mathbf{S}$, without C_i^{t-1}}
15: **for all** $S_j \in \mathbf{S} \setminus C_i^{t-1}$ **do**
16: {loop on each available operator ψ}
17: **for all** $\psi_\theta \in \Psi$ **do**
18: {optimize parameter θ}
19: $\theta = \text{optimize}_\psi(C_i^{t-1}(\mathcal{T}), S_j(\mathcal{T}), (y_l)_{l=1}^{|\mathcal{T}|})$
20: {construct new candidate}
21: $\mathbf{C}^t = \mathbf{C}^t \cup \text{new}(\psi_\theta, C_i^{t-1}, S_j)$
22: **end for**
23: **end for**
24: **end for**
25: $\forall M \in \mathbf{C}^t : l_M^t = \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \mathcal{L}(y, M(\boldsymbol{x}))$
26: $\mathbf{C}^t = \underset{M \in \mathbf{C}^t}{\overset{B}{\operatorname{argmin}}}[l_M^t]$
27: $\mathbf{S} = \mathbf{S} \cup \mathbf{C}^t$
28: $l_{min}^t = \min_{M \in \mathbf{C}^t}[l_M^t]$
29: **until** $t == t_{max}$ *or* $l_{min}^t \geq l_{min}^{t-1}$
30: **return** $\underset{M \in \mathbf{C}^t}{\operatorname{argmin}}[l_M]$

Figure 3.1: Algorithm by Huang, Gedeon and Nikravesh

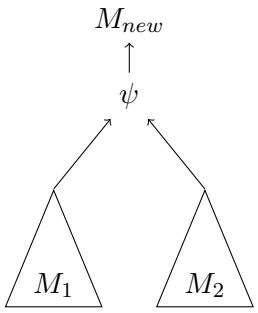


Figure 3.2: Creating a new candidate tree in a bottom-up manner.

role [70]. In these fields, utility functions usually are defined by an expert in order to evaluate the utility of different alternatives.

Keeping this in mind, it was proposed to create the trees in a two step procedure.

1. An expert is asked to determine the structure of the tree, i.e., he determines the hierarchical structure, the basic criteria (leaf nodes) and the way these basic criteria are aggregated into an overall evaluation by choosing the operator class (t-norm, t-conorm or averaging operator) for each inner node.

   In order to provide flexibility for the operator classes of t-norms and t-conorms, the authors propose to use the Dubois & Prade families (2.3). They are parameterized by one real-valued parameter, which takes values from the unit interval. In addition to this and just like Huang et al. the WA and the OWA operators may be used.

2. All parameters of the tree (those of operators and fuzzy sets) are collected into one parameter vector $\boldsymbol{\theta}$. This parameter vector is then optimized in order to best fit a set of example evaluations, i.e., a training set of instances $\mathcal{T}$.

Let M be the model the expert has determined. In order to accomplish the second step, it was proposed to use an *evolutionary algorithm* (see Section 3.7.1) to find the best parameter vector $\boldsymbol{\theta}^*$. The concrete optimization task is

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \mathcal{L}(M_{\boldsymbol{\theta}}(\boldsymbol{x}), y) \ , \tag{3.7}$$

where $\mathcal{L}$ is a suitable loss function.

The approach by Yu et al. is interesting for the reason of its hybrid nature, combining expert knowledge and machine learning techniques. Nevertheless, for many applications, no expert is available to reasonably accomplish the first step of the procedure. Because of this, we subsequently focus on learning the whole tree, including both its structure and parameterization. We take the method of Huang et al. as a point of departure and present several analyses, extensions and completely new algorithms.

3.4 Study on Surrogate Loss Functions

So far, we discussed about some loss function $\mathcal{L}$, which we try to minimize with the help of induction algorithms[1]. Depending on the learning task, indeed, this loss function can be varied. Nevertheless, if for example the learning task is classification and our *target loss* is classification accuracy, it does not mean, that we necessarily have to use accuracy during induction. Instead, it sometimes is beneficial to use a different loss function, a so-called *surrogate*. A prominent example is classification error. Although it is used as final evaluation measure for most classification tasks, it exhibits at least two weaknesses. First, it is not differentiable in every point and even worse, it includes large constant regions. Like it is the case for many machine learning algorithms also the following ones comprise an iterative optimization scheme. Hence, it is preferable to either be able to calculate a non-zero gradient or difference quotient which guides the search for an optimum during optimization.

From a theoretical point of view, it is difficult to anticipate which surrogate function – practically there are many possibilities – might be especially suitable. Since Huang et al. do not give any clear recommendation either, we tried to answer this question experimentally. More specifically, we conducted a number of experiments in which we compared loss functions of different type in terms of classification performance.

Apart from *root mean squared error* (RMSE) and a Jaccard-based error measure (JacE), which have also been used in [49], three more measures have been tested. These are the *mean absolute error* (MAE) and a *mean sigmoidal error* (MSigE) and standard

[1] Parts of this section were already published in [93].

classification error (ClassE).

$$\mathrm{RMSE}(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - b_i)^2} \tag{3.8}$$

$$\mathrm{JacE}(\boldsymbol{a}, \boldsymbol{b}) = 1 - \frac{\sum_{i=1}^{n} \min(a_i, b_i)}{\sum_{i=1}^{n} \max(a_i, b_i)} \tag{3.9}$$

$$\mathrm{MAE}(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{n} \sum_{i=1}^{n} |a_i - b_i| \tag{3.10}$$

$$\mathrm{MSigE}(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{n} \sum_{i=1}^{n} (1 + \exp(-8((a_i - b_i) - 0.5)))^{-1}) \tag{3.11}$$

$$\mathrm{ClassE}(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 0 & \mathrm{sgn}(a_i - 0.5) = \mathrm{sgn}(b_i - 0.5) \\ 1 & \text{otherwise} \end{cases} \tag{3.12}$$

where n denotes the length of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

As a potential disadvantage of the RMSE measure, note that the quadratic error function, like all convex losses, is quite sensitive toward outliers: If an instance cannot be classified correctly, it may yield a disproportionately high error, thereby devaluing a model with otherwise strong performance. It was mainly for this reason that we also tried the measure (3.11) with a sigmoid-shaped error function. The idea here is to pay special emphasis on errors around 1/2, since this point is critical for the correctness of the classification rule (3.3), whereas less distinction is needed for errors close to 0 or close to 1. A sigmoid-shaped function, which is a smooth version of the step function jumping from 0 to 1 at 1/2, does obviously comply with these requirements, so that MSigE appears advantageous from this point of view. On the other hand, there are also arguments against the expectation that MSigE may outperform RMSE:

- In contrast to problems like regression, the squared error is upper-bounded (namely by 1), which means that the sensitivity toward outliers is less severe.
- One should note that a devaluation caused by a true outlier will probably apply to all candidate models in more or less the same way. In other words, it is unlikely to change the relative performance, and hence the selection, of a pattern tree in comparison to other candidates.
- A sigmoid-shaped loss function is arguably less suitable for handling pseudo-outliers, i.e., instances for which the current model has a high error even though

they are not true outliers: A small improvement on such instances, moving their score a bit to the correct direction, will hardly be rewarded, thereby preventing the right model adaptation.

As can be seen, there are arguments both in favor and against a loss function like (3.11).

Regarding the standard classification error (ClassE), there is at least one good reason for, and one good reason against it. Since, minimizing classification error is the ultimate goal of classification, using ClassE directly evaluates each candidate tree on the actual measure, we are interested in. There is no surrogate function involved. However, ClassE comes with a major disadvantage, namely it is neither convex nor continuous. In fact, it is a step function. These steps can cause a premature stop of iterative algorithms like the one of Huang et al..

Experiments

In order to evaluate the suitability of each surrogate loss function for the PTBU algorithm, we compared five variants of PTBU. All parameters of these variants have been identical[1] except for the choice of loss function. The complete results can be seen in Appendix A.

Loss	ClassE	JacE	MAE	MSigE	RMSE
Avg. rank	2.56	3.83	3.37	3.36	1.86
Num. wins	11	1	2	2	21

Table 3.3: Average rank and number of wins for each surrogate loss function.

Table 3.3 summarizes the results in terms of the average rank each variant achieves on our 40 benchmark datasets. Additionally, also the number of wins (ranked first place) of each loss is shown. The results are clearly in favor of RMSE that performed best on 21 of the 40 datasets (while the second-best, ClassE, was only 11 times the winner). Our explanation for the strong performance of RMSE is the convex shape of the quadratic error function. In particular, convexity implies that larger deviations from the target prediction (1 for positive and 0 for negative examples) are punished disproportionately high, whereas small deviations are less critical. In connection with

[1] Parameter setting: $t_{max} = 5$, $B = 5$

the "argmax" classification rule (3.3), this appears to be quite reasonable. This classification rule is indeed robust toward small deviations from the target prediction. In fact, it will predict the correct class as long as the errors are all smaller than 1/2.

The second best performing measure is ClassE. It seems, that the disadvantage of ClassE being a step function predominates.

Anyway, our experiments have clearly shown that, regarding accuracy, our recommendation should be to use the RMSE measure (3.8). Specifically, we shall do so in the experimental studies presented in all following sections.

3.5 Accelerating the Bottom-up Approach

In [49], it was shown that the bottom-up approach is quite competitive to many state-of-the-art classification methods in terms of predictive accuracy. To further improve the usability of the approach for real world applications, a crucial property is a small runtime during the training and the test phase. Although testing (i.e., executing a tree for a specific instance) is fast, training an FPT can be time consuming. In order to improve this, three heuristics have been developed and will be introduced in the remainder of this section. All three heuristics aim at reducing the search space (hypothesis space) without significantly worsening the predictive accuracy of the models. An experimental study will demonstrate their effectiveness.

3.5.1 Sparse Search

The first one is based on a rather simple idea, but as will be shown in the experimental section, it has a strong positive effect on the runtime. In line 22 of the bottom-up algorithm, a new candidate tree is created for every combination of an operator and a candidate tree and a slave tree. This results in $|\mathbf{C}^{t-1}| \cdot (|\mathbf{S}|) \cdot |\Psi|$ many possible new candidate trees to fill up $\mathbf{C}^t$ in the first place. With the sparse search heuristic, we reduce the number of trees included in $\mathbf{C}^t$ in the following way. A candidate $M_{new} = new(\psi_\theta, C_i^{t-1}, S_j)$ is only included into $\mathbf{C}^t$ if it holds that

$$l_{M_{new}} < min(l_{C_i^{t-1}}, l_{S_j}) \text{ , where} \tag{3.13}$$

$$l_M = \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \mathcal{L}(y, M(\boldsymbol{x})) \tag{3.14}$$

In words, a new candidate is only considered to be a valid element in the new candidate set C^t, if it has a lower error on the training data in comparison to the minimum of both of its children. At first sight, this seems to be an obvious condition. However, by enforcing this condition, the search algorithm becomes more greedy and myopic. In fact, the algorithm is only able to look one step ahead. Without the above condition, it is possible to keep a candidate, which not directly (after one iteration) was found to be helpful in combination with the current candidates, but still could be later on.

3.5.2 Dynamic Operator Exclusion

The second heuristic to introduce again aims at reducing the number of candidates created in line 22. This time however, we make use of the order of operators (see Section 2.2). According to this order, it holds that:

$$\begin{gathered}\forall\ \alpha,\beta: \\ LUK \preceq EIN \preceq ALG \preceq MIN \\ \preceq CI_{\alpha,\beta} \preceq \\ COMAX \preceq COALG \preceq COEIN \preceq COLUK\end{gathered}$$

As described in Section 2.2.1, the CI operator is able to express the minimum and the maximum operator as special cases for $(\alpha,\beta) = (0,0)$ and $(\alpha,\beta) = (1,1)$, respectively. Additionally, optimizing α and β can be accomplished efficiently. Therefore, we will utilize α and β, more precisely $(\alpha+\beta)$ to decide, whether or not we create candidates using t-norms or t-conorms.

Given two candidates C_i^{t-1} and S_j, the procedure consists of three steps:

1. Create a new candidate using the CI operator. Doing this includes the calculation of the optimal parameters α^* and β^*.

2. Only if $\alpha^* + \beta^* \geq \tau$, the CI aggregation is considered similar to a conjunctive aggregation, and therefore further candidates are created using t-norms (ALG, EIN, LUK).

3. Only if $\alpha^* + \beta^* \leq 2 - \tau$, the CI aggregation is considered similar to a disjunctive aggregation, and therefore further candidates are created using t-conorms ($COALG$, $COEIN$, $COLUK$).

According to this procedure, the creation of candidates using t-norms is omitted if the CI parameters suggest the best aggregation to be "or-like". Likewise, t-conorms are ommited if the parameters suggest the best aggregation to be "and-like". A reasonable choice of the additional parameter τ seems to be 0.2, as will be seen in the experiments.

3.5.3 Limited Candidate History

In line 28 of the bottom-up algorithm, the slave set $\mathbf{S}$ is updated by adding the selected candidates for the current iteration. $\mathbf{S}$ therefore contains the history of candidate trees, which have at least once been among the B best candidates in a previous iteration. This forces $\mathbf{S}$ to grow in each iteration, which also increases the number of possible combinations of new candidates, what in turn increases the runtime in each iteration.

$\mathbf{S}$ can also be considered as a sort of candidate set history. Limiting this history to a fixed number of iterations, say k, restricts the growth of $\mathbf{S}$ and therefore the runtime per iteration. Hence, after adding another candidate set to $\mathbf{S}$ in iteration $t > k$, the implementation of this last heuristic removes $\mathbf{C}^{t-k}$ from $\mathbf{S}$.

Again, we refer to the upcoming experimental section for a reasonable choice of k, which will turn out to be 5.

3.5.4 Experiments on Heuristics

In this section, we will empirically show that the heuristics introduced in the last sections yield runtime improvements without compromising the predictive accuracy of the learned models. To accomplish this task, we will conduct one experiment per heuristic.

Sparse Search

The first experiment compares the original bottom-up algorithm (PTBU) with a variant, which incorporates the sparse search heuristic (PTBU-S). Table 3.4 reports the results of the cross-validation procedure on the first 40 classification datasets in terms of predictive accuracy. Table 3.5 reports the runtime results accordingly.

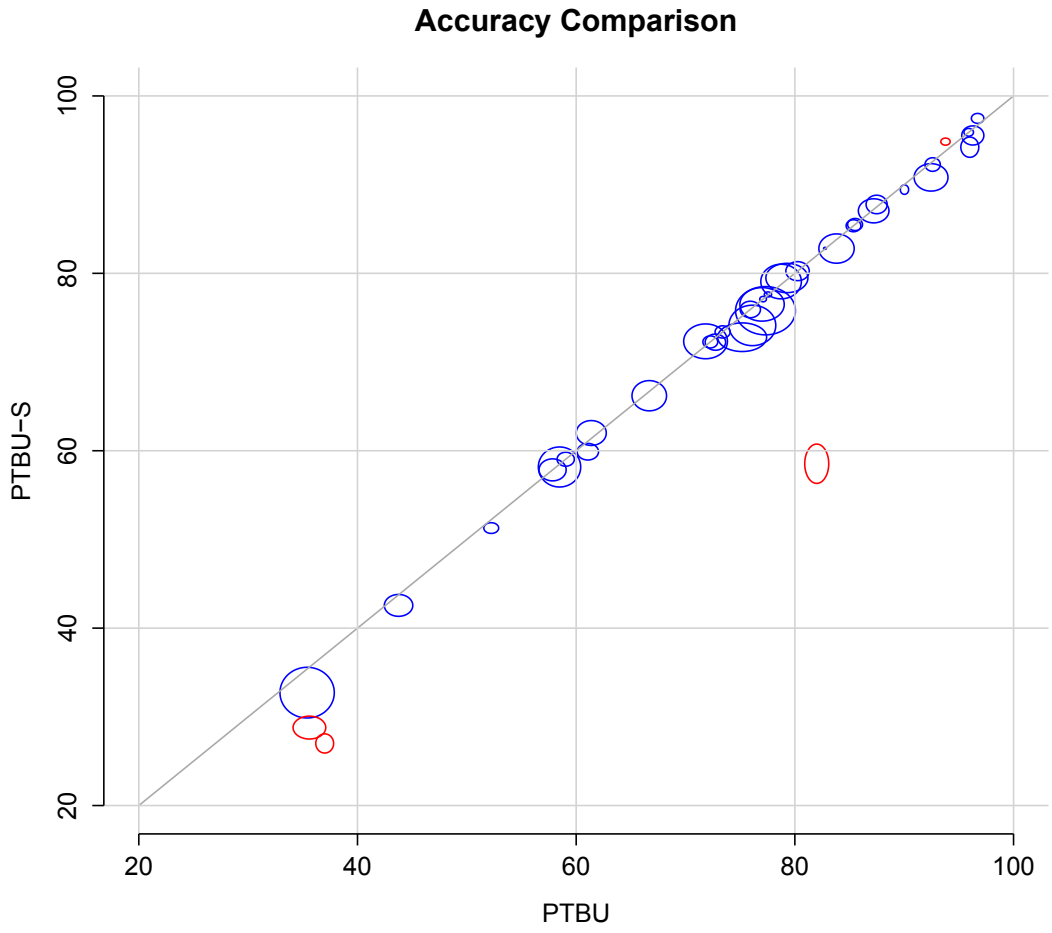


Figure 3.3: Comparison of the accuracy of PTBU and PTBU-S on 40 datasets (one ellipsoid per dataset). If both methods have the same mean accuracy, the center of the ellipsoids lay on the dashed diagonal. Ellipsoids depart from the diagonal indicate a difference in accuracy. The dimensions of the ellipsoid are determined by the standard error of the mean estimate.

In terms of training runtime, PTBU-S clearly outperforms PTBU. It wins all 40 times and the average relative runtime decrease is 40.80%. However, comparing predictive accuracy is more difficult. We actually want to know, if applying the heuristic affects the predictive accuracy of the method seriously. Of course, in general there is always a trade-off between accuracy and runtime.

Looking at the win-loss statistic, PTBU-S wins 9 times but looses 21 times with another 10 ties. Although, PTBU seems superior in this regard, Figure 3.3 shows the

actual difference between both methods. Each ellipsoid represents a dataset. The x-axis determines the accuracy of PTBU, whereas the y-axis determines the accuracy of PTBU-S. The centers of the ellipsoids are located where the corresponding accuracies of the two methods meet. The dimensions of the ellipsoids (height and width) refer to the standard error of the mean estimates. Roughly speaking, this plot illustrates that the differences between PTBU and PTBU-S for most datasets are minor except for four, which are marked in red.

Dynamic Operator Exclusion

The second experiment aims at measuring the effect of the dynamic operator exclusion heuristic (PTBU-DOE). To this end, we first compare several different values of τ regarding predictive accuracy and training time. Figure 3.4 illustrate the results. The precise accuracy and training time values can be found in the Appendix B. For now, the exact values are not important. More important, however, are the trends, which can be seen in the figure. On the one hand, accuracy does not seem to be affected by τ, at least no clear up- or downtrend is visible. On the other hand, however, training runtime is effected significantly. Note, that for runtime, the figure provides a logarithmic scale in order to be able to differentiate datasets with a very low runtime in the figure. As τ increases, runtime does too.

These two findings provide an empirical evidence, that the proposed heuristic indeed works as expected. Next, we will fix τ to be 0.1 and compare PTBU-DOE and PTBU. Tables 3.4 and 3.5 show the results and as the previous analysis already suggested, in terms of accuracy, we find PTBU-DOE winning 18 times, loosing 19 times with another 3 ties. Contrary to this balanced accuracy comparison, the comparison in terms of runtime clearly favors PTBU-DOE with 39 wins and one loss. On average, runtime declines by 44.3% relative to PTBU.

Limited Candidate History

The third heuristic is evaluated in the same manner as the second. First, a reasonable value for the history size parameter k will be chosen empirically. PTBU-LCH then denotes the PTBU variant, which is using the limited candidate history heuristic (LCH). Then we will again compare to PTBU. In contrast to the previous experiment, we set

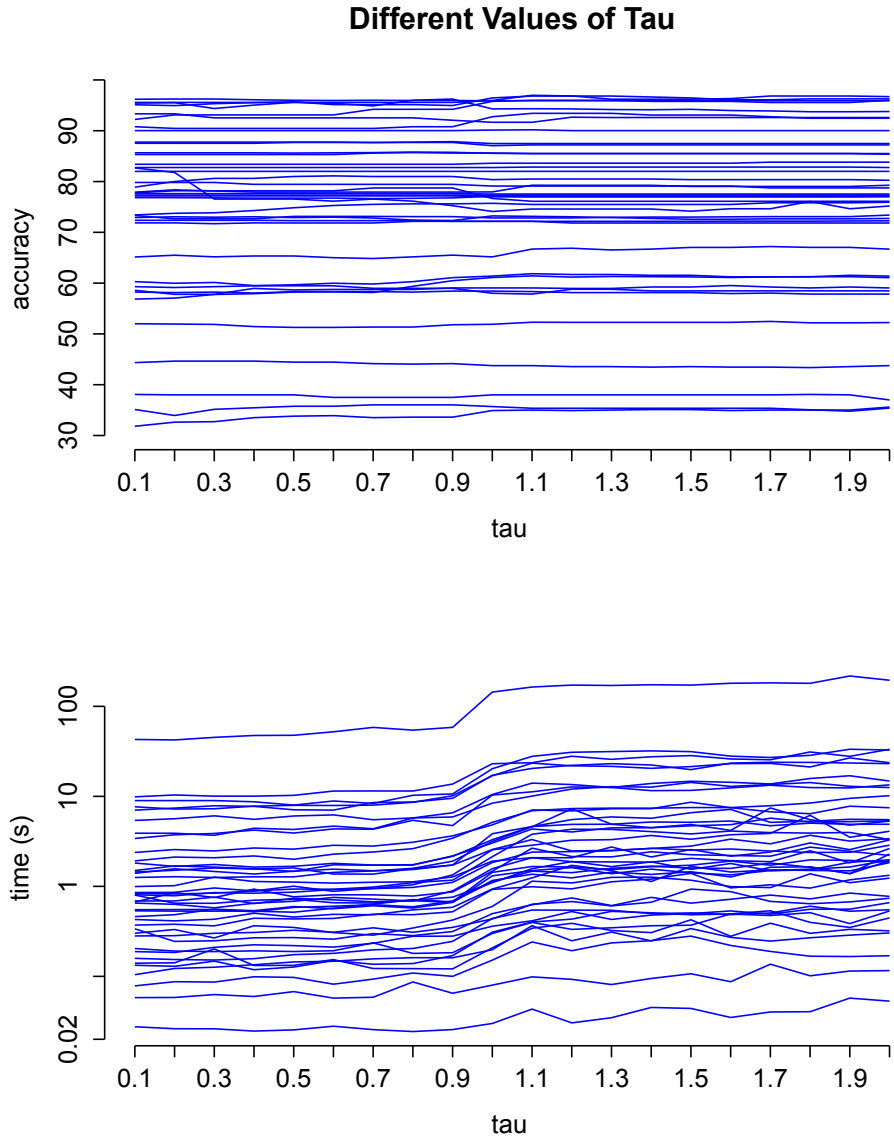


Figure 3.4: Predictive accuracy and training runtime of PTBU-DOE for different values of τ. Each line represents one of the 40 classification datasets.

the parameter $t_{max} = 10$, because the effect of LHC mainly prevails in later iterations. This conclusion can be drawn on the basis of the results shown in Figure 3.5.

While runtime steadily increases with an increase of k, starting from $k = 5$, accuracy does not increase anymore for most datasets. For many datasets, accuracy is even almost constant for all values of k. Nevertheless, we choose $k = 5$ for a comparison to PTBU, which corresponds to PTBU-LCH with $k = 10$. The precise accuracy and runtime values of Figure 3.5 can be found in Appendix C.

Comparing PTBU and PTBU-LCH with $k = 5$ in terms of accuracy, PTBU-LCH wins 16 times, looses 18 times with another 6 ties. Again, this is a balanced result and we can consider both variants as similarly strong. However, what Figure 3.5 already suggested, the runtime of PTBU-LCH is significantly smaller. With 39 wins, using LCH decreases the training time by 40.43%. The details of this comparison are also shown in Tables 3.4 and 3.5.

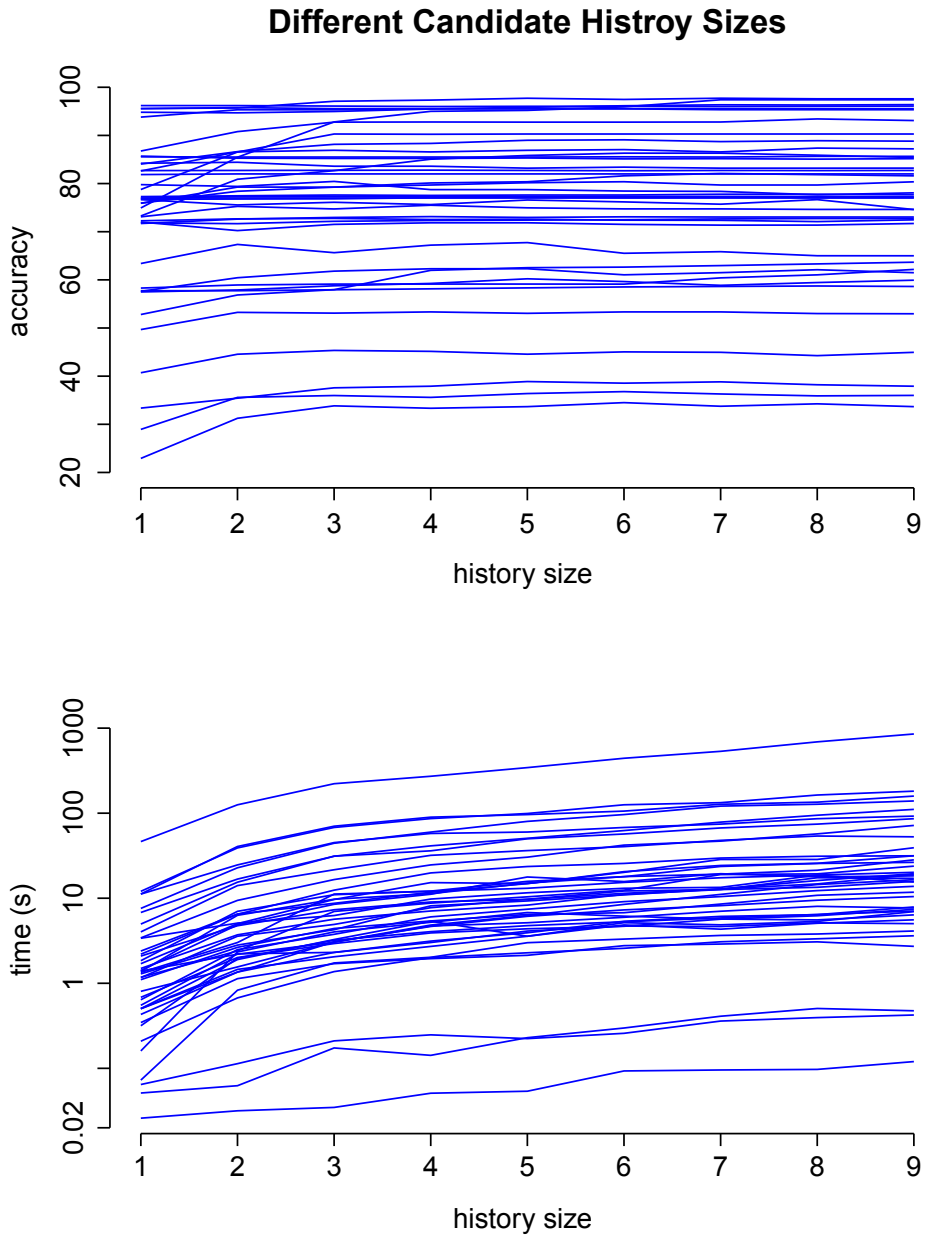


Figure 3.5: Predictive accuracy and training runtime of PTBU-LCH for different values of k. Each line represents one of the 40 classification datasets.

	$t_{max} = 5$			$t_{max} = 10$	
No.	PTBU	PTBU-S	PTBU-DOE	PTBU	PTBU-LCH
1	77.54 ±1.84	77.62 ±1.56	77.62 ±1.56	77.70 ±1.64	77.62 ±1.56
2	78.74 ±10.16	79.07 ±10.69	79.81 ±11.35	78.07 ±11.03	78.74 ±10.16
3	37.00 ±4.47	27.00 ±5.92	38.08 ±4.03	33.67 ±4.58	33.67 ±5.56
4	83.81 ±8.85	82.79 ±9.05	83.39 ±8.81	83.21 ±9.44	83.20 ±9.49
5	96.71 ±3.13	97.47 ±3.06	95.57 ±2.85	97.60 ±2.93	97.72 ±2.95
6	85.51 ±3.73	85.51 ±3.73	85.65 ±3.76	85.65 ±3.77	85.51 ±3.73
7	93.78 ±2.35	94.85 ±2.19	95.44 ±2.23	96.39 ±1.81	95.84 ±1.81
8	58.48 ±10.61	58.17 ±12.36	59.30 ±11.98	62.17 ±10.21	59.13 ±9.23
9	90.02 ±2.09	89.44 ±2.91	90.02 ±1.97	90.29 ±1.83	90.29 ±1.83
10	87.21 ±7.67	87.05 ±7.44	87.53 ±6.54	87.21 ±8.19	86.90 ±8.06
11	77.09 ±1.67	77.09 ±1.67	76.83 ±1.31	77.14 ±1.28	77.14 ±1.56
12	59.04 ±4.31	59.04 ±4.31	58.58 ±5.15	59.93 ±4.59	60.22 ±4.15
13	95.90 ±2.38	95.90 ±2.38	96.19 ±2.23	96.10 ±2.32	96.05 ±2.32
14	72.73 ±5.12	72.24 ±4.99	72.90 ±5.85	72.74 ±4.77	72.41 ±5.60
15	35.39 ±13.61	32.73 ±15.66	35.12 ±14.91	37.91 ±13.37	38.88 ±13.60
16	52.23 ±3.72	51.28 ±3.31	52.01 ±4.03	52.96 ±3.43	53.05 ±3.71
17	75.18 ±12.26	72.80 ±8.83	82.68 ±10.82	85.42 ±10.23	85.83 ±9.49
18	85.36 ±3.70	85.36 ±3.70	85.31 ±3.67	85.17 ±3.80	85.31 ±3.68
19	76.11 ±11.74	74.13 ±12.49	77.78 ±12.26	74.60 ±12.08	76.59 ±11.22
20	66.68 ±8.65	66.20 ±9.33	65.14 ±8.17	65.00 ±11.18	67.74 ±9.16
21	82.75 ±0.68	82.82 ±0.61	82.75 ±0.68	82.72 ±0.81	82.72 ±0.74
22	72.27 ±3.71	72.27 ±3.71	72.37 ±3.85	72.47 ±3.72	72.60 ±3.84
23	61.38 ±7.53	62.00 ±7.68	60.29 ±7.39	61.49 ±6.77	62.31 ±8.05
24	75.92 ±5.09	75.92 ±5.09	73.42 ±2.18	74.63 ±4.19	74.96 ±4.38
25	80.25 ±5.86	80.25 ±5.86	78.89 ±5.93	81.60 ±5.54	80.37 ±5.93
26	87.48 ±5.28	87.76 ±5.71	87.76 ±5.54	88.81 ±5.82	88.99 ±5.62
27	96.00 ±4.50	94.22 ±6.25	92.22 ±6.80	95.56 ±5.05	95.56 ±5.05
28	92.60 ±3.83	92.27 ±4.19	93.33 ±3.54	97.40 ±2.30	95.20 ±3.18
29	77.31 ±15.00	75.79 ±14.91	77.31 ±13.79	77.31 ±15.00	77.31 ±15.00
30	79.27 ±10.53	79.48 ±9.05	77.90 ±9.48	80.37 ±8.56	80.17 ±9.05
31	35.60 ±8.18	28.79 ±7.06	31.82 ±7.50	36.00 ±7.60	36.39 ±7.75
32	73.39 ±3.78	73.39 ±3.78	73.31 ±4.90	73.05 ±4.06	72.96 ±4.17
33	43.76 ±7.12	42.56 ±6.75	44.35 ±6.71	44.95 ±7.03	44.56 ±7.46
34	77.00 ±11.11	76.50 ±10.35	77.17 ±11.12	77.00 ±11.11	77.00 ±11.11
35	82.00 ±6.01	58.53 ±12.14	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	57.84 ±6.79	57.84 ±6.79	58.24 ±6.53	58.63 ±7.28	58.33 ±6.78
37	71.83 ±10.96	72.33 ±10.75	71.86 ±10.59	71.71 ±9.77	71.85 ±10.17
38	61.07 ±5.34	59.89 ±5.12	56.85 ±4.29	63.67 ±5.63	62.52 ±5.66
39	96.27 ±5.53	95.54 ±5.91	95.12 ±4.13	95.34 ±4.86	95.52 ±4.24
40	92.45 ±8.48	90.82 ±8.47	90.79 ±7.77	93.09 ±7.46	92.76 ±8.59

Table 3.4: Mean accuracy measures with standard deviation comparing PTBU, PTBU-S and PTBU-DOE with $\tau = 0.1$ and $t_{max} = 5$ on the left side. On the right side comparing PTBU and PTBU-LCH with $k = 5$ and $t_{max} = 10$.

	$t_{max} = 5$			$t_{max} = 10$	
No.	PTBU	PTBU-S	PTBU-DOE	PTBU	PTBU-LCH
1	7.14 ±1.93	3.53 ±0.36	2.38 ±0.39	56.95 ±8.78	36.29 ±7.96
2	1.65 ±0.33	0.65 ±0.09	11.26 ±17.33	6.71 ±3.13	3.57 ±0.95
3	2.26 ±0.76	1.28 ±0.45	0.81 ±0.08	18.88 ±2.66	13.09 ±2.41
4	2.65 ±0.78	1.22 ±0.38	0.83 ±0.23	16.04 ±10.12	6.26 ±2.89
5	0.31 ±0.07	0.16 ±0.04	0.14 ±0.07	4.18 ±1.84	3.00 ±1.20
6	4.88 ±1.35	2.66 ±0.43	1.25 ±0.20	32.96 ±4.77	14.88 ±3.62
7	172.90 ±35.73	40.45 ±6.99	42.86 ±8.59	876.40 ±81.95	343.28 ±42.63
8	19.80 ±1.73	7.80 ±1.70	7.03 ±0.69	114.34 ±12.81	50.67 ±6.00
9	1.66 ±0.19	1.03 ±0.10	0.65 ±0.14	26.05 ±4.64	15.79 ±3.65
10	0.47 ±0.11	0.35 ±0.11	0.16 ±0.05	3.28 ±1.05	2.28 ±0.73
11	0.70 ±0.27	0.43 ±0.08	0.34 ±0.17	4.95 ±0.63	3.58 ±1.38
12	0.49 ±0.07	0.28 ±0.05	0.19 ±0.06	5.35 ±1.41	4.11 ±1.64
13	1.18 ±0.11	0.77 ±0.10	0.46 ±0.08	17.54 ±4.77	9.73 ±2.56
14	1.95 ±0.18	1.16 ±0.22	0.67 ±0.08	14.89 ±2.54	11.58 ±1.71
15	0.72 ±0.25	0.34 ±0.07	0.28 ±0.09	8.09 ±2.07	3.92 ±0.86
16	10.68 ±0.92	8.89 ±4.95	3.41 ±1.08	92.14 ±11.30	60.02 ±8.86
17	0.35 ±0.08	0.16 ±0.05	0.20 ±0.10	6.74 ±2.01	4.76 ±1.88
18	4.50 ±0.27	3.07 ±1.10	1.50 ±0.27	30.87 ±7.88	15.05 ±3.23
19	4.40 ±2.55	1.53 ±0.20	1.39 ±0.13	18.91 ±3.11	9.29 ±1.42
20	29.73 ±4.60	8.95 ±1.34	8.97 ±1.44	156.87 ±28.67	80.07 ±12.70
21	19.68 ±1.59	11.95 ±1.85	7.66 ±1.75	162.70 ±21.25	96.25 ±6.24
22	12.51 ±1.31	4.57 ±0.55	3.89 ±0.51	84.90 ±20.12	30.39 ±6.35
23	2.05 ±0.18	1.24 ±0.14	0.99 ±0.28	18.19 ±2.75	17.76 ±12.54
24	0.20 ±0.04	0.16 ±0.05	0.10 ±0.04	2.59 ±0.96	2.13 ±1.02
25	0.81 ±0.10	0.60 ±0.14	0.30 ±0.09	6.60 ±1.62	4.37 ±0.79
26	5.00 ±0.53	1.68 ±0.30	1.82 ±0.32	22.09 ±3.41	10.39 ±3.04
27	0.41 ±0.07	0.13 ±0.05	0.13 ±0.06	8.12 ±2.79	5.15 ±1.56
28	0.88 ±0.10	0.57 ±0.11	0.37 ±0.06	6.37 ±1.10	6.79 ±1.05
29	0.09 ±0.06	0.05 ±0.04	0.06 ±0.04	0.51 ±0.45	0.23 ±0.21
30	2.67 ±0.26	1.80 ±0.37	0.85 ±0.10	16.75 ±4.17	8.29 ±1.46
31	3.44 ±1.36	1.32 ±0.27	1.44 ±0.38	31.60 ±2.93	23.59 ±6.01
32	1.41 ±0.19	0.79 ±0.13	0.55 ±0.16	12.47 ±4.00	7.43 ±2.61
33	22.40 ±1.52	11.56 ±2.43	9.87 ±0.80	183.36 ±20.75	99.12 ±5.81
34	0.14 ±0.08	0.11 ±0.06	0.08 ±0.07	0.54 ±1.05	0.22 ±0.32
35	0.03 ±0.05	0.03 ±0.04	0.03 ±0.04	0.09 ±0.37	0.05 ±0.16
36	1.30 ±0.20	0.80 ±0.11	0.43 ±0.09	11.48 ±2.56	6.45 ±2.38
37	6.54 ±0.63	2.98 ±0.72	1.92 ±0.17	34.26 ±7.30	14.86 ±2.31
38	16.90 ±4.73	7.02 ±0.76	5.40 ±0.77	83.57 ±8.68	49.98 ±6.46
39	1.42 ±0.20	1.02 ±0.21	0.53 ±0.12	18.66 ±7.83	9.70 ±3.58
40	1.66 ±0.48	0.93 ±0.23	0.80 ±0.40	10.21 ±2.33	6.13 ±1.62

Table 3.5: Mean training runtime measures with standard deviation comparing PTBU, PTBU-S and PTBU-DOE with $\tau = 0.1$ and $t_{max} = 5$ on the left side. On the right side comparing PTBU and PTBU-LCH with $k = 5$ and $t_{max} = 10$.

3.6 Top-down Approach

3.6.1 Discussion

As explained above, pattern tree induction seeks to create a (fuzzy) logical representation for each class in an iterative way. This is done in a bottom-up manner by repeatedly combining two (of the currently best) candidate trees with a new root. The bottom-up strategy can indeed be motivated intuitively. For example, it can be seen as an iterative combination and construction of complex features, corresponding to trees and subtrees, from basic features, given by the original attributes.

Nevertheless, we believe that a top-down (PTTD) instead of a bottom-up (PTBU) construction of pattern trees is an interesting alternative[1]. Instead of merging two trees into a completely new tree, being much bigger and having a quite different structure, the idea is to modify the current tree only slightly. This can be done by expanding one of its leaf nodes, namely by replacing a basic feature $F_{i,j}$ with a compound feature (two basic features, one of them being $F_{i,j}$ itself, combined by a single operator). Thus, new operators are introduced at the bottom of the tree and not on the top, which normally means that later operators have a smaller influence on the input-output behavior than those that were chosen earlier. We shall explore this idea in more detail in Section 3.6.2 below.

To help in understanding the main differences between bottom-up and top-down induction, and to highlight potential advantages of the latter, it is appealing to compare them with operators used in genetic algorithms (even though this analogy is admittedly not perfect): While the merging of two pattern trees into a new tree in bottom-up induction can be seen as a kind of *recombination* operator (combining two solutions into a new one), the expansion of a leaf node in top-down induction is more in line with a *mutation* (modifying a single solution). Now, it is well-known that recombination can be quite beneficial, at least if the two solutions are complementary. However, to actually reach an optimum once being close to it, the mutation operator is indispensable. In particular, by making only small steps in the search space, it allows one to fully explore this space, whereas a recombination alone does not. Indeed, by combining complete candidate trees, bottom-up induction tends to make "large jumps" in the search space ("genotype space"). For example, the combination operator may double the number of

[1] Parts of this section were already published in [93].

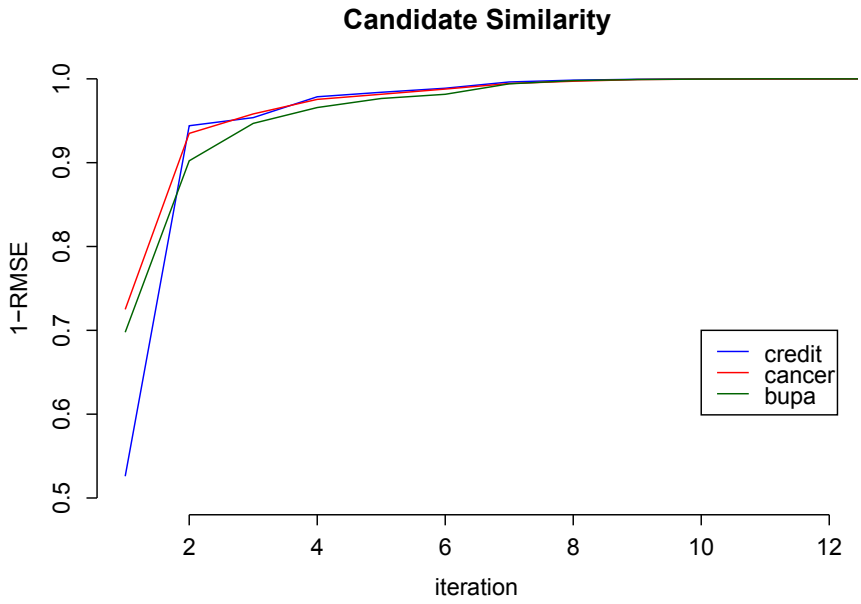


Figure 3.6: Pairwise similarity between candidate models (averaged over all pairs of candidates and over 50 random samples) for three binary datasets: credit, bupa, cancer.

nodes of the candidate trees in each iteration. As opposed to this, top-down induction implements a more fine-grained exploration of the search space, trying a large number of small steps in each iteration.

Besides, we also found another problem of the bottom-up approach, namely a lack of diversity. In fact, despite strong differences on the structural ("genotype") level, it turns out that, after only a few iterations, all candidate trees tend to be very similar to each other in the sense of implementing very similar $\mathbb{X} \to [0,1]$ mappings (i.e., the trees are similar on the "phenotype" level). Consequently, a combination of two such trees will remain almost ineffective. Figure 3.6 illustrates this problem by plotting the average pairwise similarity between two candidate models M_1 and M_2, in terms of

$$1 - \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} (M_1(\boldsymbol{x}) - M_2(\boldsymbol{x}))^2}$$

(RMSE-based similarity), as a function of the number of iterations.

Against the background of these considerations, one may expect that bottom-up induction is more likely to get trapped in local optima or to find suboptimal solutions, whereas top-down induction is more often able to reach a real optimum. Our

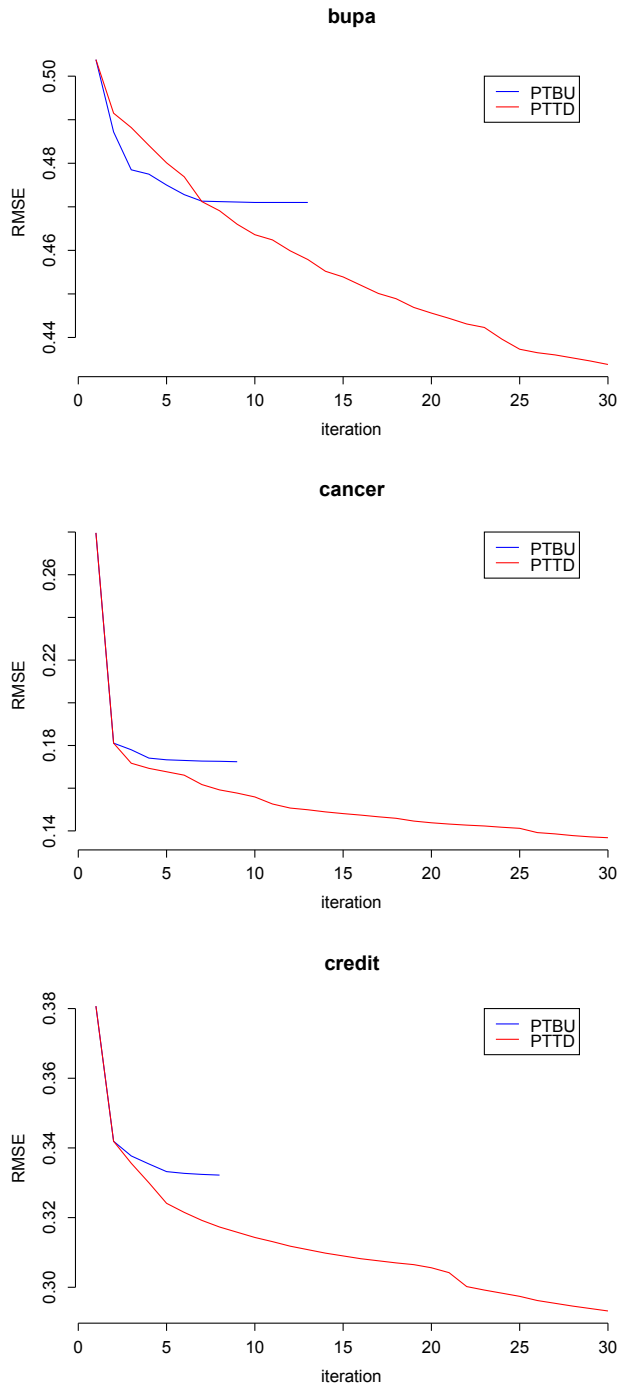


Figure 3.7: Error curve (top-down strategy in dashed, bottom-up strategy in solid line, averaged over the classes) on three datasets: bupa, cancer and credit.

experimental results are indeed in agreement with this conjecture. As an illustration, Figure 3.7 shows some typical error curves of the two approaches, plotting the error of the best model (on the training data) in each iteration against the number of the iteration. As can be seen, the two approaches produce quite similar results at the beginning, where the trees are very small; of course, a difference between top-down and bottom-up induction cannot be expected at this stage. However, while bottom-up induction reaches a saturation level quite quickly and apparently converges to a suboptimal solution, top-down induction is still able to achieve improvements of the solution.

Termination Criteria

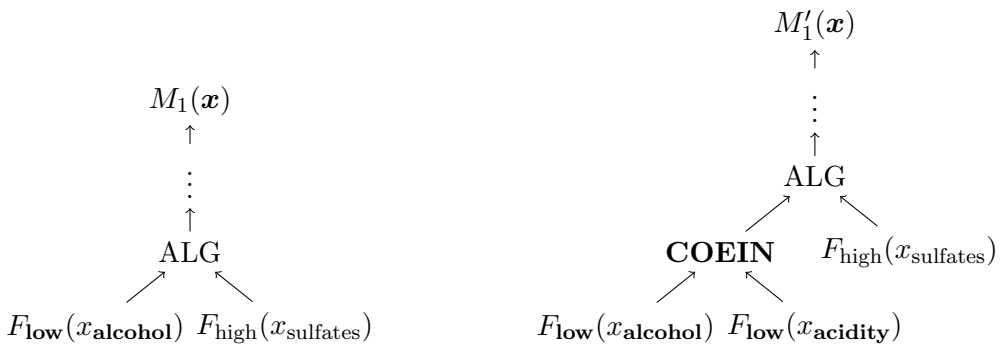
Recall that two termination criteria are used in the PTBU algorithm. The first criterion is satisfied if no improvement is achieved, i.e., if the tree with the lowest error in iteration $t-1$ could not be improved by the lowest error in the following iteration t. Thus, if the condition $l^t_{min} \geq l^{t+1}_{min}$ holds, the induction process is stopped and the best tree of iteration t is chosen to be the final tree for the respective class. This is a standard stopping condition commonly used in iterative optimization procedures.[1]

The second termination criterion is satisfied if a candidate tree reaches a certain depth. Note that, since candidate trees grow by one level in each iteration, this criterion is equivalent to limiting the number of iterations, and hence the runtime of the algorithm.

To get an idea of the relevance of the two termination criteria, we conducted an experimental study in which we applied the pattern tree classifier to a number of datasets and checked which of the two termination criteria was responsible for stopping the algorithm. More specifically, as proposed in [49], we used PTBU with $t_{max} = 5$. This classifier was applied ten times to each of the forty datasets. As a result, we found that the algorithm terminates due to the maximum-depth criterion in almost 90% of the cases, whereas the no-improvement condition is responsible in only 10%.

Despite being the more important stopping condition, or perhaps rather because of that, the maximum-depth criterion can be criticized. Obviously, it does not take the complexity of the learning task into consideration or, more specifically, the decision

[1] In general, this criterion can be criticized for being myopic. A standard way to improve it is to use a kind of lookahead search.



(a) M_1 before top-down extension.

(b) M_1 after top-down extension M_1'.

Figure 3.8: Top-down induction: A leaf node is expanded through replacement with a three-node tree.

boundaries separating the classes in the input space. In fact, while some classes can be characterized quite easily with a small pattern tree, more complex trees might be needed for other classes. Thus, it is clear that the maximum-depth criterion comes with the danger of stopping either too early or too late.

More specifically, looking again at the performance curves in Figure 3.7, it seems that the original pattern tree learner may actually stop too late: Given that the improvements in later iterations are negligible (in fact hardly visible graphically), the algorithm could stop earlier, thereby producing smaller models of the same quality.[1] For top-down induction, on the other hand, the condition is prone to premature stopping, thereby preventing this method from finding an optimal solution. To avoid this deficiency, we shall propose an *adaptive* termination criterion, that is, a criterion which seeks to adapt the model complexity to the difficulty of the learning task in an optimal way.

3.6.2 Top-down Induction

Based on the above discussion of possibilities to improve the original pattern tree learner, concrete modifications will be proposed in this section.

The algorithm for learning pattern trees in a top-down manner (PTTD) is presented in pseudo-code in Listing 3.9. It implements a beam search and maintains the B best

[1] Following the principle of Occam's razor [16], these smaller trees should be preferred.

Top-down Algorithm

{initialization}
$\mathbf{C^0} = \mathbf{P} = \{..., F_{i,j}, ... | i = 1, ..., n; j = 1, ..., m\}$
$M^* = \underset{M \in \mathbf{C}^0}{\text{argmin}}[err_{\mathcal{T}}(M)]$
$\epsilon = 0.0025$
$t = 0$
$d_{max} = 0$
{loop on iterations}
while $|\mathbf{C}^t| == 0$ **or** $l^t_{min} > (1 + \epsilon) l^{t-1}_{min}$ **do**
  $t = t + 1$
  {loop on each leaf}
  for all $L \in \text{leafs}(M^*)$ **do**
    {skip leafs of max depth}
    if $d_{max} == 0$ **or** $\text{depth}(L) < d_{max}$ **then**
      {loop on each available operator}
      for all $\psi \in \Psi$ **do**
        {loop on nearly each primitive pattern tree}
        for all $P \in \mathbf{P} \backslash L$ **do**
          $\mathbf{C}^t = \mathbf{C}^t \cup \text{replaceLeaf}(M^*, L, \psi, P)$
        end for
      end for
    end if
  end for
  {evaluate candidate trees and select}
  $M^* = \underset{C^t_i \in \mathbf{C}^t}{\text{argmin}}[l_{\mathcal{T}}(C^t_i)]$
  $l^t_{min} = \min_{C^t_i \in \mathbf{C}^t} [l_{\mathcal{T}}(C^t_i)]$
end while
return M^*

Figure 3.9: Top-down algorithm

models so far ($B = 5$ is used as a default value).

The algorithm again starts by initializing the set of all primitive pattern trees $\mathbf{P}$.

Additionally, the first candidate set, $\mathbf{C}^0$, is initialized by the B best primitive pattern trees, i.e., the trees with the lowest error.

After initialization, the algorithm iterates over all candidates trees. Starting from line 11, it seeks to improve the currently selected candidate C_i^{t-1} in terms of a lower training error. To this end, new candidates are created by tentatively replacing exactly one leaf node L (labeled by a fuzzy term) of C_i^{t-1} by a new subtree. This new subtree is a three-node pattern tree that again contains L as one of its leaf nodes (see Figure 3.8 for an illustration). The new candidate tree thus obtained is then evaluated by computing its error. Having tried all possible replacements of all leaf nodes of the trees in $\mathbf{C}^i$, the B best candidates are selected and passed to the next iteration, unless the termination criterion is fulfilled.

In order to lower the overall runtime of the algorithm, all possible three-node trees can be precomputed in advance and the outputs they produce can be stored for later use. Additionally, an implementation should also consider to recalculate the output of an adapted candidate tree in the following way: Starting from the substitute three-node tree, only the nodes on the path from that subtree to the root node of the candidate need to be recalculated.

To make the termination criterion adaptive and problem-specific (cf. Section 3.6.1), our idea is to look for the *relative* improvement of the best model in the t-th iteration as compared to the $(t-1)$-st iteration. More specifically, our algorithm stops if

$$l_{min}^t \geq (1-\epsilon) l_{min}^{t-1} \ , \tag{3.15}$$

i.e., if the relative improvement is smaller than ϵ, where $\epsilon \in (0,1)$ is a user-defined parameter. Based on empirical evidence, we propose $\epsilon = 0.25\%$ as a suitable value for this parameter. Noting that the error measures l_{min}^t $(t = 1, 2, \ldots)$ form a monotone decreasing sequence upper-bounded by 1 and lower bounded by 0, the above termination criterion appears to be reasonable. In fact, as shown in Figure 3.7, when plotting l_{min}^t against t, one typically obtains a curve with a convex shape, strictly decreasing at the beginning but flattening and converging toward a saturation level after a period of time.

Despite proposing this new criterion, the pseudo code includes a second termination criterion, i.e., trees can be limited by their hight. Leaf nodes only get extended if they did not reach a maximum depth d_{max} yet. In fact, this criterion is equivalent to the t_{max}-criterion of PTBU in terms of the resulting maximum tree size. In each iteration

of the PTBU algorithm at most one tree level can be added. Hence, the largest tree possible is a full binary tree of hight t_{max}. The same holds for the d_{max}-criterion of the PTTD algorithm.

3.6.3 Experiments with PTTD

After the theoretical discussion about the potential advantages of PTTD over PTBU, in this section, we will support them by empirical evidence.

In the following experiments, we included two variants of the original pattern tree algorithm in the experiments, PTBU5 ($t_{max} = 5$) and PTBU10 ($t_{max} = 10$); these variants were also used in [49]. Likewise, we included two variants of the top-down pattern trees, namely PTTDϵ.5 and PTTDϵ.25. These two variants only differ with respect to the parameter ϵ of our new termination criterion (3.15). PTTDϵ.5 is parameterized with $\epsilon = 0.5\%$, whereas PTTDϵ.25 uses $\epsilon = 0.25\%$. As an evaluation criterion, we used RMSE (3.8) for both top-down and bottom-up learning. Moreover, for all variants, a beam size of $B = 5$ and the same fuzzy partitions for input attributes were used, namely those described in Section 3.1.

Recall that our new pattern tree learner differs from the original one with regard to two properties, namely the direction of tree construction (top-down versus bottom-up) and the termination criterion. In order to differentiate the effects produced by these two modifications, we included yet another variant of the top-down pattern tree learner, namely PTTD5. This variant does not use the adaptive termination criterion (3.15) but the two standard termination criteria of the original algorithm, namely the no-improvement and the maximum-depth (with $d_{max} = 5$) conditions.

3.6.3.1 Classifier Accuracy

In the first experimental study, we measured the classification rate (percentage of correct predictions) of the different methods. The results (mean accuracy with standard deviation in brackets) are summarized in Table 3.6. Note, however, because some of the implementations are not able to deal with missing values, we conducted a full-case-study, i.e., for each dataset, we removed all instances, which had missing values. Therefore, the results deviate from those in the previous experiments.

In order to verify that the top-down induction strategy, even without the adaptive termination criterion, is able to improve upon the original bottom-up strategy, we first

No.	PTTDϵ.5	PTTDϵ.25	PTTD5	PTBU5	PTBU10
1	76.01±2.1 (4)	75.82±2.48 (5)	76.25±2.3 (3)	76.6 ±1.91 (1)	76.5 ±2.22 (2)
2	77.6±9.6 (4)	75.96±11.16(5)	79.56±10.26 (1)	79.33±10.94 (2)	77.96±10.47 (3)
3	39.4±6.11 (3)	39.5 ±6.42 (2)	40.3 ±6.34 (1)	37.25±6.13 (4)	36.5 ±6.67 (5)
4	85.51±8.46 (3)	90.95±7.83 (1)	88.83±9.02 (2)	83.9 ±8.11 (4)	83.4 ±8.17 (5)
5	94.92±2.72 (3.5)	94.77±2.4 (5)	94.92±2.37(3.5)	95.52±2.82 (2)	95.83±2.88 (1)
6	85.51±3.59 (3.5)	85.57±3.6 (2)	85.51±3.59(3.5)	85.62±3.58(1)	85.39±3.68 (5)
7	98.36±1.11 (2)	99.07±0.9 (1)	96.62±1.83(3)	94.63±3.03 (5)	95.65±2.61 (4)
8	72.82±9.66 (2)	74.65±9.51 (1)	69.36±9.26 (4)	65.88±9.43 (5)	71.32±8.4 (3)
9	87.14±3.92 (2)	87.29±3.32 (1)	86.05±3.46 (3)	85.43±3.09 (4.5)	85.43±3.09 (4.5)
10	87.05±7.89 (2)	88.5 ±6.82 (1)	86.37±7.54 (3)	85.62±7.3 (4.5)	85.62±7.36 (4.5)
11	76.93±1.8 (5)	77.03±1.84 (2)	77.33±1.85 (1)	76.98±1.42 (4)	77.01±1.47 (3)
12	70.74±7.49 (2)	70.69±7.04 (3)	71.37±7.74 (1)	68.71±8.25 (5)	69.36±7.4 (4)
13	96.22±2.45 (3.5)	96.55±2.19 (2)	96.78±2.25 (1)	95.9 ±2.33 (5)	96.22±2.49 (3.5)
14	72.21±4.43 (3)	72.6 ±4.35 (1)	72.41±4.5 (2)	71.08±4.09 (4)	70.98±4.37 (5)
15	42.31±14.55(2)	41.25±14.41(4)	42.33±15.42 (1)	42 ±15.38 (3)	40.84±14.47 (5)
16	54.92±3.37 (3)	55.49±3.29 (1)	55.17±3.07 (2)	53.86±4 (5)	53.93±3.39 (4)
17	86.5±9.43 (3)	87.11±9.26 (2)	87.64±9.37 (1)	83.96±9.84 (4)	83.14±10.96 (5)
18	85.51±3.75 (2.5)	85.42±3.68 (4)	85.51±3.75 (2)	85.62±3.78 (1)	85.39±3.78 (5)
19	87.67±12.19(1)	83.24±11.73(3)	87.62±11.06 (2)	74.52±12.83 (5)	79.95±12.4 (4)
20	67.41±8.63 (2)	68.29±7.47 (1)	66.36±7.58 (4)	67.36±8.2 (3)	65.3 ±9.2 (5)
21	82.72±0.78 (4)	82.74±0.75 (3)	82.67±0.74 (5)	82.82±0.63 (2)	82.84±0.63 (1)
22	73.38±2.81 (4.5)	74.28±3.35 (1)	73.64±3.25 (2)	73.52±3.28 (3)	73.38±3.01 (4.5)
23	64.4±8.8 (3)	67.51±8.34 (1)	64.78±8.64 (2)	60.83±7.3 (5)	62.98±8.37 (4)
24	74.37±6.29 (5)	74.64±5.85 (2)	74.9 ±5.89 (1)	74.44±5.68 (3.5)	74.44±5.88 (3.5)
25	82.07±6.08 (3.5)	82.59±5.51 (1.5)	82.07±5.85 (3.5)	82.59±5.75 (1.5)	81.78±6.01 (5)
26	91.28±5.4 (3)	91.97±4.87 (1)	91.68±5.04 (2)	88.49±5.39 (5)	88.84±5.42 (4)
27	95.6±5.6 (2)	95.6 ±5.6 (2)	95.6 ±5.6 (2)	95.33±5.7 (4)	95.07±6.23 (5)
28	97.68±2.09 (2)	98.68±1.86 (1)	97.28±2.15 (5)	97.64±2.14 (3.5)	97.64±2.14 (3.5)
29	75.08±12.5 (5)	77 ±12.85(3)	75.67±13.08 (4)	77.44±12.28 (1.5)	77.44±12.28 (1.5)
30	82.41±7.25 (2)	83.8 ±7.61 (1)	82.29±7.73 (3)	79.46±9 (5)	79.87±8.73 (4)
31	37.33±5.94 (5)	38.11±6.89 (4)	38.82±6.41 (2)	38.77±5.56 (3)	40.56±5.05 (1)
32	76.3±5.22 (3)	76.49±4.92 (2)	76.54±4.93 (1)	75.75±5.11 (4)	75.68±4.99 (5)
33	42.01±5.33 (4)	43.18±6.13 (2)	43.48±6.48 (1)	41.83±5.7 (5)	42.3 ±6.23 (3)
34	75.2±11.22 (3)	77.5 ±11.01(1)	76.7 ±11.21 (2)	73.5 ±10.11 (4.5)	73.5 ±10.11 (4.5)
35	83.84±6.88 (1)	83.76±6.9 (2.5)	83.76±7.08(2.5)	83.6 ±6.85 (4.5)	83.6 ±6.85 (4.5)
36	66.94±7.82 (2)	67.59±7.34 (1)	66.65±7.12 (3)	64.29±7.3 (5)	66.29±7.28 (4)
37	80.69±8.73 (1)	80.38±8.92 (2)	74.86±8.48 (3)	70.81±9.85 (4)	69.56±10.41 (5)
38	69.34±3.84 (2)	71.47±3.39 (1)	66.76±4.17 (3)	63.05±4.73 (5)	64.37±4.4 (4)
39	98.2±3.06 (2)	98.31±2.81 (1)	97.19±3.75 (3)	93.35±5.93 (5)	94.25±5.72 (4)
40	90.25±7.02 (5)	91.07±6.65 (3)	90.45±7.15 (4)	92.67±7.89 (2)	93.47±6.73 (1)

Table 3.6: Average classification rates and standard deviation on test sets.

compared PTTD5 and PTBU5. To this end, we applied a *Wilcoxon signed-rank test* [111]. This is a non-parametric test that ranks the differences in performances of two classifiers for each dataset, ignoring the signs, and comparing the ranks for the positive and the negative differences. For the results in Table 3.6, the test rejects the null

hypothesis of equal performance at the 5% significance level (the p-value is 0.00017, the average positive and negative rankings are $R_+ = 17.225$ and $R_- = 3.275$, respectively). Thus, one can conclude that a top-down induction strategy is significantly better than a bottom-up strategy, a finding that is furthermore confirmed by a simple sign test (PTTD5 wins 30 times and PT5 only 10 times).

Next, we evaluated the effect of our adaptive termination criterion. To this end, we compared PTTD5 with PTTDϵ.25. Again, the Wilcoxon test finds a significant difference in performance (at the significance level of 5%), this time in favor of PTTDϵ.25 or, in other words, in favor of our adaptive stopping condition (the p-value is 0.033, the average positive and negative rankings are $R_+ = 12.937$ and $R_- = 5.587$, respectively). In terms of wins and losses, PTTDϵ.25 wins 23 times, loses 15 times and ties 2 times.

Summarizing this experimental study, we can conclude that the top-down pattern tree learner PTTDϵ.25 is significantly stronger than the original algorithm for pattern tree induction. In particular, both modifications, the top-down induction strategy and the use of an adaptive stopping criterion, seem to pay off and improve predictive performance.

3.6.3.2 Overfitting

As one of the advantages of their approach, Huang et al. emphasize the fact that the learning method apparently does not tend to overfit the training data. In order to demonstrate this property, they compared the performance of the learner on the training data and the test data, and found that the former is not much higher than the latter.

Since we modified the original algorithm in several ways, notably by using a different termination criterion, an obvious question is whether or not this robustness toward overfitting is preserved by our learner. For this reason, we repeated the same kind of experiment. Before presenting the results, however, we like to point out that the experiment itself should be considered with caution. Even though it is true that a small training error coming along with a high test error normally indicates an overfitting effect, the difference between these two errors alone does actually not, as it loses information about the absolute error rates. Consequently, an overfitting effect cannot be distinguished from an underfitting effect. For example, although a learner with 10% training and 20% test error seems to overfit a bit, it is still preferable to a learner with

No.	PTTDϵ.25	PTTDϵ.5	PTTD5	PTBU5	PTBU10
1	2.77	3.06	2.33	2.48	6.96
2	21.98	11.06	18.38	12.42	6.58
3	1.00	3.85	0.30	6.25	20.75
4	3.91	0.91	11.17	2.52	15.36
5	0.69	2.05	0.92	2.20	3.03
6	0.06	0.14	0.00	0.17	5.62
7	0.45	3.23	3.02	2.52	3.52
8	12.66	3.77	23.32	19.49	23.32
9	0.07	0.06	1.31	1.77	4.65
10	2.89	0.51	5.02	3.37	7.20
11	0.11	0.48	0.19	0.43	3.88
12	2.35	2.89	3.12	4.91	15.28
13	0.23	0.26	1.02	0.73	1.73
14	0.30	0.05	0.99	1.34	24.59
15	15.23	16.02	21.56	14.48	57.31
16	0.16	0.68	0.98	1.53	17.22
17	3.17	2.39	4.02	6.31	12.69
18	0.09	0.14	0.00	0.03	5.62
19	16.76	0.03	12.38	20.86	15.43
20	13.67	9.39	23.33	10.99	17.69
21	0.73	0.28	0.81	0.19	0.17
22	0.58	0.32	2.96	1.08	12.12
23	1.65	5.69	5.78	10.20	33.28
24	1.83	2.43	1.24	2.68	2.69
25	3.33	3.11	6.81	2.96	11.19
26	3.76	1.25	6.33	1.82	10.88
27	0.40	0.40	1.07	0.67	2.93
28	0.12	0.28	1.52	0.24	1.16
29	2.29	3.08	1.34	0.72	1.87
30	10.79	2.05	15.01	5.00	13.38
31	4.16	4.93	0.77	3.19	30.87
32	0.08	0.91	0.03	1.20	8.44
33	6.37	6.08	9.91	11.56	22.30
34	2.50	0.70	4.80	1.00	24.50
35	1.04	0.96	1.04	1.20	4.80
36	2.71	0.71	7.18	4.24	30.18
37	15.30	0.56	23.70	12.84	28.52
38	0.52	3.97	8.54	3.62	32.55
39	1.69	2.70	2.81	3.84	4.62
40	4.97	7.77	5.59	7.33	5.54
avg.	4.08	2.73	6.01	4.76	13.76

Table 3.7: Difference in accuracy between training and test data

30% training and test error, simply because the latter seems to strongly underfit the training data. (In particular, note that the majority classifier, which always predicts the most frequent class, is likely to have a very similar training and test error.) Seen

from this point of view, one may indeed argue that the truly relevant measure is the performance on the test data, which has already been presented above.

Despite these reservations, a comparison of training and test error is of course usefull and may provide some interesting information. Table 3.7 shows the differences between the average accuracy of a method on the training data and on the test data (corresponds to the results shown in Tables 3.6).

As can be seen, PTTDϵ.5 achieves the best result on average, followed by PTTDϵ.25. Overall, the results are plausible in the sense of being in agreement with the flexibility of the methods.

The larger the trees (due to a relaxed stopping condition), the worse the performance becomes. Worth mentioning in this regard is the poor performance of PTBU10, which is in contrast to what was reported in [49]. Yet, given that pattern trees of depth 10 are indeed rather complex models, this result is perhaps not surprising.

3.6.4 Fast Top-down Learning

In the last sections, we tried to improve predictive accuracy of the learning algorithms. In the following section however, we focus on the runtime of the top-down algorithm[1].

Recalling the general structure of the top-down algorithm, in each iteration many slightly different candidate models are created – by extending each leaf in a variety of ways. The number of these candidate models depend on the number of fuzzy sets (fuzzy partitions of attribute domains), the number of possible aggregation operators and the number of leaves of the current model, which we seek to improve. Depending on the dataset at hand, the number of candidate models can be quite large. Then, each model gets evaluated by first applying each model to the training instances and then comparing the predictions and the actual output in terms of a loss function. The runtime of this evaluation strongly depends on the number of training instances.

To be precise, the number of leaf nodes of M^* in the t-th iteration is t, and taking into consideration that the fuzzy sets combined in a three-node subtree (L and P in Fig. 3.8) should pertain to different variables, is easy to see that the number of candidate models is

$$|\mathbf{C}^t| = t \cdot a \cdot (m-1) \cdot |\mathcal{F}| \ , \tag{3.16}$$

[1]Parts of this section were submitted to be published in [94].

where a is the number of aggregation operators, m the number of variables, and $|\mathcal{F}|$ the size of the fuzzy partitions (number of fuzzy sets n_i in (3.1), for simplicity assumed to be the same for all variables). Needless to say, this number might be large, while many of the candidates will differ only slightly. Moreover, all candidates are tested on the entire training data, giving rise to $|\mathbf{C}^t| \cdot |\mathcal{T}|$ tree evaluations.

Two major changes will be introduced, which significantly reduce the runtime especially for large datasets with either many attributes or many instances. The first one seeks to speed up the identification of the best model among a set of candidate models (subsections *A*–*B*), whereas the second one restricts the total number of such candidates (subsection *C*).

3.6.4.1 Racing Algorithms

In [64] Maron et al. first introduced the idea of so-called *Hoeffding races* intended to accelerate the selection of a good model from a set of candidate models, which is a common task in machine learning methods. Especially in cases, where the set of candidate models and the number of test instances are large, evaluating all candidate models on every test example can be unnecessarily time consuming. Instead of evaluating the models on every test example (e.g. in a cross-validation setup), Maron suggests to have a *race* between the models.

A race starts with all available candidate models. These are treated as black boxes, that is why they were also called *learning boxes*. The racing algorithm iterates over a random order of the test set. In each iteration, one example (or a small portion of examples) is drawn to evaluate each of the models. For each model M_j, two variables are stored, i.e. the number of examples n_j it has been evaluated on and the current mean estimate of its error err_j.

We have to consider, however, that err_j is only an estimate of the true error of the j-th model. Though, we are able to calculate a confidence interval for the sample mean of the error using *Hoeffding's formula*:

$$\varepsilon = \sqrt{\frac{B^2 log(2/\delta)}{2n}}$$

Given a predefined confidence level $1 - \delta$ and a maximum possible error B, for each model ε_j determines how close the current estimate is to the true error of the j-th

model[1]. Then, we can eliminate every model from the race whose best possible error is still greater than the worst possible error of the current best model. This way, by evaluating on more and more examples the confidence intervals become tighter and more and more candidate models can be eliminated and hence are not evaluated on all test examples. See Figure 3.10 for an illustration.

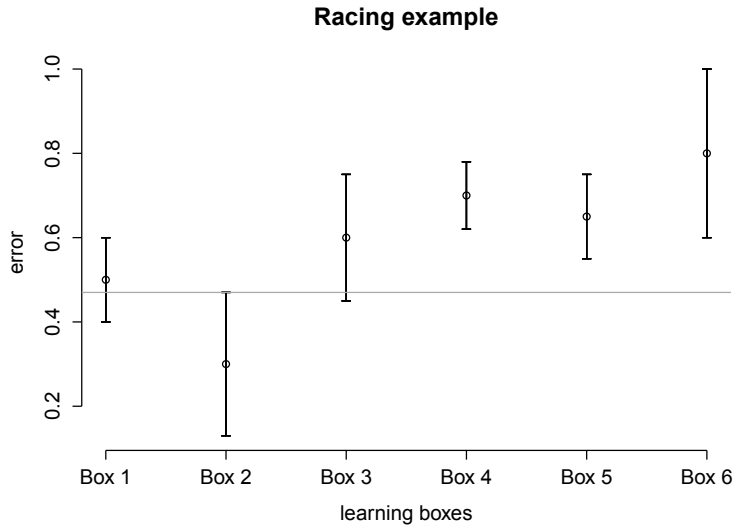


Figure 3.10: In this example, three of the six models (Box 1–3) remain in the race after this iteration. Models 4–6 can be excluded because their mean error is unlikely ($< \delta$) to become better than the one of the current best model (Box 2).

The race ends if one of the following three conditions holds:

1. Only one of the learning boxes remains in the race.
2. A predefined maximum number of test examples has been used (e.g. N).
3. ε has reached a predefined threshold.

The second and third condition can be chosen to best meet application requirements. If more than one model remains in the end, they are indistinguishable by $2\varepsilon_n$, where n is the number of test instances used.

Applying the Hoeffding bound for a given confidence level $1 - \delta$ produces a valid interval for one learning box in one iteration. In order to provide the validity of all

[1]Do not confuse the two different parameters ϵ and ε. The former constitutes the relative improvement threshold while the latter determines the error threshold for the racing algorithm.

confidence intervals in every iteration we have to account for multiple applications of the Hoeffding bound. Maron et al. suggest to simply use a Bonferroni correction [61]. To assure the confidence level $1 - \delta$ at the end of the racing algorithm, the individual confidence intervals have to be derived for $\delta' = \frac{\delta}{N|\mathbf{C}^t|}$, where N is the maximum number of iterations and $|\mathbf{C}^t|$ is the number of candidate models. The validity of the update rule then follows immediately from the union bound of probability.

In [65] Maron et al. introduced various extensions to the general idea of racing. Two of them will be briefly explained in the following.

Bounding Errors

B was said to be the maximum error, that we are able to measure for a learner. For classification tasks, B can easily be determined to be 1. For other error measures used for regression this might be not as easy. In these cases, Maron et al. suggests to estimate B by adding a few standard deviations to the mean of the error of the learner.

In our case, since our output always resides in $[0, 1]$ and we are measuring (1.2) an obvious bound is also 1.

Shrinking the Intervals

During the process of racing, all mean error estimates err_j move. Nonetheless, all intermediate intervals are valid as we have discussed before. This means that e.g. the lower bound of the j-th model at iteration k, say $lower_j^{(k)}$, remains valid for all later iterations. The same applies for the upper bound $upper_j^{(k)}$. These facts can help to shrink the intervals more efficiently by applying the following update rules:

$$lower_j^{(k+1)} = \max\{lower_j^{(k)}, err_{j+1} - e_{j+1}\}$$
$$upper_j^{(k+1)} = \min\{upper_j^{(k)}, err_{j+1} + e_{j+1}\}$$

Empirical Bernstein Stopping

In addition to the above extensions, in [68] Mnih et al. introduce the use of the *empirical Bernstein bound* [8] instead of the Hoeffding's inequality. It states that with probability at least $1 - \delta$

$$\varepsilon \leq \bar{\sigma}_n \sqrt{\frac{2log(3/\delta)}{n}} + \frac{3Blog(3/\delta)}{n} \ ,$$

where $\bar{\sigma}_n$ in our case denotes the empirical standard deviation of the error measured for a learning box. The term involving B decreases at the rate of $\frac{1}{n}$. The term involving the square root, however, does not depend on B anymore. Hence, when $\bar{\sigma}_n \ll B$ the empirical Bernstein bound becomes tighter than Hoeffding's inequality.

3.6.4.2 Application of Racing to Top-down Induction

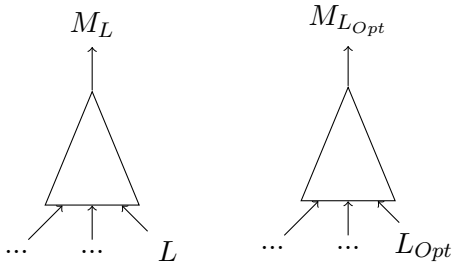
Starting from line 11 until line 23 in the top-down algorithm various candidate models are created. Then, these models are evaluated in order to select the best one, which becomes the starting point of the consecutive iteration. This selection process currently calculates the mean error of each candidate model on the given training data and then selects the model with the lowest one (lines 24-26). For this reason, it is actually the most time consuming step in the algorithm and we will replace it by the racing algorithm.

To this end, we can apply the racing algorithm as it is. We only have to consider a proper termination criterion for the race. Three potential criteria have already been proposed and we will use the first and third one. That is, we will stop when there is only one candidate model left or when ε has reached the threshold of $\epsilon/2$ for all remaining models. Our termination criterion of the top-down algorithm is parameterized by ϵ. This means we are not interested in smaller performance improvements. Hence, we stop the race if the remaining candidate models are not distinguishable by $2\varepsilon = \epsilon$.

3.6.4.3 The Potential-Heuristic

The second change aims at narrowing the search by using a heuristic. Recall that, in each iteration of the original top-down algorithm, new candidate models $\mathbf{C}^t$ are created by expanding every leaf node (line 17) of the current model. Our idea is to restrict this expansion to a constant (and typically small) number $p < |\mathbf{C}^t|$ of leaf nodes with the highest *potential*, a measure characterizing the leaf's ability to contribute to the overall improvement of the model. The general idea of the heuristic is to evaluate each leaf node in terms of its potential to improve the overall output of the tree, hence the name *potential-heuristic* (PH). The potential of a leaf node is defined as follows.

Let L be a leaf of a candidate tree $M_L \in \mathbf{C}^t$. We define a model $M_{L_{Opt}}$ by replacing L in M_L by L_{Opt}, where L_{Opt} is a fictitious "oracle leaf" that provides the ideal output



(a) M_L **(b)** $M_{L_{Opt}}$

Figure 3.11: Models compared to calculate the potential of leaf L.

y_i for every training instance $(\boldsymbol{x}_i, y_i) \in \mathcal{T}$ (cf. Fig. 3.11). Then, we define the potential of L in terms of the possible reduction of empirical risk:

$$\mathcal{P}_M(L) = err(M_{L_{Opt}}) - err(M_L)$$

Due to the monotonicity of the internal nodes, $\mathcal{P}_M(L)$ is non-negative, and the higher the potential, the more promising the leaf node appears to be. In fact, the potential provides an upper bound on the improvement that can be achieved by expanding L. Therefore, the p leafs with the highest potential are selected.

3.6.4.4 Runtime Considerations

Before presenting an empirical runtime analysis in the next section, we first have a look at the theoretical improvement of our fast PTTD variant.

The number of candidate models to be considered by the original PTTD algorithm in the t-th iteration is given by (3.16). Thus, upon termination of PTTD, the total number of models evaluated is $O(|M|^2 \cdot m)$, where $|M|$ denotes the size of the model produced, measured in terms of the number of leaf nodes—the number of aggregation operators, a, and the size of the fuzzy partitions, $|\mathcal{F}|$, are considered as constants here. Moreover, since the complexity of a single evaluation is linear in the size of the data set $\mathcal{T}$, the overall runtime complexity is $O(|M|^2 \cdot |\mathcal{T}| \cdot m)$. This essentially means that the runtime is linear in the size of the data set, namely the number of examples and the number of attributes, and quadratic in the size of the tree produced.

As for our fast variant, PTTD-fast, the use of the racing algorithm is supposed to reduce the number of the overall model evaluations in each iteration. However, there is

no guarantee. In the worst case, when all candidate models are almost equally good, it might still take a long time (many evaluations) to eliminate models until we are able to select the best one. Nevertheless, for the purpose of top-down induction, we are only interested in a ϵ-good solution. That is, we do not need to run each race until a single winner is found. We rather stop the race whenever we are satisfied with every one of the remaining candidate models. In the experimental section we will demonstrate the resulting decreased runtime empirically.

Using the potential-heuristic limits the number of leaf nodes that are expanded in each iteration to a constant number of p. That is, the number of candidate models created in later iterations no longer linearly depend on the number of leafs. Therefore, the PH strategy becomes more and more effective as the trees becomes larger. However, calculating the heuristic introduces an overhead cost, which arises for every leaf node in every iteration. This cost, though, basically only constitutes two evaluations of the current best model per leaf node. Before, we had to evaluate $a \cdot |\mathcal{F}|$ extensions. Needless to say that $a \cdot |\mathcal{F}|$ usually is much bigger than 2.

3.6.4.5 Experiments

For the purpose of an empirical study on the runtime behavior of PTTD-fast, we present a case-study on a large real world dataset. It has been used in the KDD Cup data mining challenge during the annual Conference on Knowledge Discovery and Data Mining in 1999. The dataset contains more than 4.8 million log entries about network connections in a simulated military network environment. Task of the challenge was to build an intrusion detection system to distinguish between "bad" connections, called intrusions or attacks, and "good" normal connections.

We evaluated the predictive performance of the models in terms of accuracy (proportion of correct predictions). Runtime of the training phase is measured in seconds (s) on a single core (2.3 GHz) with enough RAM available (120GB, no swapping).

In order to measure the effect of both modifications, we conducted two experiments. The first one aims at evaluating the effect of the PH strategy. As explained earlier, the PH strategy becomes more effective in the case of larger trees. Therefore, in the first setup we vary parameter d_{max} between 1 and 6. Additionally, ϵ was set to 0. This accounts for models that tend to be bigger with larger d_{max}. We compare the original PTTD algorithm to PTTD-PH, the variant using the PH strategy with $p = 3$.

d_{max}	PTTD	PTTD-PH
	Time	
1	0.5460 ± 0.0107	0.6397 ± 0.0819
2	9.0150 ± 0.7037	7.8127 ± 0.2851
3	39.9633 ± 1.3087	39.9193 ± 1.7134
4	189.5567 ± 6.7535	137.1877 ± 6.1803
5	558.8950 ± 31.566	253.7933 ± 12.116
6	1954.1900 ± 194.73	414.9607 ± 27.842
	Accuracy	
1	0.9488 ± 1.9E-03	0.9488 ± 1.9E-03
2	0.9824 ± 1.5E-04	0.9824 ± 1.5E-04
3	0.9931 ± 1.7E-04	0.9931 ± 1.7E-04
4	0.9953 ± 1.3E-04	0.9952 ± 1.2E-04
5	0.9957 ± 1.7E-04	0.9955 ± 1.7E-04
6	0.9960 ± 1.5E-04	0.9958 ± 1.6E-04

Table 3.8: Mean time and accuracy results (including standard error) of the comparison between PTTD and PTTD-PH for different d_{max} values.

We expected the two variants to behave equally for a maximum depth up to 2. Then, however, starting from $d_{max} = 3$, PTTD-PH should exhibit a shorter runtime because the number of leaf nodes for a potentially full binary tree of this hight exceeds 3.

In this experiment, we repeated the following procedure 30 times and averaged the results. First we shuffled the data and split it into two parts. The test part contained 10% of the whole dataset. Then, we randomly selected 2000 examples from the remaining training part to train the model[1].

Looking at the results shown in Figure 3.12 (precise values are given in Table 3.8) we can confirm our expectation. With an increasing size of the tree, the runtime increases for both methods, however, PTTD-PH becomes more efficient starting from $d_{max} = 3$. In terms of accuracy the figure nicely illustrates, that PTTD-PH still keeps the same level of accuracy in comparison to PTTD. This result indicates that using the potential measure (3.11) effectively helps in selecting the right leafs to be extended. Dropping low-potential leafs did not affect the predictive quality of the resulting models.

In our second experiment we compare three algorithmic variants. PTTD again denotes the original top-down induction algorithm, PTTD-R makes use of racing but

[1]We limited the number of training examples in order to facilitate a reasonable runtime for PTTD.

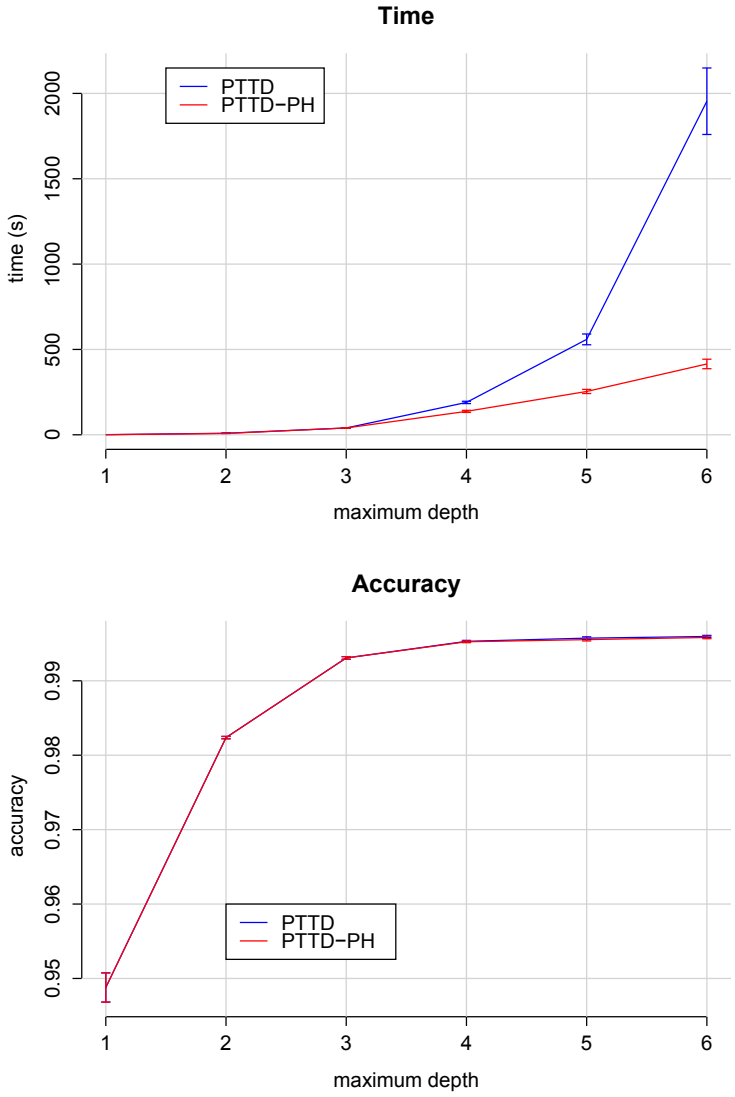


Figure 3.12: Result of the comparison between PTTD and PTTD-PH for different d_{max} values. The upper diagram shows the training time and the lower shows predictive accuracy of the methods including standard error bars.

not of PH, while PTTD-fast uses both racing and PH. This time, however, we keep $d_{max} = 3$ constant and vary the size of the training set. Additionally, we set $\epsilon = 0.01$ for all variants. The racing algorithm is parametrized with a confidence level of 0.95 ($\delta = 0.05$). We conducted the experiments with both bounds, Hoeffding and Bernstein but we found, that the results for the Bernstein bound hardly differ from the one achieved using the Hoeffding inequality. Therefore and for the sake of clearness, we only present the results for Hoeffding. Finally, for PTTD-fast, we set the number of high-potential leaf nodes to be 3. We used the same experimental setup as before. This time however the number of training examples varied between 250 and 1024000, whereas d_{max} was kept constant.

The results of the second experiment are shown in Figure 3.13 (see Table 3.9 for the detailed values). First of all, note that the number of examples are presented on a logarithmic scale. Keeping this in mind, PTTD clearly shows a linear increase in training time with an increasing number of training examples. This verifies our runtime considerations, identifying $|\mathcal{T}|$ as a linear factor for PTTD. On the contrary, PTTD-R and also PTTD-fast exhibit a considerable smaller runtime. The difference in comparison to PTTD grows for larger training sets. In fact, the training runtimes of PTTD-R and PTTD-fast grow only sub-linearly. The difference between PTTD-R and PTTD-fast is relatively small, however, PTTD-fast still clearly outperforms the variant without PH strategy. Moreover, the first experiment suggests that the difference can be even bigger for larger trees. Looking at the predictive accuracy of the three methods the bottom diagram of Figure 3.13 shows a standard learning curve with a saturation starting from 32000 instances. All three curves are very close to each other, however, PTTD-fast slightly hangs behind of the other two variants. Yet, the differences are not significant.

size	PTTD	PTTD-R	PTTD-fast
		Time	
250	10.2587 ± 0.8360	3.7200 ± 0.2779	3.1877 ± 0.2074
500	23.7040 ± 1.4132	8.5853 ± 0.5283	6.0333 ± 0.4127
1000	53.0553 ± 2.5585	17.1697 ± 0.8692	13.9383 ± 0.8639
2000	179.9030 ± 5.9025	37.9033 ± 1.2048	27.6983 ± 1.7574
4000	377.5310 ± 12.6147	67.6223 ± 3.0631	44.1893 ± 2.9369
8000	805.2417 ± 17.4475	135.3683 ± 4.3245	89.6927 ± 6.3608
16000	1574.1090 ± 47.5882	253.3040 ± 10.1971	153.5180 ± 8.6051
32000	3304.6977 ± 78.5015	439.7010 ± 17.0745	273.8480 ± 12.5710
64000	6394.3067 ± 138.9887	540.4683 ± 16.9115	423.1977 ± 7.5181
128000	12567.8710 ± 250.4649	666.9857 ± 15.7962	534.6757 ± 11.7633
256000	23936.8793 ± 510.9649	944.4373 ± 29.2997	750.5740 ± 15.4544
512000	48250.0403 ± 838.9186	1392.5327 ± 39.1369	1181.7050 ± 20.5539
1024000	98569.1807 ± 1633.0795	2279.9153 ± 42.7369	1993.1643 ± 30.3936
		Accuracy	
250	0.9921 ± 5.12E-04	0.9921 ± 4.96E-04	0.9921 ± 4.89E-04
500	0.9932 ± 4.06E-04	0.9932 ± 4.06E-04	0.9931 ± 4.16E-04
1000	0.9946 ± 2.16E-04	0.9946 ± 2.16E-04	0.9944 ± 2.14E-04
2000	0.9953 ± 1.26E-04	0.9954 ± 1.30E-04	0.9952 ± 1.23E-04
4000	0.9956 ± 1.64E-04	0.9956 ± 1.69E-04	0.9953 ± 1.68E-04
8000	0.9958 ± 1.08E-04	0.9958 ± 1.09E-04	0.9956 ± 1.14E-04
16000	0.9960 ± 8.96E-05	0.9960 ± 9.11E-05	0.9959 ± 8.91E-05
32000	0.9963 ± 2.92E-05	0.9963 ± 2.89E-05	0.9961 ± 4.46E-05
64000	0.9962 ± 2.30E-05	0.9962 ± 2.35E-05	0.9962 ± 1.94E-05
128000	0.9962 ± 5.89E-05	0.9961 ± 7.30E-05	0.9961 ± 7.04E-05
256000	0.9962 ± 1.84E-05	0.9961 ± 5.35E-05	0.9961 ± 5.23E-05
512000	0.9962 ± 1.69E-05	0.9962 ± 1.66E-05	0.9962 ± 1.78E-05
1024000	0.9962 ± 1.69E-05	0.9961 ± 5.80E-05	0.9961 ± 5.74E-05

Table 3.9: Mean time and accuracy results (including standard error) of the comparison between PTTD, PTTD-R and PTTD-fast for different training set sizes.

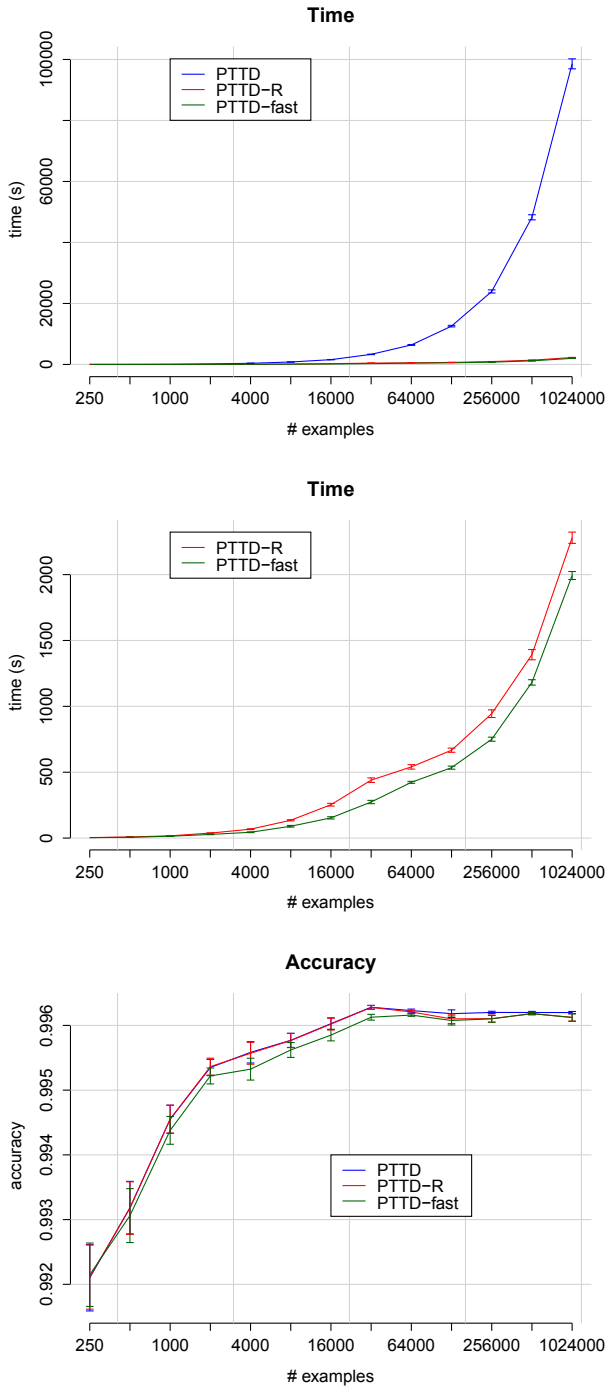


Figure 3.13: Result of the comparison between PTTD, PTTD-R and PTTD-fast for different training set sizes. The upper diagram shows the training time of all three variants. The middle one only shows PTTD-R and PTTD-fast to be able to visually distinguish them. Finally, the lower shows predictive accuracy of the methods. All diagrams are equipped with standard error bars.

3.7 Co-evolutionary Approach

PTBU as well as PTTD as outlined above can be criticized for several reasons, notably the following[1]. First, the learning algorithms implement a kind of greedy search in the hypothesis space. Since this space is extremely complex, it is likely to get stuck in local optima. Clearly, the complexity of the search space and the highly non-linear nature of the models prevents the use of search algorithms which guarantee optimality. Yet, there is hope that better solutions can be found at the cost of an increased though still acceptable search effort. To this end, we shall resort to search methods from the field of evolutionary optimization.

Second, one may argue that for classification the problem is made more difficult than necessary. In fact, as described above, the learning algorithm seeks to find, for each class y_i, a pattern tree that delivers outputs close to 1 for instances $\boldsymbol{x}$ from this class and outputs close to 0 for instances from other classes. This property is indeed a *sufficient* criterion for correct classification, but actually not a *necessary* one. Indeed, according to (3.3), a prediction is made by combining the outputs of all pattern trees using the $argmax$ operator. Therefore, a prediction is correct as soon as the true class receives the highest score. This does not mean, of course, that the score must be close to 1, while all other scores are close to 0. Trying to comply with this much stronger property will presumably lead to models that are more complex than necessary.

As an illustration, suppose that all classes are correctly characterized by simple linear functions (i.e., the trees have depth 2 and a WA operator as a root node). Combined with $argmax$, these functions will always produce the correct prediction, even though the outputs will not always be close to 0 and 1, respectively. Instead, more complex, non-linear models will be needed to produce these type of predictions.

3.7.1 Evolutionary Algorithms

Evolutionary algorithms (EA) are population-based stochastic search methods which seek to optimize a solution by mimicking the process of biological evolution. They can be applied in a quite universal way and have been used in a wide spectrum of application domains.

[1] Parts of this section were already published in [92].

To apply evolutionary algorithms to complex problems more efficiently, a modularization technique, referred to as *co-evolution*, has recently been proposed [80]. The general idea of co-evolution is to evolve the sub-components of a (structured) solution, also referred to as *species*, in different sub-populations.

To assure that the sub-components can be assembled into a globally optimal solution, the fitness of an individual in a species is evaluated by its ability to participate in a cooperative team consisting of one representative per species. The global fitness function used in this context is also referred to as the *shared domain model*.

Determining the fitness of individuals of a certain species can simply be achieved by the evaluation of collaborations formed with representatives from each of the other species. Representatives of a species can be individuals of a certain fitness, or even the whole population. Essential for the validity of an individuals fitness, which can also be seen as a measure of the individual's contribution to the overall solution, is the selection of representatives of the other species.

3.7.2 Co-evolutionary Fuzzy Pattern Tree Learning

In our concrete application of pattern tree induction, an individual is a single pattern tree. We evolve one species per class and denote by $I_i^{(t)}$ the t-th generation of the i-th species $S_i \in \mathcal{S}$, where $\mathcal{S}$ denotes the set of species. A hypothesis h consists of exactly one individual for each species, that is, one pattern tree for each class. As a fitness criterion (shared domain model), we use the classification accuracy of a hypothesis on the training data.

The evolutionary process comprises the following steps:

Step 1 - Initialization

To obtain a first generation of pattern trees, random pattern trees of size 1 or 3 are created. Here again, the size of a tree is defined as the number of nodes in the tree, including both, internal and leaf nodes.

Step 2 - Evaluation

After a new generation has been created, the fitness of all individuals of each species must be calculated. This is done by building every possible classifier, that is, every

combination

$$M^{(t)}_{j_1,...j_k} = (M^{(t)}_{1,j_1}, M^{(t)}_{2,j_2}, ..., M^{(t)}_{k,j_k}) \in I^{(t)}_1 \times I^{(t)}_2 \times ... \times I^{(t)}_k \tag{3.17}$$

of pattern trees with k denoting the number of species (classes) and M_{i,j_i} being the j_i-th pattern tree of the i-th species. The number of possible combinations is m^k, with m the size of each population (we evolve each species using the same population size).

As mentioned above, a hypothesis M is evaluated by its accuracy on the training data, i.e., acc(M). Moreover, the evaluation (fitness) of a single pattern tree $M^{(t)}_{j_i}$ is given by the best hypothesis in which it has participated:

$$Fit\left(M^{(t)}_{j_i}\right) = \max_{j_1,...,j_{i-1},j_{i+1},...,j_k} \left[\text{acc}(M^{(t)}_{j_1,...,j_k})\right] \quad .$$

Step 3 - Termination

The proposed Co-EA terminates if one of the following conditions holds. First, the iteration stops after a maximum number t_{max} of generations. Second, the Co-EA also stops if no significant improvement can be achieved during a certain number of iterations.

Therefore, two thresholds δ and $t_{improve}$ have been introduced to track the accuracy improvement of the most accurate (best) hypothesis M^t_{best} of each generation. If the condition

$$\text{acc}(M^{(t)}_{best}) + \delta \geq \text{acc}(M^{(t+l)}_{best})$$

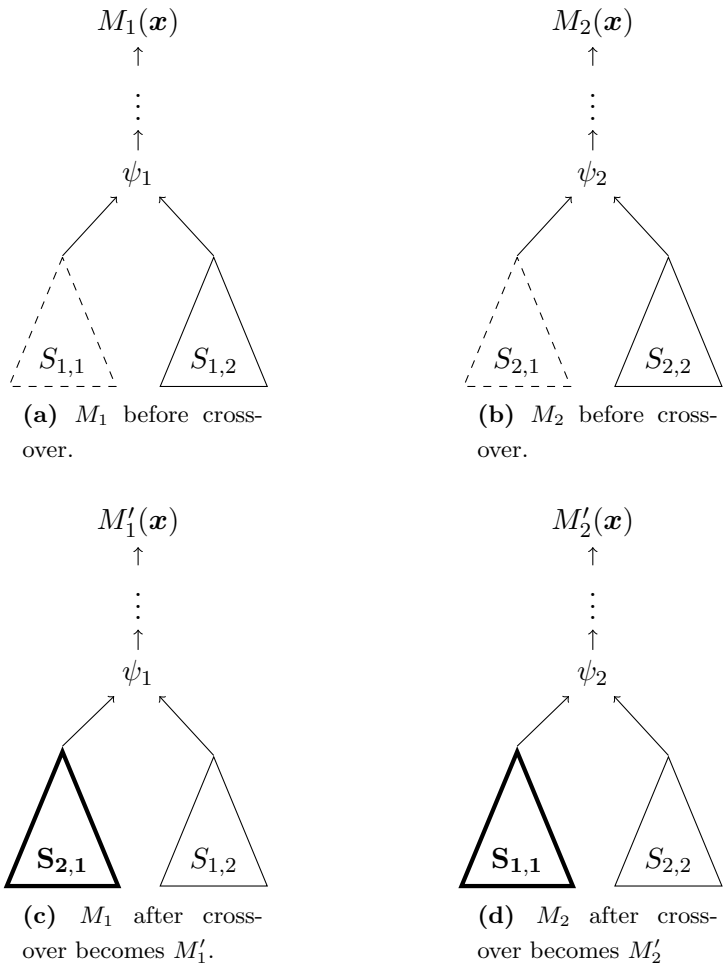
holds for the last $t_{improve}$ generations, the algorithm stops.

Step 4 - Reproduction

Reproduction in terms of EAs means the creation of a new generation. Therefore, individuals of the current (parent) generation are selected at random, with a probability proportional to their fitness, to form a set of parents.

To make sure that the best solution found so far is not lost, elitist selection is applied, which means that the best hypothesis of each generation is directly transfered to the new generation.

Given two individuals $M^{(t)}$ and $M'^{(t)}$, a cross-over operator on trees is used for reproduction, that is, to create their children $M^{(t+1)}$ and $M'^{(t+1)}$. To this end, one node



(a) M_1 before cross-over.

(b) M_2 before cross-over.

(c) M_1 after cross-over becomes M_1'.

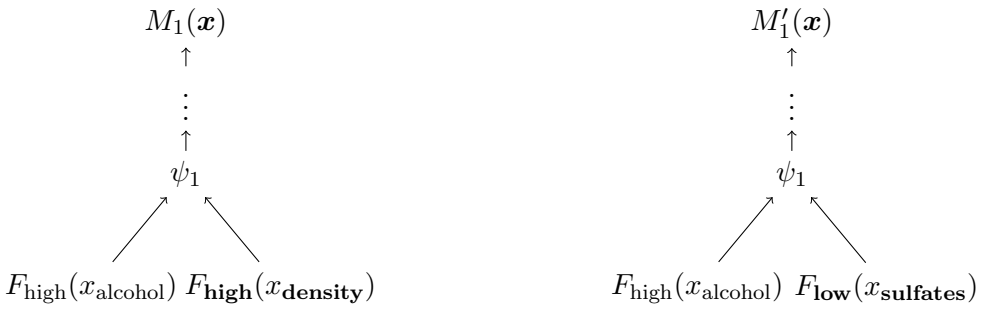
(d) M_2 after cross-over becomes M_2'

Figure 3.14: Example for the cross-over procedure. The dashed subtrees in (a) and (b) are chosen randomly. They are interchanged to create two new individuals M_1' and M_2' for the next generation.

within each tree is chosen at random and the corresponding subtrees are interchanged. Figure 3.14 illustrates this operation by means of an example.

Step 5 - Mutation

To guarantee a proper level of diversity, a set of randomly chosen individuals are mutated after reproduction. The probability of an individual being chosen for mutation is uniform over the population and determined by the predefined mutation rate p_{mutate}. If an individual has been chosen for mutation, one of three different mutation operators



(a) M_1 before leaf mutation.

(b) M_1 after leaf mutation becomes M_1'.

Figure 3.15: Example for the leaf mutation procedure.

is applied. The selection of the mutation operator is again random.

1. *Mutate Leaf:* Randomly selects a leaf node and replaces it by another randomly chosen leaf node. This comes down to replacing a fuzzy term F_{ij} of an attribute A_i by another term F_{lj} of another attribute A_l.

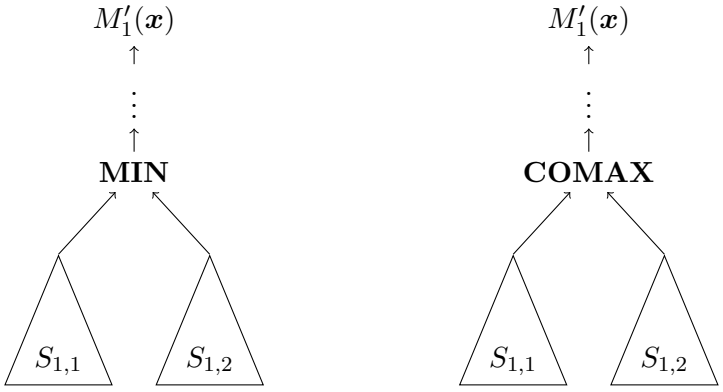
2. *Mutate Operator:* Randomly selects an internal node and replaces it with a randomly chosen different one. Since an internal node represents a fuzzy operator, this mutation changes the type of aggregation of its children.

3. *Mutate Tree:* Randomly selects a subtree and replaces it by a new, randomly created tree. This operator combines the first and the second one.

After the mutation step, the algorithm continues with step 2.

Experiments

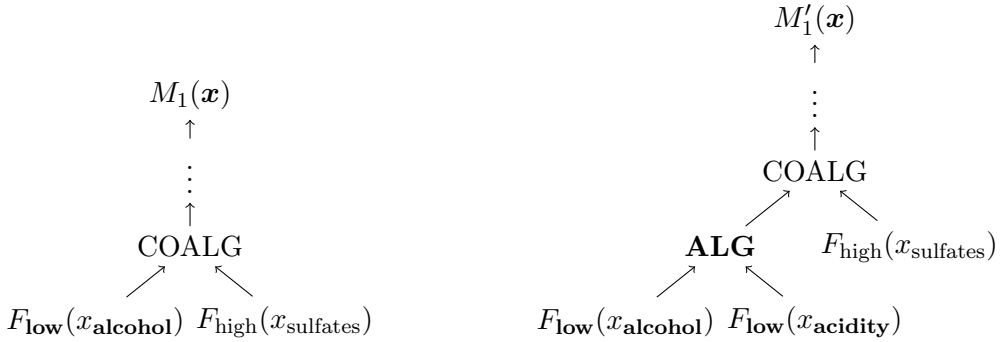
In the next chapter, we will compare this co-evolutionary approach – that we call PTCoEvo – to PTBU and PTTD in terms of predictive accuracy and training runtime.



(a) M_1 before operator mutation.

(b) M_1 after operator mutation becomes M_1'.

Figure 3.16: Example for the operator mutation procedure.



(a) M_1 before sub tree mutation.

(b) M_1 after sub tree mutation becomes M_1'.

Figure 3.17: Example for the subtree mutation procedure.

4

Experiments

This chapter consists of four sections. Each section concentrates on a specific question that we try to answer empirically. The first section is dedicated to the evaluation of the CI operator as a substitute for WA and OWA. Next, we compare the three main variants of fuzzy pattern tree induction algorithms discussed in this thesis empirically in terms of predictive accuracy and training runtime. Moreover, the third and forth section contain comparisons of the strongest fuzzy pattern tree variant to current state-of-the-art algorithms for classification and regression tasks.

4.1 CI vs. WA, OWA

We compare two sets of operators: the original one (the four t-norms and t-conorms, WA and OWA) as in [93] (denoted by PTTD) and a second one where we substitute WA and OWA by our new CI operator (denoted by PTTD-CI).

In order to compare the two operator sets by means of predictive accuracy and training runtime we further configured the PTTD algorithm as follows: $\epsilon = 0$, $d_{max} = 5$, $B = 1$.

The results of the experiments are shown in Table 4.2.

	PTTD	Tie	PTTD-CI
Accuracy	22	1	17
Runtime	13	0	27

Table 4.1: Wins of PTTD vs. PTTD-CI in terms of predictive accuracy and training runtime.

The results can be summarized by means of a win-loss statistics (see Table 4.1). In terms of predictive accuracy, the original operator set variant PTTD has the edge over PTTD-CI. However, the difference (22 vs. 17 wins) is not significant. Regarding training runtime, the difference is bigger, this time in favor of PTTD-CI. When conducting a Wilcoxon signed-rank test we found that the difference is significant even for a confidence level of 0.99 yielding a p-value of 1.23E-3.

To some extend, this result is astonishing. Due to the greater expressiveness of the CI operator one could have expected more accurate results. However, keeping in mind that we measure generalization error instead of training error, the effects of a higher flexibility and a higher risk of overfitting might have canceled out each other.

	Accuracy		Runtime (s)	
No.	PTTD	PTTD-CI	PTTD	PTTD-CI
1	76.32 ± 2.96	76.40 ± 2.60	4.08E+03 ± 1.13E+03	2.33E+03 ± 5.77E+02
2	73.93 ± 12.29	73.93 ± 12.83	5.27E+02 ± 2.61E+02	4.93E+02 ± 1.98E+02
3	33.42 ± 6.52	34.58 ± 4.96	1.47E+02 ± 7.25E+01	1.27E+02 ± 1.00E+02
4	92.41 ± 8.23	81.70 ± 8.03	1.53E+03 ± 1.47E+03	1.89E+03 ± 2.32E+03
5	96.95 ± 3.24	96.70 ± 2.80	8.48E+00 ± 5.10E+00	7.01E+00 ± 9.64E+00
6	85.65 ± 3.66	85.46 ± 3.68	4.71E+03 ± 2.18E+03	4.13E+03 ± 1.40E+03
7	99.66 ± 0.57	98.82 ± 0.00	3.02E+05 ± 9.43E+04	2.96E+05 ± 4.51E+04
8	72.17 ± 8.62	73.02 ± 9.15	1.86E+04 ± 4.28E+03	1.42E+04 ± 3.59E+03
9	90.99 ± 1.08	91.84 ± 0.76	8.05E+02 ± 3.64E+02	7.63E+02 ± 2.28E+02
10	87.71 ± 7.93	89.13 ± 7.39	4.11E+01 ± 5.73E+01	5.22E+01 ± 9.19E+01
11	78.83 ± 2.91	79.01 ± 2.77	5.19E+01 ± 4.00E+01	4.69E+01 ± 4.05E+01
12	71.24 ± 8.45	69.69 ± 8.58	3.02E+01 ± 2.65E+01	3.84E+01 ± 3.13E+01
13	96.64 ± 1.96	96.25 ± 2.26	1.26E+03 ± 4.57E+02	3.56E+02 ± 4.67E+02
14	71.92 ± 4.56	73.55 ± 5.12	2.20E+02 ± 1.20E+02	3.12E+02 ± 1.21E+02
15	32.33 ± 11.33	34.55 ± 13.76	1.90E+02 ± 7.32E+01	1.63E+02 ± 7.21E+01
16	54.99 ± 3.97	54.40 ± 3.46	1.23E+04 ± 4.31E+03	9.70E+03 ± 4.02E+03
17	86.31 ± 10.88	85.77 ± 10.20	8.01E+00 ± 2.96E+00	8.34E+00 ± 3.34E+00
18	86.47 ± 3.60	86.04 ± 3.33	8.05E+03 ± 3.35E+03	6.30E+03 ± 2.97E+03
19	78.02 ± 14.54	39.76 ± 24.94	6.02E+02 ± 2.59E+02	1.99E+02 ± 8.37E+01
20	65.02 ± 10.03	63.32 ± 9.38	2.60E+04 ± 5.21E+03	1.48E+04 ± 3.71E+03
21	82.22 ± 1.18	81.94 ± 1.35	1.95E+04 ± 5.14E+03	1.01E+04 ± 2.79E+03
22	74.13 ± 3.58	73.67 ± 3.48	3.12E+04 ± 8.88E+03	3.15E+04 ± 1.29E+04
23	68.07 ± 7.54	67.73 ± 8.56	1.02E+03 ± 2.79E+02	1.04E+03 ± 2.53E+02
24	72.97 ± 2.17	73.20 ± 2.36	9.53E+00 ± 7.70E+00	1.27E+01 ± 1.42E+01
25	81.60 ± 7.37	80.25 ± 6.97	1.30E+03 ± 7.29E+02	1.00E+03 ± 4.93E+02
26	92.02 ± 5.00	92.12 ± 4.99	4.64E+03 ± 3.67E+03	3.69E+03 ± 3.08E+03
27	95.78 ± 4.79	96.22 ± 4.53	2.32E+01 ± 1.61E+01	2.73E+01 ± 1.42E+01
28	98.33 ± 2.17	98.60 ± 2.04	8.12E+01 ± 4.25E+01	2.67E+01 ± 4.69E+00
29	76.53 ± 14.99	76.16 ± 15.65	3.91E+00 ± 7.04E+00	2.21E+00 ± 2.89E+00
30	84.48 ± 7.85	83.54 ± 9.14	3.22E+03 ± 8.25E+02	2.66E+03 ± 9.70E+02
31	35.41 ± 6.68	35.01 ± 5.76	9.99E+01 ± 2.90E+01	1.02E+02 ± 3.43E+01
32	77.26 ± 3.55	76.61 ± 4.24	7.65E+02 ± 5.63E+02	5.17E+02 ± 3.44E+02
33	42.28 ± 5.71	42.00 ± 6.26	1.79E+04 ± 3.72E+03	7.58E+03 ± 1.60E+03
34	78.00 ± 11.26	78.67 ± 11.96	5.49E-01 ± 2.75E+00	2.03E+00 ± 8.70E+00
35	82.00 ± 6.01	84.93 ± 6.19	1.98E+00 ± 8.26E+00	1.22E+00 ± 2.02E+00
36	65.88 ± 7.31	65.10 ± 8.45	8.31E+02 ± 3.94E+02	7.84E+02 ± 6.59E+02
37	83.67 ± 7.15	85.11 ± 6.29	3.05E+04 ± 1.52E+04	2.78E+04 ± 1.62E+04
38	72.47 ± 3.98	73.25 ± 3.90	1.30E+04 ± 3.94E+03	1.53E+04 ± 3.98E+03
39	98.49 ± 2.94	98.88 ± 2.28	8.32E+02 ± 3.62E+02	8.41E+02 ± 3.62E+02
40	96.03 ± 6.19	93.79 ± 7.96	9.75E+01 ± 2.58E+01	9.92E+00 ± 6.88E+00

Table 4.2: Predictive accuracy and training runtime results of PTTD and PTTD-CI for 40 benchmark datasets.

4.2 Comparing the Main Variants

Because of the rather large amount different variants covered in this work, we restrict the comparison to only three of them, one for each main algorithm: bottom-up, top-down and co-evolutionary. For each of these algorithms, we select a configuration, which arguably best trades off the accuracy of the resulting models and the runtime of the algorithm. In order to provide a fair comparison, we optimally select one parameter by means of an internal cross-validation procedure, namely the maximum depth d_{max} of the resulting trees. The following configurations have been chosen:

PTBU : This variant corresponds to the algorithm listed in Figure 3.1. Additionally, as the experiments in Section 3.5.4 suggest, we used the limited candidate history and the dynamic operator exclusion heuristic. The parameter τ for the DOE heuristic was set to 0.1, the LCH parameter k was set to 5.

PTTD : This variant corresponds to the algorithm listed in Figure 3.9. However, we used the potential heuristic elaborated on in Section 3.6.4 with $p = 3$.

PTCoEvo : The co-evolutionary algorithm presented in Section 3.7 was parameterized as follows: $t_{max} = 5000$, $t_{improve} = 500$, $\delta = 0.01$, $m = 20$, $p_{mutate} = 0.3$.

The experiments were again conducted by means of a three times 10-fold cross-validation. For each algorithm and dataset, the empirically best value for d_{max} was found, searching in the range of $[2, 10]$. The detailed results can be found in Appendix D.

To allow for a comparison in terms of two opposed criteria, Figures 4.1, 4.2 and 4.3 show the runtime on a logarithmic scale of seconds and the classifiers classification error. Each figure relates to one of three pairwise comparisons of the three methods. For each dataset there exists an arrow directing from PTBU to PTTD, from PTBU to PTCoEvo and from PTTD to PTCoEvo, respectively. The color of the arrows encode the following cases:

red : The arrow directs down-left. This corresponds to a Pareto dominance of the second method, being faster and more accurate.

violet : The arrow directs up-right, what corresponds to a Pareto dominance of the first method.

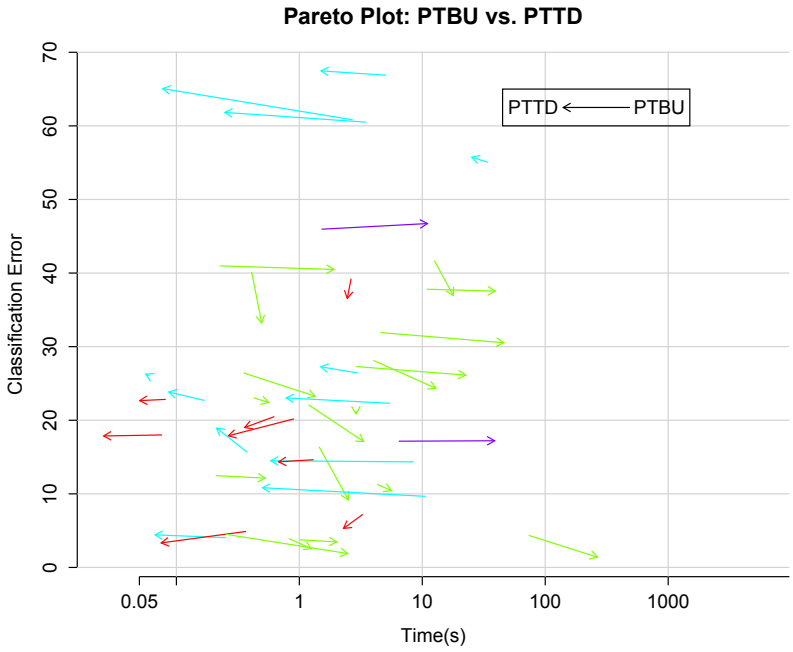


Figure 4.1: Pareto comparison of PTBU and PTTD.

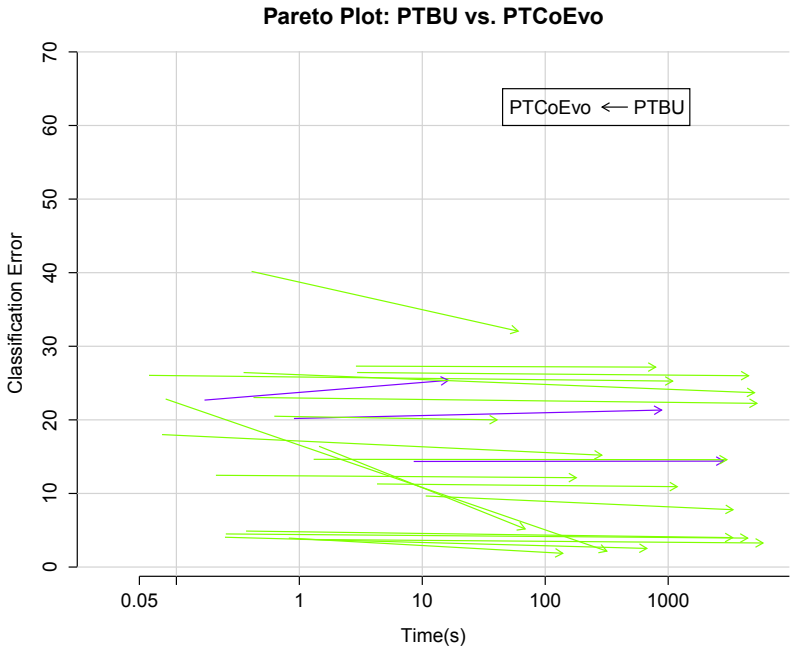


Figure 4.2: Pareto comparison of PTBU and PTCoEvo.

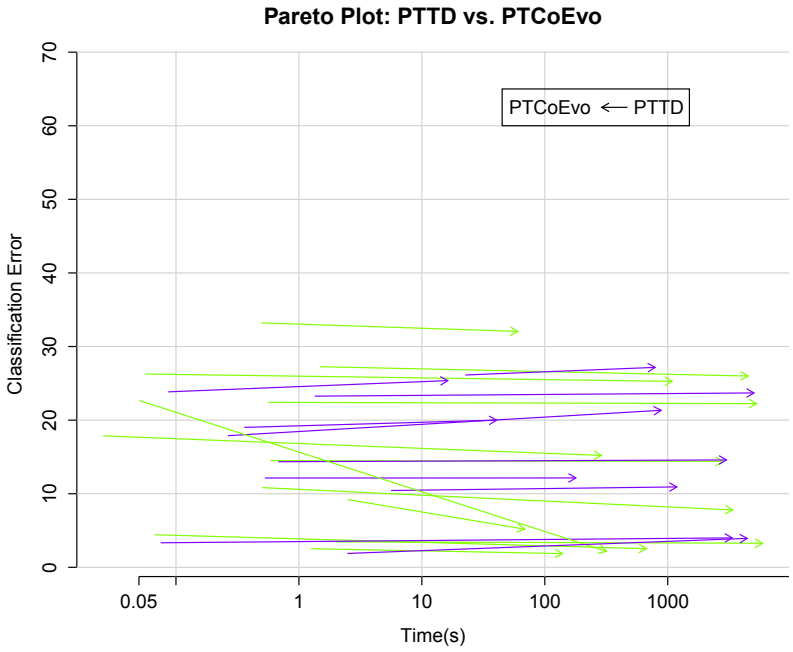


Figure 4.3: Pareto comparison of PTTD and PTCoEvo.

green : The arrow directs down-right. This corresponds to the non-dominance situation, where the second method is more accurate but also slower.

blue : The arrow directs up-left, what corresponds to the second non-dominance situation. This time, the second method is less accurate but faster.

Looking at Figure 4.1 (PTBU $\rightarrow$ PTTD), most arrows (26/40) aim downwards, indicating a better accuracy for PTTD. However out of these, most arrows (18/26) are green, what means that PTTD has a higher runtime than PTBU on these datasets. For 10 datasets, we can find Pareto-dominance, 2 times in favor of PTBU and 8 times in favor of PTTD. Summarizing these results, one can state that PTTD usually finds models with a lower error but in turn also exhibits a higher runtime.

The main drawback of the co-evolutionary approach (PTCoEvo) is its runtime. This is also reflected in the experiments. One of the crucial reasons is the evaluation step for each generation. During this step, m^k different classifiers have to be evaluated. Since the runtime of the proposed EA exponentially depends on the number of classes, we restricted the experiments to those datasets with k smaller than 4. Additionally,

we only used datasets with less than 1000 instances and at most 10 attributes. In the end, we used 22 out of the 40 classification datasets.

In Figure 4.2 the results of comparing PTBU and PTCoEvo (PTBU → PTCoEvo) are shown. For every dataset PTCoEvo needs much more time to train the models, however in almost every case (19/22) the evolutionary search finds more accurate models than PTBU. In two cases, the difference in accuracy is even extreme. This is first prnn_crabs, where PTCoEvo is 20.6 percentage points more accurate than PTBU and second analcatdata_homerun, with a difference of 11.6. One explanation for this extreme finding could be, that the greedy search strategy of PTBU might got stuck in a local minimum in these cases.

In the third comparison, shown in Figure 4.3, the picture is not as clear in terms of accuracy. In fact, both methods, PTTD and PTCoEvo perform equally well: 10 wins for PTCoEvo and 12 wins for PTTD. However, regarding the runtime, PTCoEvo again needs several levels of magnitude more time. Again, especially interesting is the huge difference in accuracy on the two datasets already mentioned above. Although for analcatdata_homerun, the difference has reduced to 4.02, the difference remains at the very high level of 20.6 for prnn_crabs.

In general, the high accuracy of PTTD together with its relatively low runtime accentuates PTTD. To some extend, it is astonishing that a greedy search strategy like the one employed in PTTD is able to compete with the evolutionary approach, spending much more search effort indeed to find a better solution. This however might be explained by the flexibility of the model class. Although in one iteration, the greedy search of PTTD might take a suboptimal decision, in later iterations the algorithm seems to be able to compensate for this.

This finding is in line with results on *over-searching* related to decision tree [69, 82] and rule-based systems induction [55]. Roughly speaking, these results suggest that it is not always beneficial to increase the search effort if you are interested in generalization performance. When comparing the greedy PTTD algorithm to PTCoEvo, which spends a lot more effort in searching for a good model, in many (not all) cases this does not seem to be beneficial.

To finalize this empirical analysis, for the method pairs PTTD vs. PTBU and PTCoEvo vs. PTBU, a paired Wilcoxon signed rank test [111] was conducted using the results of the three methods on the 40 datasets. The test provides information

about the statistical significance of the difference between the methods in terms of accuracy. The first comparison, i.e. PTTD vs. PTBU, results in a p-value of 0.01651. Deciding for a significance level of 0.05, this rejects the null-hypothesis of equality. For the second comparison, the p-value has been found to be 1.26E-3, also indicating a significant difference. Being strict on statistical testing methodology, we have to account for multiple testing. The most conservative way of doing so is to use the *Bonferroni correction* [30], which divides the significance level by the number of tests conducted. In this case, this would yield a corrected confidence level of 0.025, which is still bigger than both p-values found.

4.3 Comparison with State-of-the-art Methods

Huang et. al. in [49] and Senge et. al. in [93] conducted an empirical comparison of fuzzy pattern trees to several state-of-the-art classification methods. In the first place, the aim of such a general comparison is not to show superiority of a new method, but only general competitiveness. In practice, whenever there is a real world problem to solve, the selection of a suitable algorithm does not depend on an average performance on a set of arbitrarily chosen datasets. Rather, besides ensuring potentially important model class properties like interpretability, monotonicity and others, only the best performing algorithm will be chosen. This is in line with the "no free lunch" theorems of optimization [113] and machine learning [34], which roughly state, that there can not be a single algorithm providing the best solutions for every problem or dataset.

Nevertheless, if an algorithm is competitive in terms of an average performance over a variety of datasets, this at least advises the algorithm to be a considerable candidate, when one has to chose the best algorithm for a given problem. Because we already have compared several variants of FPT induction algorithms, we chose PTTDϵ.25 configured like in Section 3.6.2. We did not perform a dataset specific parameter optimization to have a fair comparison.

In the experiment, we included a number of well-known classification methods from the field of machine learning as baselines: The C4.5 decision tree learner (C4.5) [85], nearest neighbor classification (NN) [1], support vector machines with linear and RBF kernel[1] (SVM), the RIPPER (Repeated Incremental Pruning to Produce Error Re-

[1]Implemented by Platt's sequential minimal optimization [77].

duction) rule learner [24]. All these algorithms are also implemented in WEKA, and we used these implementations with their default parameterization.[1] Additionally, we included the fuzzy rule learner SLAVE [39, 40]. An implementation of this algorithm is offered by the KEEL suite [2], another machine learning framework. We ported this implementation to WEKA.[2]

To compare multiple classifiers with a control classifier, García et. al. [37] suggest the use of a two-step procedure. First, the non-parametric Quade test [81] is conducted in order to test the null-hypothesis of equal performance of all classifiers. In case this hypothesis is rejected, i.e., if there are statistically significant differences between the methods, the Holm post hoc test [30, 47] is used to analyze the differences between PTTDϵ.25 and the benchmark classifiers in a pairwise manner. Both tests are based on the ranks of the methods: For each dataset, the methods are sorted according to their performance, i.e., each method is assigned a rank (in case of ties, average ranks are assigned). After that, the ranks are weighted, respectively, by the difference between the minimum and the maximum performance values for each dataset. The average weighted ranks are shown in Table 4.4.

In our case, the Quade test rejects the null-hypothesis quite clearly with a p-value of 2.7E-16. The results of the pairwise comparisons between PTTDϵ.25 and the baseline classifiers using the Holm test are also summarized in Table 4.4. As can be seen, our PTTDϵ.25 learner shows the best performance on average, though except for SLAVE, the differences are not statistically significant.

[1] The following parameters are used in WEKA: C4.5: -C 0.25 -M 2; RIPPER: -F 3 -N 2.0 -O 2 -S 1; NN: -K 3 -W 0 -A LinearNNSearch -A "EuclideanDistance -R first-last"; SMO: -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "PolyKernel -C 250007 -E 1.0"

[2] We used the following parameter setting: 5 fuzzy sets, 500 iterations without change, mutation probability 0.01, use weights, population size 100.

Table 4.3: Average classification rates and standard deviation on test sets.

No.	SVM	NN	RIPPER	C4.5	SLAVE	PTTDϵ.25
1	78.25±1.82	75.23±4.14	76.4±2.25	76.2±3.28	77.04±6.63	75.82±2.48
2	77.47±9.39	74.96±9.46	79.4±11.79	80.8±9.16	75.78±13.63	75.96±11.16
3	19.4±5.62	7.4±3.64	38.65±6.6	37.1±5.8	25±5.71	39.5±6.42
4	88.15±7.61	51.82±11.19	95.53±6.09	95.57±6.01	74.98±10.77	90.95±7.83
5	93.87±1.99	97.49±2.7	97.73±2.75	98.26±2.55	93.48±5.48	94.77±2.4
6	84.84±3.65	85.48±4.01	85.74±4.01	85.86±4.05	85.51±3.9	85.57±3.6
7	99.67±0.63	99.52±0.79	94.08±2.41	94.24±2.49	76.74±5.58	99.07±0.9
8	72.25±10.6	68.48±10.61	72.98±9.8	80.3±9.93	28.22±11.88	74.65±9.51
9	87.78±2.16	86.91±2.71	80.6±3.54	77.89±3.52	68.4±6.54	87.29±3.32
10	86.61±6.83	90.06±6.25	89.86±5.71	88.01±8.69	78.17±9.87	88.5±6.82
11	76.15±0.47	74.92±3.57	77.86±3.33	77.64±3.44	76.01±4.62	77.03±1.84
12	58.04±1.15	62.67±8	67.09±7.47	66.38±7.7	56.98±6.13	70.69±7.04
13	97.07±2	96.87±2.13	96.16±2.45	95.64±2.53	89.55±4.42	96.55±2.19
14	73.3±5.84	72.76±4.93	78.81±4.98	84.68±5.36	63.36±6.68	72.6±4.35
15	29.67±9.79	21.56±9.82	50.65±16.68	71.69±17.73	17.85±11.72	41.25±14.41
16	48.56±3.93	46.8±3.05	52.54±3.48	51.57±3.53	42.7±3.63	55.49±3.29
17	64.57±11.66	83.54±10.3	75.39±14.55	77.86±12.95	63.86±21.44	87.11±9.26
18	85.01±3.7	85.94±3.82	85.3±3.34	85.68±3.69	85.3±4.37	85.42±3.68
19	73.71±8.96	81.33±12.58	80.05±13.41	81.29±15.16	71.71±16.41	83.24±11.73
20	61.37±10.36	54.87±10.67	61.65±8.32	64.73±9.47	25.01±10.93	68.29±7.47
21	83.01±0.36	81.9±1.14	82.74±0.76	82.93±0.35	83.01±3.33	82.74±0.75
22	75.08±3.53	72.06±3.28	72.2±4.23	71.38±2.7	70.18±4.54	74.28±3.35
23	57.81±7.92	70.12±6.84	66.59±9.58	69.05±8.01	62.61±11.42	67.51±8.34
24	73.46±1.01	69.74±4.48	72.04±5.61	71.12±5.3	73.93±7.55	74.64±5.85
25	83.85±5.96	78.59±6.08	79.85±6.25	78.44±6.4	71.78±8.09	82.59±5.51
26	88.16±4.99	86.33±4.85	88.9±4.85	89.63±4.8	80.92±6.04	91.97±4.87
27	96.27±4.84	95.2±5.5	93.47±8.69	94.93±5.43	92.13±8.16	95.6±5.6
28	98.8±1.6	98.72±1.59	98.8±1.6	98.8±1.6	98.12±1.78	98.68±1.86
29	75.61±14.1	74.81±16.66	75.44±12.42	77.53±12.07	75.75±16.57	77±12.85
30	85.91±7.45	81.14±8.28	75.36±12.18	75.7±11.62	69.27±12.47	83.8±7.61
31	27.62±4.65	39.42±6.69	36.48±6.86	39.95±7.84	8.99±6.02	38.11±6.89
32	76.85±4.25	74.24±4.65	75.62±4.71	74.85±5.35	66.3±5.01	76.49±4.92
33	46.37±5.68	39.65±5.4	39.41±5.42	40.59±5.64	24.83±6.85	43.18±6.13
34	96±4	95.5±4.5	90.6±6.61	93.7±4.88	61.7±12.4	77.5±11.01
35	85.6±6.74	86.32±7.25	83.68±7.37	84.48±7.14	79.52±8.07	83.76±6.9
36	59.24±5.85	64.29±9.3	81.24±8.3	80.76±6.66	50.47±7.82	67.59±7.34
37	76.27±8.95	83.36±8.46	73.12±10.12	73.86±8.46	63.48±13.44	80.38±8.92
38	74.37±3.89	71.03±3.84	68.53±4.35	71.92±4.79	29.47±4.59	71.47±3.39
39	99±2.41	95.97±4.03	93.58±7.13	93.59±5.42	81.23±8.94	98.31±2.81
40	93.65±6.53	92.49±7.56	88.93±8.73	92.49±7.01	86.98±11.01	91.07±6.65

Algorithm	Avg. Rank	p-Value	Reject
PTTDϵ.25	2.4902	—	—
C4.5	2.6932	0.7890	no
SVM	3.0792	0.4377	no
RIPPER	3.3939	0.2338	no
NN	3.6567	0.1243	no
SLAVE	5.6865	2.54E-5	yes

Table 4.4: Average ranks of the algorithms (Quade) and results of the Holm test (p-value and rejection of null hypothesis at the 5% significance level)

4.4 Comparison with State-of-the-art Methods for Regression

In this section, we adapt the method of pattern tree induction so as to make it applicable to another learning task, namely regression. Thus, instead of predicting one among a finite number of discrete class labels, the problem is to predict a real-valued target output. We have already seen, that in theory FPT is able to approximate every real-valued function (Section 2.4), here we are showing that the PTTD algorithm is indeed able to find such trees.

As described in Section 3.1, for regression it is possible to choose a simple linear scaling of the output variable into the interval $[0, 1]$ (see Equation 3.4). This approach will be denoted as PTTDreg. Additionally, we include a fuzzy systems variant PTTDsys, partitioning the domain of the output variable in a uniform way by means of three triangular fuzzy sets; the first with core $\{a\}$ and support (a, b), the second with core $\{b\}$ and support (a, c), and the third with core $\{c\}$ and support (b, c), where $[a, c]$ is the (observed) domain of the output and $b = (a + c)/2$). PTTDreg and PTTDsys were run with the adaptive termination criterion with $\epsilon = 0.001$ and $\epsilon = 0.0025$, respectively.

For comparison, we included several other regression methods implemented in WEKA: Standard linear regression (LR), multilayer perceptrons (MLP), support vector machines with linear (SVMlin) and RBF kernel (SVMrbf), and regression trees (REPtree); all these algorithms are used with their default parameterization.[1] Finally, we also included the fuzzy rule learner (FR) proposed in [109], which is implemented in the KEEL software [2] and which we ported out to WEKA. The datasets used in the experiments are shown in Table 3.2.

Table 4.5 provides a summary of the results in terms of the RMSE (root mean squared error). As can be seen, the pattern tree learners are quite competitive. In terms of average ranks, PTTDreg is even the best method.

To test for statistical significance of the differences, we compared the methods in a pairwise way by means of a Wilcoxon test. It turns out, however, that the test fails to reject the null-hypothesis of equal performance most of the time, maybe due to

[1] LR: -S 0 -R 1.0E-8; MLP: -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a; SVM-lin -C 1.0 -N 0 -I "RegSMOImproved -L 0.001 -W 1 -P 1.0E-12 -T 0.001 -V" -K "PolyKernel -C 250007 -E 1.0"; SVM-rbf: -C 1.0 -N 0 -I "RegSMOImproved -L 0.001 -W 1 -P 1.0E-12 -T 0.001 -V" -K "RBFKernel -C 250007 -G 0.01"; REPtree: -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

the limited number of datasets included in this study. An exception is the fuzzy rule learner, which performs significantly worse than both PTTDreg and PTTDsys (at a significance level of 5%). In any case, the results suggest that pattern tree learning is fully competitive to state-of-the-art regression methods in terms of predictive accuracy.

No.	LR	MLP	SVMlin	SVMrbf
44	3.35(5)	3.24(3)	3.43(6)	3.54(7)
45	10.47(5)	8.01(3)	10.91(7)	10.8(6)
46	0.44(1)	0.68(7)	0.46(5)	0.44(3)
47	0.77(1)	0.99(7)	0.83(5)	0.83(6)
48	47.6(4)	82.89(8)	46.56(2)	46.51(1)
49	4.87(5)	4.09(1)	4.95(6)	5.5(7)
50	2665.71(3)	2660.27(2)	2566.86(1)	2921.9(6)
51	145.43(6)	160.43(8)	62.72(1)	79.2(5)
52	1.11(5)	0.56(1)	1.27(7)	1.44(8)
53	2.7(2)	0.67(1)	2.77(4)	4.94(6)
54	0.65(2)	0.73(7)	0.66(3)	0.66(4)
55	0.75(2)	0.79(6)	0.76(3)	0.76(5)
avg. rank	3.42	4.50	4.17	5.33

No.	PTTDreg	REPtree	PTTDsys	FR
44	2.94(1)	3.34(4)	2.95(2)	5.87(8)
45	7.97(2)	7.21(1)	9.08(4)	13.7(8)
46	0.47(6)	0.44(2)	0.45(4)	0.83(8)
47	0.83(4)	0.77(2)	0.79(3)	1.15(8)
48	61.67(6)	46.86(3)	52.61(5)	75.67(7)
49	4.28(2)	4.56(3)	4.66(4)	7.43(8)
50	2876(5)	3560.71(7)	2825.72(4)	8198.64(8)
51	69.6(2)	147.82(7)	72.08(3)	78.23(4)
52	0.68(2)	0.74(3)	0.76(4)	1.17(6)
53	2.73(3)	5.04(7)	3.45(5)	7.41(8)
54	0.65(1)	0.68(6)	0.67(5)	1.13(8)
55	0.76(4)	0.75(1)	0.79(7)	1.3(8)
avg. rank	3.17	3.83	4.17	7.42

Table 4.5: Experimental results in terms of RMSE. Additionally, for each dataset, the rank of each method is shown in brackets.

5

Related Model Classes

In the last chapters, FPTs have been carefully introduced and discussed. This chapter, however, deals with related model classes. Many model classes actually are related to some extend but we will focus on the arguably most related ones. These are:

- (hierarchical) fuzzy rule-based systems ((h)FRBS)
- fuzzy decision trees (FDT)
- sum-product networks (SPN)
- genetic programming (GP)

Table 5.1 already gives an overview about some distinctive properties of the above model classes. They will be explained in detail in the remainder of this chapter.

	Structure	Information Flow	Operators	Building
FPT	hierarchical	bottom-up	t-norms, t-conorms, averaging	expert modeling / PTBU,PTTD, PTCoEvo
FRBS	flat	–	t-norms, t-conorms	expert modeling iterative covering EA
hFRBS	modular hierarchical	bottom-up	t-norms, t-conorms	expert modeling iterative covering EA
FDT	hierarchical	top-down	fuzzy predicates	recursive partitioning
SPN	hierarchical	bottom-up	sum, product	back-propagation EM
GP	hierarchical	bottom-up	arbitrary/ often arithmetic	genetic algorithm

Table 5.1: An overview of some properties of the related model classes discussed in this chapter.

5.1 Fuzzy Rule-based Systems

After Zadeh had introduced fuzzy set theory, the first fuzzy systems that were proposed have been *fuzzy rule-based systems*, using IF-THEN rules to describe the relationship between the input and the output variables of a control system [75].

Many different variations have been proposed. Two of the most popular ones are the one by Mamdani [63] and another one by Takagi and Sugeno [99], where the latter one follows a slightly more complex scheme. Mamdani rules, however, follow the simple syntax

IF X is A **and** ... **and** Y is B **THEN** Z is C ,

where X,Y are input attributes, Z is the output attribute and A, B and C are fuzzy sets. The rule basically consists of two parts, the *antecedent* (IF part) and the *consequence* (THEN part). The antecedent determines, to which part of the instance space this rule is applied and to what extend. It basically constitutes a fuzzy subset of the instance space. The consequence determines its conclusion. Roughly speaking, if X belongs to fuzzy set A and Y belongs to fuzzy set B, then Z is predicted to be in C. In a system with many rules, potentially many rules are applicable (antecedent fuzzy set membership is greater than zero) in parallel for a specific query instance and the respective conclusions are calculated. Therewith, the fulfillment of the antecedent is taken into account, when aggregating the consequence fuzzy sets of these rules.

Just like for FPTs, rule-based fuzzy systems require a fuzzification and a defuzzification step. The fuzzification step either constitutes the modeling by an expert to create suitable fuzzy sets or fuzzy partitions [58, 63] or the use of unsupervised learning techniques like *fuzzy k-means* clustering [20, 38]. Evaluating the rulebase, we obtain a fuzzy output from aggregating the consequence fuzzy sets of the applicable rules. Defuzzification methods then take the resulting fuzzy set as an input and produce a crisp value to predict, which is required in most predictive applications. Several defuzzification methods have been proposed [58]. The maximum approach e.g. takes the value for which the fuzzy set produces the highest membership. In case this value is not unique the mean of all that have been found is taken. Another possibility is the use of the *centroid* method. This method returns the value that splits the area under the fuzzy set in half.

Besides modeling rule-based systems by hand, several induction algorithms have been proposed. Many have been motivated by existing algorithms designed for non-fuzzy rule-based systems [24]. Approaches like [25, 28, 39, 50] involve several common steps. The first step usually comprises an iterative local covering strategy that generates an initial set of rules. After finding a rule that explains a part of the training examples, these examples are either removed from the training set or their weight is reduced. This reduces their influence in subsequent iterations. The second step then deletes or adjusts contradicting rules. This step is called the *pruning* step and is necessary because the first step does not account for potential contradictions in the rulebase. The final step then tunes the parameters of fuzzy sets or adjusts the antecedents of single rules to minimize generalization error. Each algorithm implements these three steps differently. Often, however, evolutionary or genetic algorithms are involved.

A Comparison to FPTs

In Section 3.1, we have already described an extension of the original FPT approach in which a model is specified in terms of an ensemble of pattern trees. For instance, instead of using a single tree in our wine example shown in Figure 3.17, which specifies the conditions for a "high quality" wine (and immediately implies a low quality if these conditions are not satisfied), one may add corresponding trees for "medium quality" and "low quality". To make an overall prediction, it is then necessary to combine the outputs of these trees. This approach is close to rule-based fuzzy systems: each FPT can

be associated with a rule or a set of rules with the same consequence; the aggregation of the outputs of the trees then essentially corresponds to the step of defuzzification in rule-based systems. In fact, using this procedure, every fuzzy rule-based system can be transformed into an ensemble of FPTs. Hence, rule-based systems are a special case of fuzzy pattern trees, or stated differently, FPTs are a generalization of rule-based systems.

Even more, rule-based systems typically do not involve averaging operators which adds valuable expressiveness to FPTs in comparison to fuzzy rule-based systems. Averaging operators are particularly helpful in situations where linear relationships have to be modeled. At first sight, this does not seem to be a hard task, though it is for rule-based systems. Since a single rule typically only describes a (usually small) local region of the input space, usually many rules are needed to accurately model a global linear relationship. Therefore, the ability of FPTs to incorporate averaging operators eliminates one of the weak points of rule-based systems.

One may argue that the use of different types of aggregation operators may compromise the interpretability of an FPT. In our opinion, however, this is not the case: Each operator itself is easily interpretable, and thanks to the modularity of an FPT, it can indeed be considered completely independent from the rest of the model.

Further comparing rule-based systems to FPTs, an obvious difference is the hierarchical structure of FPTs. A hierarchical instead of a flat structure often allows for representing models in a more compact way. One reason is that, in the class of FPTs, much more transformations between formally equivalent expressions are possible than in the class of rule models; for example, while the expression $\max\{\min\{A, B\}, \min\{A, C\}\}$ can be considered as a disjunction of two rules with antecedents $A \wedge B$ and $A \wedge C$, respectively, the same is not true for the logically equivalent expression $\min\{A, \max\{B, C\}\}$. Moreover, the class of analytical expressions that can be represented, as well as the "degree of nonlinearity", are significantly increased thanks to the possibility of recursion. For example, with an FPT of depth k it is possible to model all monomials of degree k by just using the simple product as a t-norm, even if all membership functions are linear. To make things worse, it has been shown that the number of rules necessary to cover the complete input space growth exponentially by the number of attributes (dimensions of the input space) [52].

In this regard, it is interesting to note that similar advantages of "deep" over "flat" structures have also been observed in other domains, currently for example in the field of *deep learning* [12]: Although it is true that neural networks with a single hidden layer exhibit universal approximation capabilities, the practical realization of this theoretical property may require an extremely large number of neurons in this layer. The same approximation quality might be achieved with a significantly smaller number of neurons if these are distributed on several layers and connected in a proper way. Likewise, the universal approximation property of fuzzy systems typically comes at the price of an excessively large number of rules.

5.2 Hierarchical Fuzzy Rule-based Systems

As pointed out above, the exponential growth of the number of rules is a problem for systems with a large number of variables. Motivated by this deficiency, several hierarchical fuzzy rule-based systems have been proposed. In these systems, rules are grouped into several modules according to their task in the system. See Figure 5.1 for two examples. Each module delivers a part of the solution which can also be used as an input for subsequent modules. Thereby, higher level modules aggregate the partial solutions into a final one. Torra [101] gives a brief overview on the different types of hierarchical structures, including [44, 56, 60, 97, 102]. Using a hierarchical structure, as already argued above, is one way of managing complexity. It enables reusability of modules and due to greater flexibility and expressiveness, the same functional relationship can be usually modeled in a more compact way.

Although, hierarchical fuzzy rule-based systems seem to be very similar to fuzzy pattern trees, there are some important differences. Indeed, every module in a hierarchical rulebase is by itself a rule-based system, containing a set of rules and using the same aggregation scheme as described earlier. Compared to an FPT, a single node in the hierarchy is therefore much more complex and harder to interpret than a single node in an FPT, either forming a simple aggregation of subtrees or a single fuzzy set of an attribute domain.

Torra [101] argues that the construction of a hFRBS is difficult, due to the experts task to both model the hierarchical structure and the rules within each module. In order to assist the expert, several algorithmic approaches have been proposed. Most

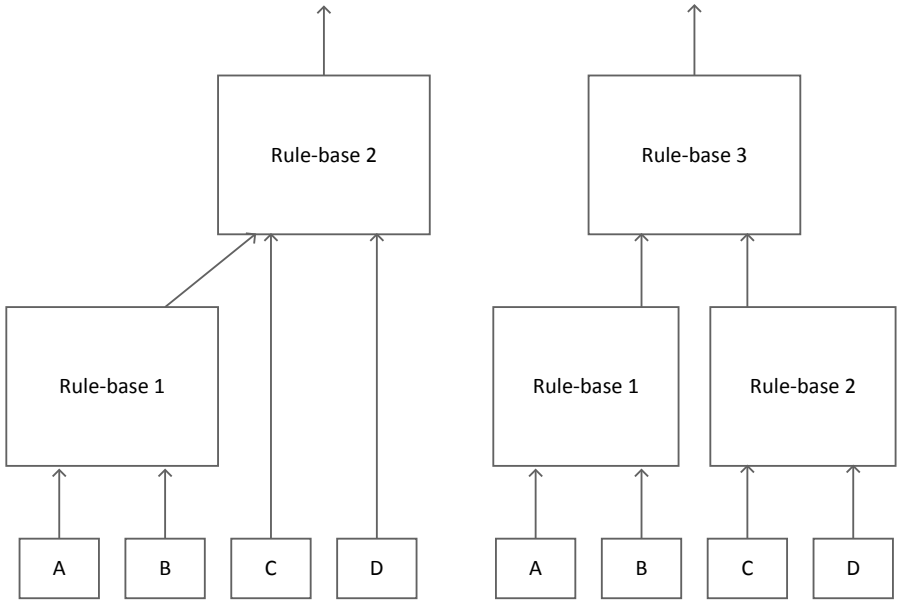


Figure 5.1: Two possible hierarchical structures for fuzzy rule-based systems.

approaches are not able to learn the whole system from scratch but rather depend on an expert to create the structure and rules of the system. Only then, they are used to tune the parameters of fuzzy sets or adapt rules. These methods differ by the extend of tuning and their optimization method used. For example, Cheong [22] uses an EA to tune the membership functions, whereas Lee [56] et al. and Wang [108] use gradient descent. Tunstel [102] even uses genetic programming (which we will discuss later) to create rules.

The hierarchical approach to FRBSs is supposed to mitigate the potential problem of a large number of rules for high-dimensional data. However, grouping rules into modules yields additional complexity to the model class that complicates interpretation and learning. Furthermore, designing a hierarchical rule-based system, the expert has to decide, whether the defuzzification step is included in a module or not. This decision is crucial to the interpretation of the output of the module and can also strongly influence the performance of the whole system. This has been analyzed by Maeda in [62]. Roughly speaking, if a defuzzification step is conducted within a module, this usually yields a loss of information. On the other hand, if no defuzzification is done,

this can lead to a spread of fuzziness and hence to very uninformative results.

5.3 Fuzzy Decision Trees

Fuzzy decision trees (FDT) [54, 74, 117] are inspired by the work of Quinlan about *decision trees* [83]. The output of a classical decision tree however is categorical or discrete. They miss to adequately handle uncertainty, which is commonly introduced by imprecise, noisy or missing data. Therefore, Quinlan suggested a probabilistic version of decision trees [84] in order to deal with noisy data. In his work, however, the type of uncertainty he takes care of is assumed to be caused by randomness. FDTs however, model uncertainties using fuzzy set theory, aiming at uncertainty, which arises from human thinking, reasoning and perception. Several approaches to the induction of fuzzy decision trees can be found in the literature. Most are extensions to classical decision tree algorithms like the one of Quinlan, called ID3 (*iterative dichotomiser 3*).

A regular decision tree constitutes a hierarchical structure like an FPT, however, they work quite differently. Each internal node represents an attribute. Every outbound edge directing to one of the nodes' children is labeled by a boolean predicate regarding the attribute. These predicates are mutually exclusive, i.e., only one condition can be fulfilled at a time. Each leaf node – in the case of classification – corresponds to a class label. Figure 5.2 illustrates an example, that has also been trained on the wine quality dataset introduced in Chapter 2.

In order to classify an instance, the tree is traversed in a top-down manner in the following way. Starting from the root node, each predicate is tested on the instance. The predicate that matches the instance determines the path to proceed on the way to one of the leaf nodes of the tree. The label of the leaf node then determines the "decision", i.e., class to predict. Although the given example refers to classification, several variants have been proposed for different types of learning scenarios like e.g. regression [27, 98].

Basically, decision trees split up instances into groups. These groups are determined by the leaf nodes of the tree. Every instance belongs to exactly one group, i.e., the group corresponding to the leaf node the instances ends up in when traversing the tree. Because a decision tree predicts the same class for every instance of the same group, these groups ideally are homogeneous in terms of their true class.

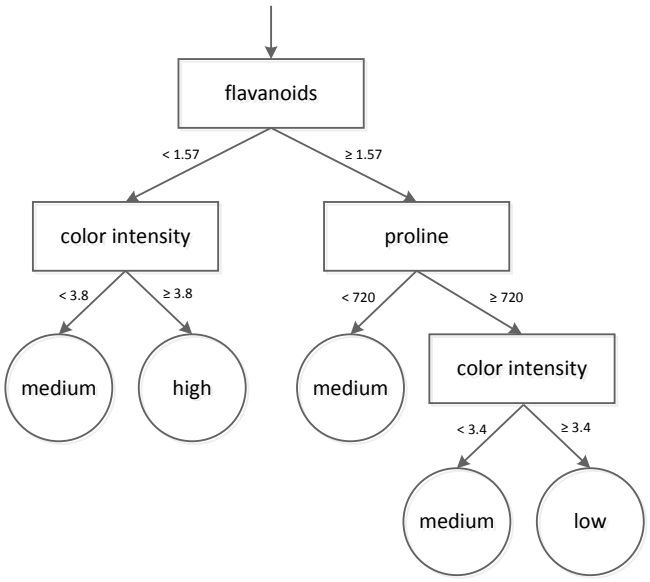


Figure 5.2: A classical (non-fuzzy) decision tree trained to predict the quality of wine. (See example introduced in Chapter 2.)

Keeping this in mind, the learning algorithm used for decision trees is straight forward. The algorithms starts with a single leaf node, representing the group of all instances and predicting the majority class. Then it finds a boolean predicate that best splits this group of instances in order to create maximally homogeneous groups. Quinlan proposed to use *information gain* for this purpose. It measures the increase of information introduced by a splitting predicate. Then this procedure is applied recursively to the newly created leaf nodes. Hence, the algorithm *recursively partitions* the input space.

Fuzzy decision trees extend regular decision trees by using fuzzy logical predicates instead of boolean predicates. To this end, the structure of the tree remains similar, however, the outbound edges of a node now represent a fuzzy set. Instances satisfy fuzzy predicates to a certain degree between 0 and 1 that is determined by the membership function of the respective fuzzy set. Furthermore, it is possible that there is more than exactly one fuzzy set for which the instance yields a positive membership degree. As a consequence, there is no longer just one unique path down the tree for each instance. Contrariwise, membership values of an instance are propagated down the

tree on potentially many paths ending up in potentially many different leaf nodes.

Another important difference refers to the leaf nodes. These represent fuzzy subsets of the domain of the target attribute. In order to infer a decision for a given instance, a similar mechanism is used like for fuzzy rule-based systems. First all fuzzy sets of those leaf nodes are aggregated that received a positive membership value. Then a defuzzification strategy is applied in order to produce a crisp prediction.

Several learning algorithms have been proposed for fuzzy decision trees [54, 74, 117]. Basically, they all share the same framework based on the original proposal described above. However, they vary in using different measures in order to extend the information gain measure to the fuzzy case.

Despite the obvious differences to FPTs, Huang et al. have shown in [49] that it is possible to transform every fuzzy decision tree into a set of fuzzy pattern trees.

5.4 Sum Product Networks

Sum product networks (SPN) have been introduced only recently by Poon et al. in [78]. The authors position them as being a compact and efficient alternative to graphical models [76]. SPNs comprise an acyclic network structure of several layers. Each inner node either represents a product or a sum operator. The input layer is formed by leaf nodes, which represent random variables and the output layer consists of a single root node. Edges link nodes indicating the flow of information, which is like for FPT bottom-up. However, unlike FPTs, SPNs comprise weights on the inbound edges of every sum operator node. Furthermore, the number of inbound edges to any node is not restricted.

The evaluation of an SPN is performed in a recursive way, just like for FPTs. However, the evaluation of a single node depends on its type. A sum node calculates the sum of its incoming values, weighted by their respective weights. To be more precise, the value of a sum node is:

$$\sum_{j \in Chi(i)} w_{ij} v_j \ ,$$

where $Chi(i)$ denote the children of the i-th node and v_j the value of the j-th node. w_{ij} then denotes the non-negative weight on the edge (i, j). The value of a product node, hence, is just the product of the values of its children.

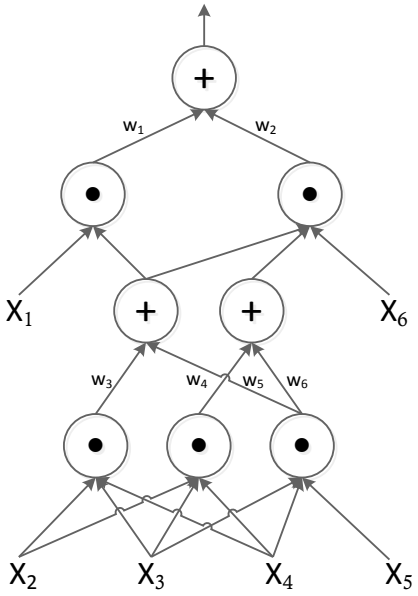


Figure 5.3: An example of a small SPN.

Poon et al. propose two possible learning methods for SPNs [78]. The first one is related to the well-known *back-propagation* algorithm of *multi-layer perceptrons* [43]. To this end, an SPN is initialized using a generic architecture, which can also be prescribed by an expert. Then, iteratively, the weights get adjusted until convergence. Updating the weights can either be accomplished by gradient descent like in back-propagation [88] or by expectation maximization (EM) [73]. In a post-processing step, all edges with weights equal to zero are removed. Then all non-root nodes without parents are removed. This way, weights and structure can be learned simultaneously.

5.5 Genetic Programming

The aim of *genetic programming* (GP) is to solve problems by automatically finding a computer program-based solution. To this end, the problem at hand is transformed into an optimization problem, for which then an evolutionary algorithm (EAs) [14] is used to solve it. The generic nature of EAs allow for applying GP to a large variety of programming languages. Traditionally, however, functional languages like LISP [96] or

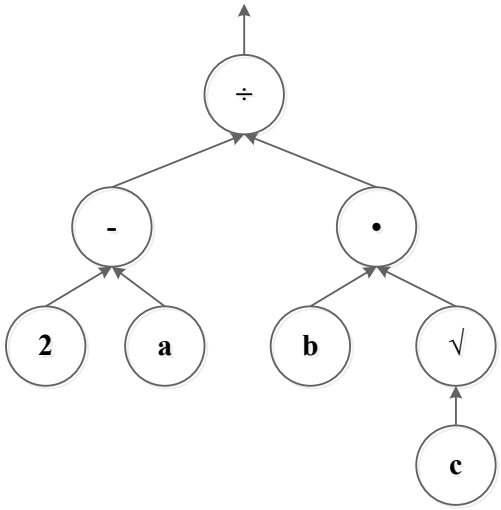


Figure 5.4: An example of a model generated by a genetic algorithm.

HASKELL [100] are preferred. Using these, programs can be written in a functional way, recursively calling to sub-functions and arguments. Then, programs can be represented in a tree structure, similar to the one of FPT. Inner nodes represent function calls taking their children as arguments. Leaf nodes either represent constant values or variables to the program.

In general, every kind of *basis functions* (i.e. functions, that are available to be used in inner nodes) is imaginable. In many practical cases, however, the selection of basis functions is limited to arithmetic operators. Figure 5.4 shows an example. Given such a set of operators, evolutionary strategies similar to the one described in Section 3.7 are used to find a well performing program. This involves cross-over as well as mutation operators.

Due to the generality of the GP approach, our co-evolutionary approach to FPT learning can roughly be seen as a specialization of GP, restricting the basis functions to fuzzy operators and membership functions and adding some additional limitations to the structure of the tree. Learning FPTs, however, because of its restrictions, can be accomplished more efficiently.

6

Conclusions and Outlook

In this work, we have elaborated on fuzzy pattern trees, a machine learning method using fuzzy set theory in order to produce interpretable predictive models. Being able to comprehend a prediction made by a machine enables experts to gain insight into the underlying relationships of the data generating process. This not only increases confidence in the decision of the machine, but also capacitates research in the domain of application.

The contributions of this work are manifold.

- In Sections 3.1 and 2.2.1, the model class itself has been extended. First we proposed a general way of fuzzifying input attributes into a comprehensive fuzzy partition. Second, new and more expressive operators have been introduced, in order to be able to create more flexible models.

- In the following sections of Chapter 2, we focused on important properties of the model class. Besides giving an overview of known properties, two new theoretical results have been presented. First, fuzzy pattern trees are universal approximators and hence are able to approximate any real-valued function on a compact set. Second, after utilizing the first result, it was shown that the VC dimension of FPTs is potentially infinite.

- The main contributions reside in Chapter 3. To start with, three heuristics are introduced, which aim at speeding up the induction method of Huang et. al. without loosing predictive accuracy. Then, two completely new algorithms for pattern tree induction are introduced. The first one (PTTD) reverses the construction of

the tree in comparison to PTBU. The second one constitutes a co-evolutionary algorithm (PTCoEvo).

- In order to make FPT induction applicable to nowadays "big data" problems, the idea of racing algorithms was used to provide scalability of the PTTD induction algorithm. Their effectiveness was demonstrated empirically by means of a real world case-study.
- FPTs were applied to regression problems as well as they were used for fuzzy systems modeling.
- All algorithms and modifications have been backed up with experiments which empirically show the effectiveness of the proposed methods.
- The fifth chapter elaborates on a comparison between FPT and related model classes. Comparing FPT to more or less similar model classes and their learning algorithms clearly reveals the benefits of FPTs.

Still there is a lot of work for the future. One line of research could aim at applying fuzzy pattern trees to different learning scenarios. Classification and regression are only the most basic ones. Due to the habile nature of the trees (comprising a fuzzy predicate, mapping from any type of input into $[0, 1]$), it is easy to think about applying them to several other tasks like multi-task learning, multi-label classification, ordinal regression, preference learning, label ranking, association analysis and many more.

A second line should deal with enabling the learning algorithms to take additional background knowledge into account. Such kind of knowledge could e.g. be introduced by constraints on the structure of the tree. One possibility is to group attributes into sets. These sets, then, form subtrees, which are aggregated only in a later stage. This could be useful in order to even increase interpretability. To give an example, imagine the task of predicting an overall evaluation of a car for a given customer. The customer might have different kinds of single criteria, which must be aggregated into an overall utility. Grouping these single criteria by categories like costs, motor, extras etc. should lead to subtrees only dealing with one of these categories. This would further improve the interpretability of the resulting models.

The third line of research could elaborate on utility-based preference learning [36], i.e., assigning a degree of *utility* to an option on the basis of which it is compared

to other options. Options in this regard usually are described by attributes just like regular instances. It is assumed that the utility of an option depends on its attributes. For example, imagine an online shop like Amazon (amazon.com) where it is possible to rate a product by means of a 5-star scale. The product, e.g. a notebook, is described by attributes like the type of CPU or the size of its hard disk. Learning a utility function in this scenario would yield a model that is predicting the utility of an unrated product given its properties.

In general, FPTs can be used to modeland learn such utility functions. The monotonicity property discussed in Section 2.3 is especially interesting in this regard. There are methods for learning so-called structured utility functions which require monotonicity and ensure additivity like the UTA method [42, 53]. Such properties are important in order to provide an interpretation of the learned models in terms of an aggregated utility function. The building blocks of such a function are the so-called *partial utility functions* that refer to the utility of a single attribute value or a subgroup of attribute values.

Extracting partial utility functions for single attributes is particularly interesting for producers. Knowing product properties that appear to be of high utility for buyers enables producers to tailor their products accordingly. From a theoretical perspective, however, this task is highly non-trivial. Nevertheless, due to the special properties of FPTs accomplishing this task comes into reach.

7

Appendices

Appendix A

All results of the study on surrogate loss functions of Section 3.4.

No.	ClassE	JacE	MAE	MSigE	RMSE
1	77.62 ±1.27	77.38 ±1.06	77.38 ±1.06	77.38 ±1.06	77.54 ±1.84
2	77.96 ±7.96	74.89 ±7.83	81.78 ±8.84	78.96 ±10.30	78.74 ±10.16
3	22.83 ±3.06	24.83 ±0.91	25.00 ±0.00	24.92 ±0.46	37.00 ±4.47
4	97.34 ±3.10	95.47 ±7.50	72.07 ±9.79	81.96 ±8.75	83.81 ±8.85
5	95.31 ±3.56	93.19 ±2.07	94.44 ±2.38	96.20 ±3.45	96.71 ±3.13
6	85.46 ±4.45	84.15 ±3.95	84.54 ±4.02	84.83 ±3.99	85.51 ±3.73
7	90.41 ±2.61	92.67 ±3.00	90.85 ±3.46	95.29 ±2.48	93.78 ±2.35
8	51.73 ±9.70	51.71 ±8.47	34.79 ±10.98	37.39 ±10.77	58.48 ±10.61
9	89.33 ±2.47	72.89 ±6.05	76.47 ±4.57	88.22 ±4.47	90.02 ±2.09
10	85.30 ±6.96	79.87 ±9.24	80.25 ±7.31	80.87 ±8.19	87.21 ±7.67
11	76.47 ±1.71	76.60 ±1.08	76.65 ±1.44	76.60 ±1.73	77.09 ±1.67
12	61.21 ±8.33	45.99 ±4.63	57.40 ±4.02	57.11 ±6.54	59.04 ±4.31
13	96.24 ±2.36	96.00 ±2.52	95.95 ±2.54	95.32 ±2.67	95.90 ±2.38
14	71.68 ±4.75	63.47 ±5.56	64.28 ±2.51	64.28 ±2.51	72.73 ±5.12
15	26.76 ±13.00	21.15 ±11.12	29.09 ±8.59	21.58 ±7.64	35.39 ±13.61
16	47.02 ±4.20	43.97 ±2.32	42.43 ±0.54	42.36 ±0.61	52.23 ±3.72
17	67.14 ±12.87	66.25 ±11.13	62.56 ±11.45	68.99 ±8.59	75.18 ±12.26
18	84.59 ±3.62	85.60 ±4.13	84.64 ±3.52	84.54 ±3.70	85.36 ±3.70
19	71.98 ±12.20	72.94 ±11.07	76.59 ±12.04	77.78 ±12.26	76.11 ±11.74
20	60.15 ±10.23	63.75 ±8.22	60.19 ±8.28	60.70 ±7.79	66.68 ±8.65
21	82.63 ±1.04	82.54 ±0.97	82.88 ±0.67	82.88 ±0.67	82.75 ±0.68
22	74.53 ±2.84	70.40 ±1.04	70.67 ±1.37	73.93 ±3.53	72.27 ±3.71
23	50.01 ±4.65	47.66 ±5.68	47.34 ±4.62	49.09 ±5.60	61.38 ±7.53
24	73.74 ±5.12	73.20 ±1.37	72.87 ±2.68	74.61 ±4.55	75.92 ±5.09
25	77.90 ±8.22	73.58 ±5.48	74.32 ±6.66	74.69 ±6.88	80.25 ±5.86
26	90.79 ±5.63	87.57 ±5.14	88.61 ±5.05	85.00 ±4.72	87.48 ±5.28
27	96.00 ±4.50	66.89 ±1.22	96.00 ±4.50	95.33 ±5.30	96.00 ±4.50
28	97.40 ±2.30	97.40 ±2.30	97.40 ±2.30	97.40 ±2.30	92.60 ±3.83
29	74.26 ±14.06	74.95 ±14.64	76.11 ±13.65	72.31 ±14.05	77.31 ±15.00
30	81.73 ±8.86	76.11 ±7.06	74.56 ±9.10	76.62 ±8.26	79.27 ±10.53
31	17.67 ±6.43	20.42 ±5.55	14.79 ±4.77	15.29 ±4.71	35.60 ±8.18
32	74.45 ±4.66	71.70 ±3.72	72.53 ±4.86	71.01 ±3.95	73.39 ±3.78
33	25.97 ±7.13	40.70 ±6.54	33.04 ±4.08	32.94 ±3.67	43.76 ±7.12
34	94.67 ±5.56	80.17 ±16.89	68.00 ±11.49	71.33 ±11.44	77.00 ±11.11
35	81.87 ±6.79	73.47 ±8.44	81.87 ±6.01	81.87 ±6.01	82.00 ±6.01
36	59.71 ±8.22	55.49 ±4.81	58.73 ±7.97	57.94 ±7.13	57.84 ±6.79
37	69.87 ±10.94	67.99 ±10.40	72.91 ±10.38	68.87 ±8.45	71.83 ±10.96
38	47.78 ±6.72	39.95 ±3.34	45.03 ±4.48	44.37 ±5.15	61.07 ±5.34
39	96.62 ±4.05	94.17 ±5.02	94.56 ±5.99	96.27 ±3.96	96.27 ±5.53
40	88.42 ±7.49	88.12 ±8.43	86.12 ±7.70	86.12 ±8.13	92.45 ±8.48
avg. rank	2.56	3.83	3.37	3.36	1.86

Table 7.1: Predictive accuracy results for different surrogate loss functions used during induction withthe PTBU algorithm. Results include mean and standard deviation.

Appendix B

All results of the dynamic operator exclusion experiment of Section 3.5.4.

No.	$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$
1	77.62 ±1.56	77.62 ±1.56	77.62 ±1.56	77.62 ±1.56	77.62 ±1.56
2	79.81 ±11.35	79.81 ±10.72	79.81 ±10.72	79.44 ±10.59	79.44 ±10.59
3	38.08 ±4.03	38.00 ±4.12	38.00 ±4.12	38.00 ±4.12	38.00 ±4.17
4	83.39 ±8.81	83.39 ±8.81	83.39 ±8.81	83.39 ±8.81	83.39 ±8.81
5	95.57 ±2.85	95.57 ±2.85	95.57 ±2.85	95.57 ±2.85	95.57 ±2.85
6	85.65 ±3.76	85.65 ±3.76	85.65 ±3.76	85.65 ±3.76	85.60 ±3.73
7	95.44 ±2.23	95.52 ±2.30	94.37 ±2.52	95.05 ±2.25	95.60 ±2.24
8	59.30 ±11.98	59.13 ±10.77	59.30 ±10.50	59.47 ±10.79	59.47 ±10.79
9	90.02 ±1.97	90.02 ±1.97	90.02 ±1.97	90.02 ±1.97	90.02 ±1.97
10	87.53 ±6.54	87.53 ±6.54	87.53 ±6.54	87.53 ±6.54	87.69 ±6.67
11	76.83 ±1.31	76.83 ±1.31	76.78 ±1.41	76.74 ±1.55	76.74 ±1.55
12	58.58 ±5.15	57.79 ±4.11	57.79 ±4.04	58.96 ±3.83	58.66 ±3.72
13	96.19 ±2.23	96.24 ±2.30	96.24 ±2.30	96.10 ±2.35	96.00 ±2.24
14	72.90 ±5.85	73.06 ±5.83	72.98 ±6.24	73.06 ±6.26	72.90 ±6.22
15	35.12 ±14.91	33.94 ±15.20	35.15 ±15.68	35.45 ±14.86	35.76 ±14.59
16	52.01 ±4.03	51.94 ±4.08	51.87 ±4.20	51.44 ±4.31	51.28 ±4.08
17	82.68 ±10.82	81.79 ±12.40	76.55 ±12.79	76.55 ±13.04	76.55 ±13.45
18	85.31 ±3.67	85.31 ±3.67	85.31 ±3.67	85.31 ±3.67	85.31 ±3.67
19	77.78 ±12.26	78.17 ±11.87	78.17 ±11.87	78.17 ±11.87	78.17 ±11.87
20	65.14 ±8.17	65.49 ±8.09	65.18 ±8.77	65.34 ±9.06	65.34 ±9.37
21	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68
22	72.37 ±3.85	72.37 ±3.85	72.37 ±3.85	72.37 ±3.85	72.30 ±3.81
23	60.29 ±7.39	59.97 ±7.10	60.13 ±7.28	59.51 ±8.97	59.67 ±9.06
24	73.42 ±2.18	73.74 ±2.64	73.85 ±2.59	74.29 ±3.05	74.84 ±2.96
25	78.89 ±5.93	80.00 ±6.50	80.62 ±5.64	80.62 ±5.64	80.99 ±5.56
26	87.76 ±5.54	87.76 ±5.54	87.76 ±5.54	87.76 ±5.43	87.76 ±5.43
27	92.22 ±6.80	93.11 ±6.43	93.11 ±6.43	93.11 ±6.43	93.11 ±6.43
28	93.33 ±3.54	93.33 ±3.54	92.53 ±3.89	92.53 ±3.89	92.53 ±3.89
29	77.31 ±13.79	77.31 ±13.79	77.31 ±13.79	77.31 ±13.79	77.31 ±13.79
30	77.90 ±9.48	78.37 ±10.04	78.14 ±9.92	77.92 ±9.48	77.92 ±10.26
31	31.82 ±7.50	32.62 ±7.21	32.72 ±7.38	33.51 ±7.83	33.80 ±7.73
32	73.31 ±4.90	72.66 ±4.66	72.66 ±4.66	72.66 ±4.66	73.13 ±3.81
33	44.35 ±6.71	44.64 ±6.42	44.64 ±6.60	44.64 ±6.60	44.44 ±6.73
34	77.17 ±11.12	77.17 ±11.12	77.17 ±11.12	77.17 ±11.12	77.17 ±11.12
35	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	58.24 ±6.53	58.14 ±6.45	58.24 ±6.15	58.04 ±6.51	58.33 ±6.74
37	71.86 ±10.59	71.86 ±10.59	71.70 ±10.85	71.86 ±10.96	71.84 ±10.86
38	56.85 ±4.29	57.05 ±4.62	57.76 ±4.58	57.92 ±4.02	58.24 ±4.01
39	95.12 ±4.13	94.95 ±4.00	95.33 ±4.19	95.52 ±3.70	95.70 ±3.50
40	90.79 ±7.77	90.45 ±8.43	90.45 ±8.43	90.45 ±8.43	90.45 ±8.43

Table 7.2: Part I: Mean accuracy measures with standard deviation comparing PTBU-DOE with different values of τ $(0.1 - 0.5)$.

No.	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$	$\tau = 1.0$
1	77.62 ±1.56	77.62 ±1.56	77.62 ±1.56	77.62 ±1.56	77.54 ±1.84
2	79.44 ±10.59	79.44 ±10.59	79.44 ±10.59	79.44 ±10.59	79.11 ±10.73
3	37.50 ±4.45	37.50 ±4.45	37.50 ±4.45	37.50 ±4.45	38.00 ±4.28
4	83.39 ±8.81	83.39 ±8.81	83.39 ±8.81	83.39 ±8.81	83.60 ±9.13
5	95.57 ±2.85	95.57 ±2.85	95.57 ±2.85	95.57 ±2.85	95.95 ±3.17
6	85.65 ±3.74	85.70 ±3.63	85.80 ±3.60	85.75 ±3.65	85.56 ±3.80
7	95.40 ±2.29	94.89 ±2.16	96.00 ±2.80	96.24 ±2.44	94.29 ±1.75
8	59.47 ±10.79	58.98 ±8.94	58.98 ±8.94	58.98 ±8.94	58.00 ±10.21
9	90.02 ±1.97	90.02 ±1.97	90.02 ±1.97	90.02 ±1.97	90.13 ±1.80
10	87.69 ±6.67	87.69 ±6.67	87.69 ±6.67	87.69 ±6.67	87.06 ±7.39
11	76.74 ±1.55	76.65 ±1.67	76.69 ±1.71	76.78 ±1.67	77.09 ±1.44
12	58.75 ±3.71	58.75 ±3.71	58.75 ±3.71	59.05 ±3.83	59.05 ±3.83
13	95.95 ±2.36	96.00 ±2.37	95.95 ±2.36	95.95 ±2.36	95.90 ±2.38
14	72.98 ±6.24	72.82 ±5.90	72.41 ±6.11	72.33 ±5.86	73.23 ±5.38
15	35.76 ±15.35	36.03 ±14.43	36.03 ±14.43	36.03 ±13.62	35.70 ±12.91
16	51.28 ±4.08	51.35 ±4.07	51.35 ±4.02	51.80 ±4.37	51.89 ±4.42
17	76.13 ±13.69	76.55 ±13.45	76.13 ±13.69	75.24 ±13.81	74.11 ±12.79
18	85.31 ±3.74	85.60 ±3.71	85.70 ±3.63	85.60 ±3.79	85.60 ±3.79
19	78.17 ±11.87	78.73 ±11.70	78.73 ±11.70	78.73 ±11.70	76.67 ±12.46
20	65.01 ±9.64	64.84 ±9.68	65.17 ±9.55	65.51 ±9.78	65.17 ±8.81
21	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68	82.75 ±0.68
22	72.30 ±3.81	72.30 ±3.81	72.30 ±3.81	72.17 ±3.63	72.17 ±3.63
23	59.98 ±9.34	59.82 ±9.09	60.29 ±8.75	61.08 ±7.72	61.39 ±7.74
24	75.28 ±4.27	75.50 ±4.60	75.61 ±4.38	75.61 ±4.46	75.71 ±4.49
25	81.11 ±5.62	80.99 ±5.56	80.99 ±5.56	80.99 ±5.56	80.37 ±5.51
26	87.76 ±4.99	87.67 ±4.95	87.76 ±4.78	87.85 ±5.00	87.48 ±5.28
27	93.11 ±6.43	94.22 ±5.46	94.22 ±5.46	94.22 ±5.46	95.78 ±4.46
28	92.53 ±3.89	92.53 ±3.89	92.53 ±3.89	92.07 ±3.62	91.67 ±3.33
29	77.31 ±13.79	77.31 ±13.79	77.31 ±13.79	77.31 ±13.79	77.31 ±15.00
30	77.92 ±10.26	77.92 ±10.26	77.92 ±10.26	77.92 ±10.26	77.95 ±10.98
31	33.91 ±8.16	33.51 ±7.93	33.61 ±7.66	33.61 ±7.66	34.91 ±8.77
32	73.13 ±3.81	73.13 ±3.81	73.09 ±3.78	73.09 ±3.78	72.83 ±4.15
33	44.44 ±6.64	44.15 ±6.94	44.05 ±7.13	44.15 ±7.11	43.76 ±6.87
34	77.17 ±11.12	77.17 ±11.12	77.17 ±11.12	77.17 ±11.12	77.00 ±11.11
35	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	58.33 ±6.74	58.33 ±6.74	58.24 ±6.93	58.43 ±6.59	58.43 ±6.81
37	71.84 ±10.86	71.84 ±10.86	72.16 ±10.84	72.17 ±10.74	72.17 ±10.74
38	58.27 ±4.28	58.15 ±4.58	59.37 ±4.35	60.47 ±4.81	61.10 ±5.20
39	95.14 ±5.03	95.14 ±5.03	95.14 ±5.03	94.97 ±4.92	96.46 ±4.50
40	90.45 ±8.43	90.45 ±8.43	90.79 ±7.77	90.79 ±7.77	92.76 ±7.75

Table 7.3: Part II: Mean accuracy measures with standard deviation comparing PTBU-DOE with different values of τ $(0.6 - 1.0)$.

No.	$\tau = 1.1$	$\tau = 1.2$	$\tau = 1.3$	$\tau = 1.4$	$\tau = 1.5$
1	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84
2	79.11 ±10.73	79.11 ±10.73	79.11 ±10.73	79.11 ±10.73	79.11 ±10.73
3	38.00 ±4.28	38.00 ±4.28	38.00 ±4.28	38.00 ±4.28	38.00 ±4.28
4	83.60 ±9.13	83.60 ±9.13	83.60 ±9.13	83.60 ±9.13	83.60 ±9.13
5	96.97 ±3.06	96.85 ±3.01	96.21 ±3.17	96.21 ±3.17	96.21 ±3.17
6	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73
7	94.33 ±2.04	94.29 ±2.13	94.17 ±1.98	94.13 ±2.07	94.21 ±2.29
8	57.84 ±10.34	58.79 ±10.27	58.79 ±10.27	58.47 ±10.34	58.47 ±10.34
9	90.18 ±1.78	90.02 ±2.09	90.02 ±2.09	90.02 ±2.09	90.02 ±2.09
10	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67
11	77.01 ±1.58	77.01 ±1.58	77.01 ±1.58	77.01 ±1.58	77.01 ±1.58
12	59.05 ±3.83	58.95 ±3.79	58.95 ±4.02	59.24 ±4.19	59.24 ±4.19
13	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38
14	73.14 ±5.33	73.06 ±5.34	72.90 ±5.18	72.74 ±5.17	72.57 ±4.93
15	35.36 ±13.19	35.36 ±13.19	35.36 ±13.19	35.36 ±13.19	35.36 ±13.19
16	52.30 ±3.64	52.28 ±3.65	52.28 ±3.65	52.28 ±3.65	52.28 ±3.65
17	74.58 ±12.95	74.58 ±12.95	74.58 ±12.95	74.58 ±12.95	74.17 ±12.72
18	85.51 ±3.81	85.51 ±3.81	85.51 ±3.81	85.51 ±3.81	85.51 ±3.81
19	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74
20	66.70 ±8.01	66.87 ±7.92	66.53 ±8.25	66.69 ±8.27	67.03 ±8.19
21	82.85 ±0.60	82.85 ±0.60	82.85 ±0.60	82.85 ±0.60	82.85 ±0.60
22	72.17 ±3.63	72.17 ±3.63	72.17 ±3.63	72.17 ±3.63	72.17 ±3.63
23	61.85 ±7.22	61.70 ±7.27	61.70 ±7.27	61.54 ±7.10	61.54 ±7.10
24	75.49 ±5.11	75.49 ±5.11	75.49 ±5.11	75.49 ±5.11	75.49 ±5.11
25	80.49 ±5.67	80.49 ±5.67	80.49 ±5.67	80.49 ±5.67	80.37 ±5.77
26	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28
27	96.00 ±4.50	96.00 ±4.50	96.00 ±4.50	95.78 ±5.10	95.78 ±5.10
28	91.67 ±3.33	92.67 ±3.84	92.60 ±3.83	92.60 ±3.83	92.60 ±3.83
29	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00
30	79.27 ±10.73	79.25 ±10.30	79.25 ±10.30	79.25 ±10.30	79.03 ±10.79
31	35.00 ±8.46	34.90 ±8.79	35.00 ±8.61	35.11 ±8.58	35.10 ±8.67
32	72.83 ±4.15	72.83 ±4.15	72.96 ±4.23	73.00 ±4.21	73.00 ±4.21
33	43.76 ±6.74	43.56 ±6.98	43.56 ±6.98	43.46 ±6.95	43.56 ±6.98
34	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11
35	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	58.33 ±6.91	58.14 ±6.63	58.14 ±6.63	58.14 ±6.63	58.14 ±6.63
37	72.17 ±10.74	71.83 ±10.96	71.83 ±10.96	71.83 ±10.96	71.83 ±10.96
38	61.42 ±5.36	61.19 ±5.24	61.30 ±5.34	61.26 ±5.37	61.22 ±5.39
39	96.83 ±4.31	96.83 ±4.31	96.83 ±4.31	96.64 ±4.29	96.46 ±4.95
40	93.42 ±7.46	93.42 ±7.46	93.42 ±7.46	93.09 ±8.27	93.09 ±8.27

Table 7.4: Part III: Mean accuracy measures with standard deviation comparing PTBU-DOE with different values of τ $(1.1 - 1.5)$.

No.	$\tau = 1.6$	$\tau = 1.7$	$\tau = 1.8$	$\tau = 1.9$	$\tau = 2.0$
1	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84	77.54 ±1.84
2	79.11 ±10.73	78.74 ±10.16	78.74 ±10.16	78.74 ±10.16	78.74 ±10.16
3	38.00 ±4.28	38.00 ±4.28	38.08 ±4.29	38.00 ±4.12	37.00 ±4.47
4	83.60 ±9.13	83.81 ±8.85	83.81 ±8.85	83.81 ±8.85	83.81 ±8.85
5	96.33 ±3.25	96.84 ±3.19	96.84 ±3.19	96.84 ±3.19	96.71 ±3.13
6	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73
7	93.98 ±2.65	93.90 ±2.35	93.74 ±2.35	93.74 ±2.35	93.78 ±2.35
8	58.48 ±10.61	58.48 ±10.61	58.48 ±10.61	58.48 ±10.61	58.48 ±10.61
9	90.02 ±2.09	90.02 ±2.09	90.02 ±2.09	90.02 ±2.09	90.02 ±2.09
10	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67	87.21 ±7.67
11	77.01 ±1.58	77.01 ±1.58	77.01 ±1.58	77.01 ±1.58	77.09 ±1.67
12	59.52 ±4.33	59.24 ±4.19	59.04 ±4.54	59.24 ±4.19	59.04 ±4.31
13	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38	95.90 ±2.38
14	72.57 ±4.93	72.65 ±4.91	72.65 ±4.91	72.65 ±4.91	72.73 ±5.12
15	35.36 ±13.19	35.36 ±13.19	35.06 ±12.83	34.79 ±13.96	35.39 ±13.61
16	52.28 ±3.73	52.46 ±3.66	52.19 ±3.83	52.19 ±3.72	52.23 ±3.72
17	74.64 ±13.42	74.64 ±13.42	76.07 ±11.68	74.64 ±11.88	75.18 ±12.26
18	85.51 ±3.81	85.51 ±3.81	85.51 ±3.81	85.51 ±3.81	85.36 ±3.70
19	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74	76.11 ±11.74
20	67.03 ±8.19	67.20 ±8.39	67.04 ±8.38	67.04 ±8.38	66.68 ±8.65
21	82.85 ±0.60	82.85 ±0.60	82.85 ±0.60	82.85 ±0.60	82.75 ±0.68
22	72.20 ±3.65	72.20 ±3.65	72.20 ±3.65	72.23 ±3.68	72.27 ±3.71
23	61.23 ±7.58	61.23 ±7.58	61.23 ±7.58	61.54 ±7.49	61.38 ±7.53
24	75.49 ±5.11	75.81 ±5.18	75.81 ±5.18	75.81 ±5.18	75.92 ±5.09
25	80.37 ±5.77	80.37 ±5.77	80.37 ±5.77	80.37 ±5.77	80.25 ±5.86
26	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28	87.48 ±5.28
27	95.78 ±5.10	95.56 ±5.05	95.56 ±5.05	95.56 ±5.05	96.00 ±4.50
28	92.60 ±3.83	92.60 ±3.83	92.60 ±3.83	92.60 ±3.83	92.60 ±3.83
29	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00
30	79.03 ±10.79	79.03 ±10.79	79.03 ±10.79	79.03 ±10.79	79.27 ±10.53
31	34.91 ±8.66	35.01 ±8.72	35.01 ±8.58	35.01 ±8.58	35.60 ±8.18
32	73.05 ±4.17	73.09 ±4.15	73.09 ±4.15	73.09 ±4.15	73.39 ±3.78
33	43.46 ±6.95	43.46 ±6.82	43.36 ±6.84	43.56 ±6.90	43.76 ±7.12
34	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11
35	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	57.94 ±6.52	58.04 ±6.64	57.84 ±6.79	57.84 ±6.79	57.84 ±6.79
37	71.83 ±10.96	71.83 ±10.96	71.83 ±10.96	71.83 ±10.96	71.83 ±10.96
38	61.11 ±5.42	61.22 ±5.39	61.26 ±5.37	61.26 ±5.37	61.07 ±5.34
39	96.09 ±5.50	96.09 ±5.50	96.27 ±5.53	96.27 ±5.53	96.27 ±5.53
40	93.09 ±8.27	92.76 ±8.59	92.45 ±8.48	92.45 ±8.48	92.45 ±8.48

Table 7.5: Part IV: Mean accuracy measures with standard deviation comparing PTBU-DOE with different values of τ $(1.6 - 2.0)$.

Appendix C

All results of the limited candidate history experiment of Section 3.5.4.

No.	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
1	77.29 ±1.56	77.46 ±1.63	77.62 ±1.80	77.54 ±1.60	77.62 ±1.56
2	79.78 ±10.36	79.41 ±10.22	80.44 ±10.35	78.74 ±10.16	78.74 ±10.16
3	22.92 ±5.50	31.25 ±4.68	33.83 ±4.81	33.33 ±5.62	33.67 ±5.56
4	84.22 ±9.05	84.42 ±9.20	83.60 ±9.13	83.60 ±9.13	83.20 ±9.49
5	95.57 ±2.85	95.70 ±2.79	97.09 ±2.60	97.34 ±2.69	97.72 ±2.95
6	85.56 ±3.70	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73	85.51 ±3.73
7	94.81 ±2.15	94.69 ±2.04	94.97 ±1.83	95.48 ±2.03	95.84 ±1.81
8	58.29 ±9.35	58.94 ±9.74	59.12 ±9.24	59.13 ±9.31	59.13 ±9.23
9	78.76 ±6.54	86.55 ±4.33	90.29 ±1.83	90.24 ±1.81	90.29 ±1.83
10	82.62 ±7.14	86.59 ±7.60	86.91 ±7.73	86.57 ±8.16	86.90 ±8.06
11	76.74 ±1.07	76.78 ±1.20	76.83 ±1.45	77.05 ±1.69	77.14 ±1.56
12	57.49 ±3.06	57.89 ±3.92	58.76 ±4.23	59.25 ±3.97	60.22 ±4.15
13	96.19 ±2.30	96.19 ±2.23	96.10 ±2.22	96.00 ±2.34	96.05 ±2.32
14	71.84 ±5.05	71.67 ±4.78	72.16 ±4.73	72.33 ±5.10	72.41 ±5.60
15	33.36 ±15.38	35.42 ±12.95	37.58 ±12.11	37.91 ±12.94	38.88 ±13.60
16	49.68 ±3.56	53.25 ±3.84	53.09 ±3.72	53.34 ±3.69	53.05 ±3.71
17	73.27 ±12.44	80.89 ±13.06	82.68 ±11.91	85.06 ±11.40	85.83 ±9.49
18	85.70 ±3.63	85.27 ±3.67	85.22 ±3.64	85.22 ±3.64	85.31 ±3.68
19	76.67 ±11.67	75.56 ±12.36	76.11 ±12.32	75.63 ±12.22	76.59 ±11.22
20	63.39 ±8.30	67.36 ±9.20	65.64 ±8.99	67.21 ±9.33	67.74 ±9.16
21	82.69 ±0.73	82.72 ±0.68	82.75 ±0.73	82.79 ±0.72	82.72 ±0.74
22	72.27 ±3.70	72.67 ±3.52	72.77 ±3.35	72.53 ±3.63	72.60 ±3.84
23	57.62 ±8.02	60.44 ±8.58	61.81 ±8.45	62.30 ±8.46	62.31 ±8.05
24	73.09 ±2.66	75.26 ±3.55	74.72 ±4.17	75.50 ±4.17	74.96 ±4.38
25	75.93 ±6.51	78.40 ±6.74	79.26 ±6.71	80.12 ±6.70	80.37 ±5.93
26	84.05 ±6.16	86.63 ±6.08	88.15 ±5.89	88.34 ±5.72	88.99 ±5.62
27	95.56 ±5.05	95.78 ±5.10	95.56 ±5.05	95.56 ±5.05	95.56 ±5.05
28	74.93 ±6.45	85.47 ±5.01	92.80 ±4.19	95.00 ±3.43	95.20 ±3.18
29	76.94 ±15.12	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00
30	75.90 ±10.11	79.27 ±10.23	79.25 ±10.81	79.70 ±8.93	80.17 ±9.05
31	28.94 ±7.51	35.58 ±7.50	35.98 ±8.06	35.59 ±7.95	36.39 ±7.75
32	71.70 ±4.36	72.62 ±4.74	72.96 ±4.35	73.14 ±4.13	72.96 ±4.17
33	40.71 ±6.18	44.55 ±7.25	45.34 ±7.26	45.15 ±7.43	44.56 ±7.46
34	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11
35	81.87 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	57.75 ±5.80	57.75 ±6.48	57.94 ±5.89	58.14 ±6.21	58.33 ±6.78
37	72.01 ±10.64	70.23 ±9.60	71.52 ±9.72	71.85 ±10.47	71.85 ±10.17
38	52.80 ±5.32	56.86 ±5.50	57.96 ±4.83	61.98 ±6.08	62.52 ±5.66
39	93.80 ±5.40	95.33 ±5.31	95.33 ±4.41	95.52 ±4.24	95.52 ±4.24
40	86.76 ±9.20	90.79 ±8.61	92.76 ±8.25	92.79 ±8.17	92.76 ±8.59

Table 7.6: Part I: Mean accuracy measures with standard deviation comparing PTBU-LCH with different values of k $(1 - 5)$.

No.	$k = 6$	$k = 7$	$k = 8$	$k = 9$
1	77.62 ±1.56	77.78 ±1.49	77.78 ±1.46	77.70 ±1.64
2	78.44 ±11.87	78.37 ±12.35	77.67 ±11.70	78.07 ±11.03
3	34.50 ±5.27	33.75 ±5.64	34.25 ±5.77	33.67 ±4.58
4	83.41 ±9.22	83.20 ±9.49	83.20 ±9.49	83.21 ±9.44
5	97.47 ±2.90	97.73 ±2.94	97.60 ±2.92	97.60 ±2.93
6	85.56 ±3.72	85.51 ±3.73	85.65 ±3.77	85.65 ±3.77
7	96.16 ±1.83	96.35 ±2.00	96.31 ±1.83	96.39 ±1.81
8	59.11 ±8.81	60.41 ±9.46	61.04 ±9.21	62.17 ±10.21
9	90.29 ±1.83	90.29 ±1.83	90.29 ±1.83	90.29 ±1.83
10	87.05 ±7.94	86.57 ±8.00	87.37 ±8.01	87.21 ±8.19
11	77.09 ±1.35	77.14 ±1.28	77.09 ±1.30	77.14 ±1.28
12	59.64 ±4.49	58.86 ±4.06	59.44 ±4.52	59.93 ±4.59
13	96.05 ±2.32	96.00 ±2.40	96.05 ±2.31	96.10 ±2.32
14	72.49 ±5.29	72.65 ±5.12	72.74 ±4.97	72.74 ±4.77
15	38.55 ±12.74	38.82 ±13.97	38.21 ±13.44	37.91 ±13.37
16	53.34 ±3.75	53.34 ±3.23	53.00 ±3.36	52.96 ±3.43
17	86.31 ±9.84	86.31 ±9.84	85.89 ±10.57	85.42 ±10.23
18	85.12 ±3.82	85.17 ±3.74	85.07 ±3.73	85.17 ±3.80
19	76.19 ±12.94	75.71 ±12.84	76.67 ±13.02	74.60 ±12.08
20	65.52 ±10.19	65.85 ±10.08	65.01 ±10.80	65.00 ±11.18
21	82.66 ±0.86	82.72 ±0.81	82.69 ±0.78	82.72 ±0.81
22	72.40 ±3.64	72.37 ±3.60	72.10 ±3.67	72.47 ±3.72
23	61.05 ±7.78	61.49 ±7.01	62.11 ±6.60	61.49 ±6.77
24	74.74 ±4.29	74.74 ±4.29	74.63 ±4.19	74.63 ±4.19
25	81.60 ±5.46	82.10 ±5.15	81.85 ±5.53	81.60 ±5.54
26	89.09 ±5.69	88.71 ±5.84	88.90 ±5.84	88.81 ±5.82
27	95.56 ±5.05	95.56 ±5.05	95.56 ±5.05	95.56 ±5.05
28	95.93 ±2.95	97.40 ±2.30	97.40 ±2.30	97.40 ±2.30
29	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00	77.31 ±15.00
30	80.38 ±9.34	79.70 ±9.50	79.70 ±9.00	80.37 ±8.56
31	36.79 ±8.01	36.30 ±8.10	35.90 ±7.63	36.00 ±7.60
32	73.09 ±3.95	73.05 ±4.06	73.05 ±4.06	73.05 ±4.06
33	45.04 ±7.97	44.95 ±7.71	44.26 ±8.01	44.95 ±7.03
34	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11	77.00 ±11.11
35	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01	82.00 ±6.01
36	58.53 ±6.66	58.63 ±7.08	58.73 ±7.01	58.63 ±7.28
37	71.52 ±9.94	71.37 ±10.09	71.37 ±10.09	71.71 ±9.77
38	62.64 ±5.52	62.96 ±5.71	63.31 ±5.60	63.67 ±5.63
39	95.52 ±4.24	95.34 ±4.86	95.34 ±4.86	95.34 ±4.86
40	92.76 ±7.82	92.79 ±8.17	93.42 ±7.07	93.09 ±7.46

Table 7.7: Part II: Mean accuracy measures with standard deviation comparing PTBU-LCH with different values of k $(6 - 9)$.

Appendix D

All results of the Pareto-comparison experiment of Section 4.2.

No.	PTBU	PTTD	PTCoEvo
1	77.70 ± 1.51	76.97 ± 1.35	
2	79.81 ± 11.35	82.11 ± 8.05	78.67 ± 10.72
3	39.50 ± 5.85	38.17 ± 4.50	
4	83.59 ± 8.72	90.80 ± 10.41	94.82 ± 8.03
5	95.95 ± 3.15	95.57 ± 2.68	97.47 ± 2.72
6	85.65 ± 3.76	85.51 ± 3.73	85.60 ± 3.97
7	95.64 ± 1.88	98.61 ± 0.99	
8	62.20 ± 10.76	62.45 ± 9.44	
9	90.34 ± 1.50	89.17 ± 2.20	92.21 ± 2.11
10	87.53 ± 6.54	87.87 ± 6.99	87.86 ± 6.57
11	76.96 ± 1.33	77.58 ± 2.46	77.76 ± 3.83
12	59.83 ± 5.45	66.79 ± 8.13	67.95 ± 8.65
13	96.24 ± 2.23	96.54 ± 2.38	96.73 ± 2.02
14	73.57 ± 5.43	72.74 ± 4.85	74.01 ± 6.89
15	39.18 ± 13.13	34.94 ± 14.19	
16	54.04 ± 3.40	53.27 ± 3.30	
17	84.35 ± 8.58	81.07 ± 13.71	
18	85.36 ± 3.68	85.65 ± 3.60	85.41 ± 3.69
19	78.73 ± 12.28	79.13 ± 10.61	
20	68.09 ± 9.44	69.46 ± 8.82	
21	82.85 ± 0.60	82.79 ± 0.60	
22	72.70 ± 3.22	73.87 ± 3.07	72.83 ± 3.60
23	60.76 ± 7.08	63.45 ± 7.49	
24	73.95 ± 2.73	73.73 ± 2.87	74.73 ± 5.07
25	79.51 ± 5.48	80.99 ± 7.95	80.00 ± 7.25
26	88.71 ± 5.25	89.56 ± 5.86	89.08 ± 4.87
27	95.11 ± 4.93	96.67 ± 4.55	96.00 ± 4.50
28	96.07 ± 2.49	97.47 ± 2.29	98.13 ± 1.89
29	77.31 ± 13.79	76.16 ± 15.38	74.63 ± 13.13
30	77.90 ± 9.48	82.90 ± 9.58	
31	33.12 ± 7.29	32.53 ± 7.77	
32	73.57 ± 5.15	76.74 ± 4.65	76.31 ± 4.30
33	44.95 ± 6.63	44.27 ± 6.22	
34	77.17 ± 11.12	77.33 ± 10.81	97.83 ± 5.20
35	82.00 ± 6.01	82.13 ± 5.92	84.80 ± 7.23
36	59.02 ± 7.32	59.51 ± 8.83	
37	71.87 ± 9.83	75.65 ± 9.40	
38	58.31 ± 5.11	63.04 ± 4.03	
39	95.50 ± 4.02	98.12 ± 3.08	96.07 ± 3.64
40	92.79 ± 8.58	94.70 ± 7.28	

Table 7.8: Accuracy results of the three main variants selected for comparison in Section 4.2.

No.	PTBU	PTTD	PTCoEvo
1	5.48 ± 1.27	0.78 ± 0.17	
2	0.90 ± 0.13	0.26 ± 0.04	883.68 ± 1067.20
3	3.50 ± 0.71	0.25 ± 0.05	
4	1.45 ± 0.55	2.50 ± 0.43	68.65 ± 36.04
5	0.25 ± 0.08	0.07 ± 0.03	670.63 ± 379.61
6	8.52 ± 3.47	0.58 ± 0.08	2827.07 ± 3327.33
7	73.50 ± 11.37	265.88 ± 21.99	
8	10.82 ± 1.79	39.24 ± 4.01	
9	10.67 ± 3.15	0.50 ± 0.11	3369.25 ± 1637.23
10	0.21 ± 0.06	0.53 ± 0.11	178.94 ± 83.88
11	0.42 ± 0.12	0.56 ± 0.09	5272.46 ± 5152.55
12	0.41 ± 0.10	0.49 ± 0.07	60.27 ± 25.41
13	1.01 ± 0.18	2.01 ± 0.35	5907.20 ± 4172.61
14	2.95 ± 0.39	1.49 ± 0.24	4506.19 ± 2484.45
15	2.72 ± 0.82	0.08 ± 0.03	
16	1.52 ± 0.20	11.02 ± 1.61	
17	0.38 ± 0.14	0.21 ± 0.07	
18	1.31 ± 0.16	0.68 ± 0.10	3004.48 ± 3690.30
19	2.89 ± 0.20	2.90 ± 0.40	
20	4.58 ± 0.45	46.08 ± 5.07	
21	6.45 ± 0.77	38.72 ± 5.57	
22	2.89 ± 0.70	22.57 ± 3.55	790.88 ± 381.06
23	2.64 ± 0.51	2.45 ± 0.22	
24	0.06 ± 0.05	0.06 ± 0.03	1087.07 ± 708.59
25	0.63 ± 0.08	0.36 ± 0.05	40.57 ± 16.06
26	4.28 ± 0.50	5.61 ± 1.27	1185.08 ± 1142.90
27	0.37 ± 0.21	0.08 ± 0.03	3343.05 ± 1119.27
28	0.82 ± 0.08	1.25 ± 0.08	138.83 ± 46.72
29	0.17 ± 0.07	0.09 ± 0.06	16.27 ± 7.67
30	1.19 ± 0.24	3.34 ± 0.49	
31	5.08 ± 0.45	1.50 ± 0.22	
32	0.35 ± 0.07	1.35 ± 0.12	5032.81 ± 4003.85
33	34.24 ± 21.64	25.25 ± 2.19	
34	0.08 ± 0.06	0.05 ± 0.04	317.07 ± 279.83
35	0.08 ± 0.20	0.03 ± 0.04	288.81 ± 232.27
36	0.23 ± 0.07	1.91 ± 0.13	
37	4.00 ± 0.55	12.88 ± 1.21	
38	12.55 ± 3.27	17.89 ± 1.98	
39	0.25 ± 0.08	2.48 ± 0.73	4432.00 ± 2590.56
40	3.28 ± 0.89	2.29 ± 0.30	

Table 7.9: Training runtime results in seconds (s) of the three main variants selected for comparison in Section 4.2.

References

[1] Aha, D. W. & Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, *6*, 37–66. 92

[2] Alcalá-Fdez, J., Sánchez, L., García, S., del Jesús, M. J., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., & Rivas, V. M. (2009). Keel: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, *13*(3), 307–318. 93, 96

[3] Allwein, E. L., Schapire, R. E., & Singer, Y. (2001). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, *1*, 113–141. 35

[4] Alpaydin, E. (2004). *Introduction to Machine Learning*. Massachusetts Institute of Technology (MIT) Press. 4

[5] Altendorf, E. E., Restificar, A. C., & Dietterich, T. G. (2012). Learning from sparse data by exploiting monotonicity constraints. *ArXiv preprint arXiv:1207.1364*. 20

[6] Angluin, D. (1988). Queries and concept learning. *Machine Learning*, *2*(4), 319–342. 11

[7] Asuncion, A. & Newman, D. (2009). UCI machine learning repository. Accessed 13 Nov 2009. 30

[8] Audibert, J.-Y., Munos, R., & Szepesvári, C. (2007). Tuning bandit algorithms in stochastic environments. In *Algorithmic Learning Theory*, (pp. 150–165). 70

[9] Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. A Rand Corporation Research Study Series. Princeton University Press. 20

[10] Bellman, R., Kalaba, R., & Zadeh, L. (1966). Abstraction and pattern classification. *Journal of Mathematical Analysis and Applications*, *13*(1), 1–7. 8

[11] Bellman, R. E. & Zadeh, L. A. (1970). Decision-making in a fuzzy environment. *Management Science*, *17*(4), B–141. 8

[12] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. 103

[13] Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific. 18

[14] Beyer, H.-G. & Schwefel, H.-P. (2002). Evolution strategies - a comprehensive introduction. *Natural Computing*, *1*, 3–52. 18, 108

[15] Bishop, C. M. & Nasrabadi, N. M. (2006). *Pattern Recognition and Machine Learning*, volume 1. Springer New York. 4

[16] Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, *24*(6), 377 – 380. 28, 60

[17] Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the vapnik-chervonenkis dimension. *Journal of the Association of Computing Machinery (ACM)*, *36*(4), 929–965. 26

[18] Bro, R. & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics*, *11*(5), 393–401. 18

[19] Bösner, S., Haasenritter, J., Becker, A., Karatolios, K., Vaucher, P., Gencer, B., Herzig, L., Heinzel-Gutenbrunner, M., Schaefer, J. R., Hani, M. A., Keller, H., Sönnichsen, A. C., Baum, E., & Donner-Banzhoff, N. (2010). Ruling out coronary artery disease in primary care: development and validation of a simple prediction rule. *Canadian Medical Association Journal*, *182*(12), 1295–1300. 19

[20] Chang, P.-C. & Liu, C.-H. (2008). A tsk type fuzzy rule based system for stock price prediction. *Expert Systems with Applications*, *34*(1), 135 – 144. 101

[21] Chen, G., Wei, Q., Kerre, E., & Wets, G. (2003). Overview of fuzzy associations mining. In *Proceedings of 2003 ISIS*. 10

[22] Cheong, F. & Lai, R. (2007). Designing a hierarchical fuzzy logic controller using the differential evolution approach. *Applied Soft Computing*, *7*(2), 481–491. 104

[23] Church, A. (1932). A set of postulates for the foundation of logic. *Annals of Mathematics*, *33*(2), 346–366. 1

[24] Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of 2009 International Conference on Machine Learning*, (pp. 115–123)., Tahoe City, CA. 93, 101

[25] Cordón, O., del Jesus, M. J., & Herrera, F. (1999). A proposal on reasoning methods in fuzzy rule-based classification systems. *International Journal of Approximate Reasoning*, *20*(1), 21–45. 101

[26] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* (2 ed.). The Massachusetts Institute of Technology (MIT) Press. 1

[27] De'ath, G. & Fabricius, K. E. (2000). Classification and regression trees: A powerful yet simple technique for ecological data analysis. *Ecology*, *81*(11), 3178–3192. 105

[28] del Jesus, M. J., Hoffmann, F., Navascués, L. J., & Sánchez, L. (2004). Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. *IEEE Transactions on Fuzzy Systems*, *12*(3), 296–308. 101

[29] Delgado, M., Marín, N., Sánchez, D., & Vila, M.-A. (2003). Fuzzy association rules: General model and applications. *IEEE Transactions on Fuzzy Systems*, *11*(2), 214–225. 10

[30] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30. 92, 93

[31] Dubois, D. & Prade, H. (1997). The three semantics of fuzzy sets. *Fuzzy Sets and Systems*, *90*(2), 141–150. 8

[32] Duivesteijn, W. & Feelders, A. (2008). Nearest neighbour classification with monotonicity constraints. In *Machine Learning and Knowledge Discovery in Databases* (pp. 301–316). Springer. 19

[33] Fine, T. L. (1973). *Theories of Probability*. Academic Press. 8

[34] Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press. 2, 4, 92

[35] Fodor, J. & Yager, R. R. (2000). Fuzzy set-theoretic operators and quantifiers. In *Fundamentals of Fuzzy Sets* (pp. 125–193). Springer. 17

[36] Fürnkranz, J. & Hüllermeier, E. (2010). *Preference Learning*. Springer. 112

[37] García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. *Information Sciences*, *180*, 2044 – 2064. 93

[38] Gath, I. & Geva, A. B. (1989). Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(7), 773–780. 101

[39] González, A. & Pérez, R. (1999). Slave: A genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, *7*, 176–191. 93, 101

[40] González, A. & Pérez, R. (2001). Selection of relevant features in a fuzzy genetic learning algorithm. *IEEE Transactions on Systems, Man and and Cybernetics*, *31*, 417–425. 93

[41] Grabisch, M., Marichal, J.-L., Mesiar, R., & Pap, E. (2009). *Aggregation Functions* (1st ed.). New York, NY, USA: Cambridge University Press. 16, 17

[42] Greco, S., Mousseau, V., & Słowiński, R. (2008). Ordinal regression revisited: Multiple criteria ranking using a set of additive value functions. *European Journal of Operational Research*, *191*(2), 416–436. 113

[43] Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*. PWS Publishing Company, Boston. 108

[44] Hagras, H. A. (2004). A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. *IEEE Transactions on Fuzzy Systems*, *12*(4), 524–539. 103

[45] Hamacher, H. (1975). *Über logische Verknüpfungen unscharfer Aussagen und deren zugehörige Bewertungsfunktionen*, volume 14 of *Arbeitsbericht*. 17

[46] Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The Elements of Statistical Learning*, volume 1. Springer New York. 2

[47] Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, *6*, 65–70. 93

[48] Höppner, F., Klawonn, F., Kruse, R., & Runkler, T. (1999). *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. John Wiley. 10

[49] Huang, Z., Gedeon, T., & Nikravesh, M. (2008). Pattern trees induction: A new machine learning method. *IEEE Transactions on Fuzzy Systems*, *16*(4), 958 –970. 13, 15, 29, 39, 43, 46, 59, 63, 67, 92, 107

[50] Hühn, J. & Hüllermeier, E. (2009). Furia: An algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, *19*(3), 293–319. 101

[51] Hüllermeier, E. (2011). Fuzzy sets in machine learning and data mining. *Applications of Soft Computing*, *11*(2), 1493–1505. 10

[52] Ishibuchi, H. & Nakashima, T. (2001). Effect of rule weights in fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, *9*(4), 506–515. 102

[53] Jacquet-Lagreze, E. & Siskos, J. (1982). Assessing a set of additive utility functions for multicriteria decision-making, the UTA method. *European Journal of Operational Research*, *10*(2), 151–164. 113

[54] Janikow, C. Z. (1998). Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics*, *28*(1), 1–14. 10, 105, 107

[55] Janssen, F. & Fürnkranz, J. (2009). A re-evaluation of the over-searching phenomenon in inductive rule learning. In *Proceedings of 2009 SIAM International Conference on Data Mining*, (pp. 329–340). 91

[56] Joo, M. G. & Lee, J. S. (1999). Hierarchical fuzzy control scheme using structured takagi-sugeno type fuzzy inference. In *Proceedings of 1999 IEEE International Fuzzy Systems Conference*, volume 1, (pp. 78–83). 103, 104

[57] Kearns, M. J. & Vazirani, U. (1994). An introduction to computational learning theory. 28

[58] Keshwani, D. R., Jones, D. D., Meyer, G. E., & Brand, R. M. (2008). Rule-based mamdani-type fuzzy modeling of skin permeability. *Applied Soft Computing*, *8*(1), 285–294. 101

[59] Klement, E. P., Mesiar, R., & Pap, E. (2002). *Triangular Norms*. Kluwer Academic Publishers. 6

[60] Linkens, D. A., Shieh, J. S., & Peacock, J. E. (1996). Hierarchical fuzzy modelling for monitoring depth of anaesthesia. *Fuzzy Sets and Systems*, *79*(1), 43–57. 103

[61] Ludbrook, J. (1998). Multiple comparison procedures updated. *Clinical and Experimental Pharmacology and Physiology*, *25*(12), 1032–1037. 70

[62] Maeda, H. (1996). An investigation on the spread of fuzziness in multi-fold multi-stage approximate reasoning by pictorial representation—under sup-min composition and triangular type membership function. *Fuzzy Sets and Systems*, *80*(2), 133–148. 104

[63] Mamdani, E. H. (1974). Application of fuzzy algorithms for control of simple dynamic plant. *Proceedings of the Institution of Electrical Engineers*, *121*(12), 1585–1588. 100, 101

[64] Maron, O. & Moore, A. W. (1993). Hoeffding races: Accelerating model selection search for classification and function approximation. *Robotics Institute*, 263. 68

[65] Maron, O. & Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, *11*(1-5), 193–225. 70

[66] Meyer, M. & Vlachos, P. (2009). Statlib data, software and news from the statistics community. Accessed 13 Nov 2009. 30

[67] Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill. 1

[68] Mnih, V., Szepesvári, C., & Audibert, J.-Y. (2008). Empirical bernstein stopping. In *Proceedings of 2008 International Conference on Machine Learning*, (pp. 672–679). 70

[69] Murthy, S. K. & Salzberg, S. (1995). Lookahead and pathology in decision tree induction. In *Proceedings of International Joint Conferences on Artificial Intelligence*, (pp. 1025–1033). 91

[70] Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press. 42

[71] Nasiri, M., Fober, T., Senge, R., & Hüllermeier, E. (2013). Fuzzy pattern trees as an alternative to rule-based fuzzy systems: Knowledge-driven, data-driven and hybrid modeling of color yield in polyester dyeing. In *Proceedings of 2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, (pp. 715–721). 13

[72] Nasiri, M., Hüllermeier, E., Senge, R., & Lughofer, E. (2011). Comparing methods for knowledge-driven and data-driven fuzzy modeling: A case study in textile industry. In *Proceedings of 2011 World Congress of the International Fuzzy Systems Association*, (pp. RW–103–1–6). 13

[73] Neal, R. M. & Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models* (pp. 355–368). Springer. 108

[74] Olaru, C. & Wehenkel, L. (2003). A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, *138*(2), 221–254. 105, 107

[75] Passino, K. M. & Yurkovich, S. (1998). *Fuzzy Control*, volume 42. Citeseer. 100

[76] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann. 107

[77] Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Learning* (pp. 185–208). Cambridge, MA, USA: Massachusetts Institute of Technology (MIT) Press. 92

[78] Poon, H. & Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Proceedings of 2011 IEEE International Conference on Computer Vision*, (pp. 689–690). 107, 108

[79] Potharst, R. & Feelders, A. J. (2002). Classification trees for problems with monotonicity constraints. *ACM SIGKDD Explorations Newsletter*, *4*(1), 1–10. 19

[80] Potter, M. A. & De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature—PPSN III* (pp. 249–257). Springer. 80

[81] Quade, D. (1979). Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, *74*(367), 680–683. 93

[82] Quinlan, J. & Cameron-Jones, R. (1995). Oversearching and layered search in empirical learning. *Breast Cancer*, *286*, 2–7. 91

[83] Quinlan, R. J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. 105

[84] Quinlan, R. J. (1987). Decision trees at probabilistic classifiers. *Proceedings of 1987 International Workshop on Machine Learning*, 31–37. 105

[85] Quinlan, R. J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers. 5, 92

[86] Reed, M. & Simon, B. (1981). *I: Functional Analysis*, volume 1. Access Online via Elsevier. 23

[87] Rubin, D. B. (1976). Inference and missing data. *Biometrika*, *63*(3), 581–592. 20

[88] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, DTIC Document. 108

[89] Schafer, J. L. & Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods*, *7*(2), 147–177. 20, 21

[90] Schweizer, B. & Sklar, A. (1983). *Probabilistic Metric Spaces*. New York. 15

[91] Senge, R., Fober, T., Nasiri, M., & Hüllermeier, E. (2012). Fuzzy Pattern Trees: Ein alternativer Ansatz zur Fuzzy-Modellierung. *at-Automatisierungstechnik*, *60*(10), 622–629. 13

[92] Senge, R. & Hüllermeier, E. (2009). Learning pattern tree classifiers using a co-evolutionary algorithm. In *Proceedings of Knowledge Discovery, Data Mining und Maschinelles Lernen*, volume 19, (pp. 105–110). 79

[93] Senge, R. & Hüllermeier, E. (2011). Top-down induction of fuzzy pattern trees. *IEEE Transactions on Fuzzy Systems*, *19*(2), 241 –252. 43, 56, 85, 92

[94] Senge, R. & Hüllermeier, E. (2014). Fast fuzzy pattern tree learning. *IEEE Transactions on Fuzzy Systems*. Submitted and under review. 37, 67

[95] Sra, S., Nowozin, S., & Wright, S. J. (2011). *Optimization for Machine Learning*. 5

[96] Steele, G. L. (1990). *Common LISP: The Language*. Digital Press. 108

[97] Stufflebeam, J. & Prasad, N. R. (1999). Hierarchical fuzzy control. In *Proceedings of 1999 IEEE International Fuzzy Systems Conference*, volume 1, (pp. 498–503). 103

[98] Suárez, A. & Lutsko, J. F. (1999). Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(12), 1297–1311. 105

[99] Takagi, T. & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, *SMC-15*(1), 116–132. 10, 36, 100

[100] Thompson, S. (1999). *Haskell: The Craft of Functional Programming*, volume 2. Addison-Wesley. 109

[101] Torra, V. (2002). A review of the construction of hierarchical fuzzy systems. *International Journal of Intelligent Systems*, 531–543. 103

[102] Tunstel Jr., E. W. (1996). *Adaptive Hierarchy of Distributed Fuzzy Control: Application to Behavior Control of Rovers*. PhD thesis. 103, 104

[103] Turing, A. (1936-7). On computable numbers, with an application to the entscheidungsproblem (1936). *Proceedings of the London Mathematical Society*, *42*, 230–265. 1

[104] Van Buuren, S. (2012). *Flexible Imputation of Missing Data*. Chapman & Hall. 21

[105] Vapnik, V., Levin, E., & Cun, Y. L. (1994). Measuring the VC-dimension of a learning machine. *Neural Computation*, *6*, 851–876. 26

[106] Vapnik, V. N. & Chervonenkis, Y. A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & its Applications*, *16*(2), 264–280. 26

[107] Vapnik, V. N. & Kotz, S. (1982). *Estimation of Dependences Based on Empirical Data*, volume 41. Springer-Verlag New York. 26

[108] Wang, L.-X. (1999). Analysis and design of hierarchical fuzzy systems. *IEEE Transactions on Fuzzy Systems*, *7*(5), 617–624. 104

[109] Wang, L.-X. & Mendel, J. M. (1992). Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, *22*(6), 1414–1427. 10, 96

[110] Weber, R. (1992). Fuzzy-id3: A class of methods for automatic knowledge acquisition. In *Proceedings of the International Conference on Fuzzy Logic Neural Networks*, (pp. 265 –268). 10

[111] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 80–83. 64, 91

[112] Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2 ed.). Morgan Kaufmann. 29

[113] Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. 92

[114] Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, *18(1)*, 183–190. 15

[115] Yager, R. R. (1997). On a class of weak triangular norm operators. *Information Sciences*, *96*(1–2), 47 – 78. 18

[116] Yi, Y., Fober, T., & Hüllermeier, E. (2009). Fuzzy operator trees for modeling rating functions. *International Journal of Computational Intelligence and Applications*, *8*(4), 413–428. 13

[117] Yuan, Y. & Shaw, M. J. (1995). Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, *69*(2), 125–139. 105, 107

[118] Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353. 5

[119] Zadeh, L. A. (1979). A theory of approximate reasoning. *Machine Intelligence*, *9*, 149–194. 8

[120] Zadeh, L. A. (1999). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, *100*, 9–34. 8

Erklärung

Ich versichere, dass ich meine Dissertation “Machine Learning Methods for Fuzzy Pattern Tree Induction” selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Marburg,