

Charles University in Prague  
Faculty of Mathematics and Physics

## DOCTORAL THESIS



Jakub Šmíd

## Computational Intelligence Methods in Metalearning

Department of Theoretical Computer Science and Mathematical  
Logic

Supervisor of the doctoral thesis: Mgr. Roman Neruda, CSc.

Study programme: Computer Science

Study branch: Theoretical Computer Science

Prague 2016

Pursuing my doctorate has been a long time effort, and I was lucky that many people supported me along the way.

Roman Neruda, my supervisor, deserves credit for all his advice, inspiring thoughts, all those late night finishes of our conference papers, and amazing pancakes he would prepare for us during our stays abroad. Because of him, the whole period of my study was an extraordinary adventure and one hell of a ride.

I would also like to thank my present and former Ph.D. colleagues – Martin Pilát, Ondřej Kazík, Klára Pešková, Tomáš Křen, Josef Moudřík, Martin Šlapák, and Jiří Vytasil. I have enjoyed all our discussions and lunches at Konírna. I am really proud that I had a chance to be a member of the team.

No conference travels or logistics would be possible without the help of Petra Novotná from our faculty. Many thanks.

I am very grateful to my whole family for their ongoing support during my research, especially to my parents. They made this journey possible.

I would further like to express appreciation to my friends who were always there for me. Mainly, Dan Kobr, Veronika Šmídová, Jaroslav Švelch, Milan Plachý, and Tomáš Potužák.

I also deeply value the help of Roman Kučera from Ataccama with speeding up my experiments by kindly providing some extra computational power.

The biggest "Thank you" belongs to my soulmate Zónička. She has bravely supported me throughout the writing, and awaits me every time I return from a conference. She is my favourite hello and hardest goodbye.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, Czech Republic, June 13, 2016

---

Jakub Šmíd

Název práce: Metody výpočetní inteligence pro metaučení

Autor: Jakub Šmíd

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí disertační práce: Mgr. Roman Neruda, CSc., Ústav informatiky AV ČR, v.v.i.

**Abstrakt:** Tato práce je zaměřena na problematiku výběru algoritmu, která má za cíl doporučit algoritmus strojového učení k nové úloze. Řešení problému vychází z myšlenky, že se algoritmy chovají podobně na podobných datech. Tato podobnost je často založena na extrakci pevného počtu metaatributů z každé úlohy. Vzhledem k tomu, že počet atributů se u různých úloh typicky liší, ztrácíme tak důležité informace. V této práci popíšeme třídu algoritmů, která dokáže zpracovat také informace o jednotlivých attributech. Naše metody jsou založeny na přiřazování atributů. Výsledná vzdálenost mezi úlohami je dána jako součet vzdáleností mezi atributy určenými optimálním přiřazením. Dále dokážeme, že za určitých podmínek můžeme zaručit, že výsledná vzdálenost mezi úlohami je metrika. Provedeme sadu experimentů na datech extrahovaných z OpenML repositáře. Vytvoříme vzdálenost mezi atributy prostřednictvím genetických algoritmů, genetického programování a několika regularizačních technik, jako je koevoluce a zavedení vícekriteriality. Výsledky experimentů naznačují, že výsledná vzdálenost mezi úlohami může být úspěšně použita na problematiku výběru algoritmu. Ačkoliv jsme naše metody použili výhradně k metaučení, lze je aplikovat i v jiných oblastech. Navržené algoritmy jsou aplikovatelné kdekoliv, kde máme definovanou vzdálenost mezi prvky nějaké množiny a potřebujeme navrhnout vzdálenost mezi prvky potenční množiny původní množiny.

**Klíčová slova:** Metaučení, Strojové učení, Metriky, Genetické algoritmy, Přiřazování atributů

Title: Computational Intelligence Methods in Metalearning

Author: Jakub Šmíd

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Institute of Computer Science, AS CR

Abstract: This thesis focuses on the algorithm selection problem, in which the goal is to recommend machine learning algorithms to a new dataset. The idea behind solving this issue is that algorithm performs similarly on similar datasets. The usual approach is to base the similarity measure on the fixed vector of metafeatures extracted out of each dataset. However, as the number of attributes among datasets varies, we may be losing important information. Herein, we propose a family of algorithms able to handle even the non-propositional representations of datasets. Our methods use the idea of attribute assignment that builds the distance measure between datasets as a sum of distance given by the optimal assignment and an attribute distance measure. Furthermore, we prove that under certain conditions, we can guarantee the resulting dataset distance to be a metric. We carry out a series of metalearning experiments on the data extracted from the OpenML repository. We build up attribute distance using Genetic Algorithms, Genetic Programming and several regularization techniques such as multi-objectivization, coevolution, and bootstrapping. The experiment indicates that the resulting dataset distance can be successfully applied on the algorithm selection problem. Although we use the proposed distance measures exclusively on metalearning, it is possible to use our methods even beyond this task. The algorithms can handle every situation where we have a notion of distance between elements of some set and are looking to define a distance on the power set of the original set.

Keywords: Metalearning, Machine Learning, Metric, Genetic Algorithms, Attribute Assignment

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Outline of the Thesis . . . . .	5
<b>2</b>	<b>Preliminaries</b>	<b>7</b>
2.1	Datasets . . . . .	7
2.2	Machine Learning Tasks . . . . .	8
2.3	Metalearning . . . . .	11
2.3.1	Metalearning Task . . . . .	11
2.3.2	Types of Recommendation . . . . .	13
2.4	Types of Metafeatures . . . . .	15
2.5	Distance Based Ranking . . . . .	18
2.6	Assessing the Ranking Quality . . . . .	21
2.7	Ranking Baseline . . . . .	23
2.8	Performance Indicators . . . . .	24
2.9	Systems for Algorithm Recommendation . . . . .	27
2.10	Related Work . . . . .	30
2.11	Principal Component Analysis . . . . .	31
<b>3</b>	<b>Global Distance</b>	<b>33</b>
3.1	Metric Spaces . . . . .	33
3.1.1	Metric Examples . . . . .	34
3.2	Distance Using Global Metadata . . . . .	40
<b>4</b>	<b>Attribute Assignment</b>	<b>42</b>
4.1	Dealing with Unstructured Data . . . . .	43
4.1.1	Word Embeddings . . . . .	43
4.1.2	Kernels . . . . .	44
4.2	Non-propositional Approach to Metalearning . . . . .	49
4.3	Distance Using Attributes . . . . .	50
4.4	Examples . . . . .	58
4.5	Theoretical Properties . . . . .	61
4.6	Distance Using Attribute Metadata . . . . .	70
4.7	Combining the Distances . . . . .	71
4.8	Normalization Based on the Number of Attributes . . . . .	72
<b>5</b>	<b>Obtaining the Data</b>	<b>74</b>
5.1	ARFF Format . . . . .	74
5.2	Machine Learning Repositories . . . . .	76
5.3	OpenML Dump . . . . .	77

5.4	Metadata . . . . .	80
5.4.1	Global Metadata . . . . .	80
5.4.2	Attribute Metadata . . . . .	82
5.4.3	Normalization . . . . .	85
<b>6</b>	<b>Experiment Proposal</b>	<b>90</b>
6.1	Optimization . . . . .	91
6.2	Genetic Algorithms . . . . .	91
6.2.1	Selection . . . . .	93
6.2.2	Mutation . . . . .	94
6.2.3	Modifications . . . . .	94
6.3	Experiments . . . . .	94
6.3.1	Results . . . . .	99
6.3.2	Visualisation of the Distance . . . . .	101
<b>7</b>	<b>Metric Relaxation</b>	<b>106</b>
7.1	Metric Spaces Revisited . . . . .	107
7.1.1	Attribute Assignment with Relaxed Attribute Measure . .	108
7.2	Genetic Programming . . . . .	109
7.2.1	Initialization . . . . .	110
7.2.2	Crossover . . . . .	110
7.2.3	Mutation . . . . .	111
7.2.4	Bloat Problem . . . . .	111
7.3	Experiment Proposal . . . . .	113
7.3.1	Functions . . . . .	114
7.3.2	Terminals . . . . .	115
7.3.3	Algorithm Specification . . . . .	115
7.3.4	Results . . . . .	117
<b>8</b>	<b>Regularization</b>	<b>120</b>
8.1	GP Modifications . . . . .	120
8.1.1	Bootstrapping . . . . .	120
8.1.2	Coevolution . . . . .	121
8.2	Multi-objectivization . . . . .	122
8.2.1	Multi-objective Optimization . . . . .	122
8.2.2	Multi-objective Evolutionary Algorithms . . . . .	123
8.3	Experiments . . . . .	124
8.3.1	Coevolution . . . . .	124
8.3.2	Antibloat . . . . .	127
8.3.3	Results . . . . .	127
8.3.4	Multi-objectivization . . . . .	129
<b>9</b>	<b>Conclusion</b>	<b>132</b>
9.1	Future Work . . . . .	133
	<b>List of Figures</b>	<b>145</b>
	<b>List of Algorithms</b>	<b>148</b>

<b>List of Tables</b>	<b>149</b>
<b>Acronyms</b>	<b>151</b>
<b>Attachments</b>	<b>153</b>

# Chapter 1

## Introduction

The ongoing rapid growth of the available amount of data drives the urge for automated processing of such data. *Data mining* – the means of finding new patterns in *datasets* – is now widely used in medicine, economics, bioinformatics and other important areas of human interest. Many different algorithms exist and are used for this task of pattern extraction. However, even for an expert, it is hard to choose the most suitable algorithm for a particular dataset. According to the *No Free Lunch* theorem (NFL) stated in [124], the average performances of data mining algorithms on all data mining problems are equal. It means that elevated performance of any algorithm over one class of problems is paid for in performance over another class. However, NFL considers all possible problems. We are mostly interested in so called *real world* problems. Therefore, the key to success when dealing with a data mining real world problem is in binding the problem with an algorithm having elevated performance on the class of the problem. Since many fields depend on data mining techniques, it is crucial to propose and improve such bindings.

*Metalearning* [16] – the learning how to learn – can tackle the issue. The main idea behind metalearning is that *machine learning* methods are supposed to perform similarly on similar datasets. Therefore, the notion of dataset similarity is crucial. In most cases, the similarity is computed using data characterizing the datasets – the *metadata*. Metalearning techniques use the metadata and previous experience to predict the performance of machine learning methods on new datasets. In essence, metalearning does not differ much from the traditional machine learning. The main difference is that metalearning works with the metadata of given datasets instead of the actual data, and that the result of metalearning is a recommendation of a machine learning method to use.

The metadata may contain general information about the dataset, like the number of instances and attributes, the number of classes, performance of some algorithm over the dataset, etc. There is, however, a fixed amount of such general information extracted out of each dataset. Many traditional methods compare datasets based on such measures.

The attribute-specific metadata include more fine-grained information about the dataset. On the other hand, each dataset can have a different number of attributes (and consequently, different amount of attribute-specific metadata), which leads to a non-propositional representation of the datasets using attribute-specific metadata. The problem of defining the similarity on the non-propositional

space is non-trivial as stated by Kalousis in [59]. The authors of [16] also explicitly recognize the problem of handling datasets described by the varying amount of metafeatures. In the past there were only few attempts to define the non-propositional similarity. Furthermore, most of these attempts loose important information in the process.

One of the goal of this thesis is to propose algorithms capable of handling such a non-propositional representation. We take into consideration other methods dealing with non-propositional data, either directly in the metalearning field or in other fields as well. Another goal of this thesis is to investigate theoretical properties of proposed distance measures and compare them with the related state of the art methods. We investigate whether the methods satisfy *metric* axioms, alternatively, what properties must be satisfied in order to do so.

The performance of the proposed methods should be evaluated on data. Several machine learning repositories are reviewed. The emphasis is put on the amount of data provided, types of metadata available, and the fact whether the data are publicly accessible allowing for the independent re-evaluation of our results.

## 1.1 Outline of the Thesis

The structure of the thesis is as follows. Chapter 2 introduces the field of metalearning. Different types of scenarios addressed by metalearning are presented. The literature that covers different ideas and advances in metalearning is reviewed. The algorithm ranking based on distance measures is addressed in more depth. Unified workflow for measuring the quality of distance based algorithm ranking is presented. We also cover some of our contributions related to metalearning that are not directly related to the main goals of the thesis. This includes our recommendation multi-agent system *Pikater* and a *hierarchical clustering* approach for metalearning.

Chapter 3 reviews the metric spaces and several well known facts about metrics spaces. We look into the commonly used dataset distance measures based on the fixed amount of metadata extracted from datasets and what properties are necessary in order to get a metric.

The main contribution of the thesis is presented in Chapter 4. First, we discuss the motivation behind dealing with objects with variable structure and review some work not directly related to metalearning, which however served as an inspiration for the work in this thesis. We also discuss why we cannot use such techniques to address the variable amount of attributes when comparing the datasets. We review state of the art approaches to handle the attribute-specific metadata in metalearning and discuss their strengths and weaknesses. The idea of attribute assignment is presented. The main idea lies in defining attribute distance measure and aligning the attributes as best as we can. The desired distance between datasets is the sum of distances between aligned attributes. Several algorithms based on this idea are proposed. We also show what properties we have to maintain in order to get the metric on the dataset space. We also discuss other ideas we considered, which could, however, violate some of metric axioms.

Chapter 5 discusses data needed to conduct our experiments. Several databases of machine learning datasets are reviewed including those containing results

of machine learning algorithms over those datasets. We discuss the metadata available about the datasets and establish attribute-specific metadata that we will extract ourselves in order to be able to build attribute assignment models.

The first batch of experiments is proposed in Chapter 6. All the pieces and algorithms are glued together and the workflow is reviewed as a whole. Experiments are proposed in such a way that we always get a metric on the dataset level. We let the *Genetic Algorithms* [50] optimize weights of the attribute assignment techniques. We also review the complexity of the whole workflow that will be important when proposing the experiment settings. Our experiments will be actually solving *reinforcement learning* [104] task with very rare feedback from the environment. The results of experiments are discussed and compared between themselves and a *baseline* algorithm.

Chapter 7 introduces more expressive language to describe attribute distance measure while relaxing the requirement on dataset metric. We will show that the relaxation will guarantee a semimetric on the attribute space. We discuss whether the missing triangle inequality is necessarily important for the distance measure. We will also prove that the semimetric on the attribute level lead to a semimetric on the dataset level using our attribute assignment workflow. We conduct another batch of experiments with the new algorithms, and discuss their results.

In Chapter 8, we discuss set of techniques that could improve the generalization abilities of some of our models. In particular, we discuss coevolution, bloat control and bootstrapping of the population as an extension of the genetic algorithms. We also discuss using *multi-objectivization* to split the objective function into two. The second – added – criterion is the metric similarity. We also define the *multi-objective optimization* and describe one algorithm in particular – *NSGA-II* [32]. We conduct experiments with coevolution, bootstrapping and bloat control and compare their results with the previous experiments. We review one of our previous works with multi-objectivization.

Chapter 9 concludes the thesis. The goals of the thesis are evaluated based on the expectations. The future work and possible opportunities to improve the results are outlined.

# Chapter 2

## Preliminaries

This chapter introduces metalearning – learning how to learn. We begin with defining instances of data called datasets using a relation theory. We also discuss possible domains of relations and establish some restrictions for the rest of the thesis. The space of datasets is defined. We elaborate on specific problems that are connected with defining a structure on this space. Machine learning tasks are introduced and divided into three basic types – *supervised*, *unsupervised* and *reinforcement learning* – based on the amount of feedback received from the environment. Then, we discuss the means to evaluate the performance of machine learning algorithms over machine learning tasks, namely *root mean squared error*, *F-measure* and *predictive accuracy*. We discuss the existence of algorithm outperforming all other algorithms. We elaborate on the No Free Lunch theorem [124] and its implication on the machine learning tasks. The work of Smith-Miles et al. is reviewed [116] as an interesting example of finding algorithms dominating a subset of data. We define the task of metalearning and we provide several examples of it – dynamic parameter adjustment, and recommending machine learning algorithms for a given – not yet seen – task. The different algorithm recommendation are analysed, depending on the amount of information one requires from recommendation techniques - single algorithm, subset of them, more detailed ranking or even estimating the performance of algorithms. We discuss how generic methods for algorithm ranking should look like and define distance based ranking. We also review some of the related literature including our own results with hierarchical clustering [67]. Means of measuring the quality of a ranking are provided. The very simple and often effective baseline algorithm for the ranking task is introduced. The different types of performance indicators are reviewed together with some interesting approaches how to combine accuracy and time. Different types of metadata are defined. State of the art means of utilizing such metadata for the sake of algorithm recommendation are reviewed. We introduce a concept of automated systems able to handle the whole task of metalearning. Our own recommendation system Pikater is reviewed. We discuss PCA [56] method that can be used for visualization of multi-dimensional data.

### 2.1 Datasets

Dataset describes the instance of data. It can be a relation, set of graphs, collection of texts or database of proteins. Throughout this thesis, unless mentioned

otherwise, we will consider the dataset to be of a form of relation [23].

**Definition 1.** Relation is a set of tuples  $(d_1, \dots, d_n)$ , where each element  $d_j$  is a member of  $D_j$ , a data domain or data type.

Given the relation, the dataset itself has a structure in the sense that every row is a vector of the same size and from the domain given by  $D_1, \dots, D_n$ . Alternatively, one can look at a dataset as a list of columns where every column has a domain and a collection of values. Sizes of all collections are the same.

We will specify three basic allowed supertypes:

1. *Categorical* supertype - domain is a finite set of values sometimes referred to as labels.
2. *Numerical* supertype - domains are real numbers –  $\mathbb{R}$ .
3. *Integer* supertype - domains are integer numbers –  $\mathbb{N}$ .

Based on these supertypes, we will put domain restrictions on datasets that will be considered in this thesis. Every domain must be one of the defined supertype, possibly equal to the supertype or with some other restrictions (e.g. specified minimum, maximum). In this sense, the numerical supertype is also a supertype of the integer supertype. We have decided to distinguish this special case, as additional useful properties may be defined for the integer supertype only. We will also allow for some values to be undefined (missing).

We will define *dataset space* as the space of all possible datasets. It is important to consider that now the dataset space is merely a set without any added structure. Although every dataset has a structure defined by the relation, one cannot easily follow this when proposing a structure on the dataset space for the following reasons:

- Number of columns may differ: the arity of relations representing datasets may vary.
- Domains may differ: domains of columns may be different.
- Number of rows may differ: the cardinality of relations may differ.

Adding the structure into the dataset space is one of the main topics in the rest of the thesis.

## 2.2 Machine Learning Tasks

*Machine learning* is a subfield of computer science. We say that algorithm is learning if it improves its performance on future tasks after making observations (receiving feedback) about the world [104]. According to the type of feedback, we can distinguish three main types of machine learning tasks.

**Definition 2.** In unsupervised learning, no explicit feedback is provided.

A common unsupervised learning task is *clustering*, when we try to sort input data to potentially interesting clusters. For example, an algorithm can eventually develop a concept of sunny and rainy days without ever being told this distinction.

On the opposite side stands the task of supervised learning, when we provide a teacher knowing the correct answers. Given a dataset, set of columns (usually only one) is designated as a target.

**Definition 3.** *The task of supervised learning is this:*

*Given a training set of  $n$  example input-output pairs (rows of dataset)*

$$(x_1, y_1), \dots (x_n, y_n),$$

*where  $y_i$  is a vector of values of target columns of row  $i$ ,  $x_i$  is a vector of values of the remaining columns, each  $y_i$  was generated by an unknown function  $y = f(x)$ , discover a function  $h$  that approximates the true function  $f$ .*

The function  $h$  is called *hypothesis*. The goal of learning is to search space of possible hypotheses finding the one that performs well, even on new examples beyond the training set. Note that in more general case, the input-output pair  $x$  and  $y$  do not have to be numbers or vectors, they can be labels, graphs or even more complex objects.

If the target attribute is only a single column, basic types of supervised learning can be defined by distinguishing the tasks according to the domain of target column  $y$  – classification or regression. In classification, we are assigning some labels to the input  $x$ . For example, when forecasting tomorrow’s weather based on today’s lookout, we could assign either sunny, cloudy or rainy.

**Definition 4.** *Classification is a supervised learning task where the output  $y$  is one of the finite set of values. If the cardinality of the set is 2, we say it is a binary or boolean classification.*

On the other hand, in regression we are estimating some numeric value like tomorrow’s temperature given today outlook.

**Definition 5.** *Regression is a supervised learning task where the output  $y$  is a real number.*

To measure the quality of some hypothesis, the *testing set* of previously unseen examples is given to the hypothesis. We say that  $h$  *generalizes* well if it correctly predicts the value  $y$  for novel examples in the testing set. Sometimes, the learning algorithm exhibits low error on the training set while having poor generalization abilities. This phenomenon is called as *overfitting* as the algorithm learned properties that are only specific for the training set. Usually, the algorithm capable of creating complex models tend to overfit on the simple data. To tackle this, the concept of *Occam’s razor* is often used. If we have two hypothesis having similar performance, we should prefer the model with the lower complexity [104].

There are many ways of how to measure the quality of the hypothesis. We will list a few:

**Definition 6.** *Root mean squared error (RMSE) of some hypothesis  $h$  is defined as*

$$RMSE(h) = \sqrt{\sum_{i=1}^n \frac{(h(x_i) - f(x_i))^2}{n}},$$

where  $n$  is the number of samples in the testing set.

In the binary classification we can define precision and recall:

**Definition 7.** Precision and recall are defined as

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

where  $tp$  stands for true positive – number of cases the classifier answered 1 correctly. Analogically, we can define  $fp$  and  $fn$  – false positives and false negatives. They denote number of cases when classifier incorrectly answered 1 or 0 respectively.

The common way to measure the quality of binary classifier is F-measure – the harmonic mean of precision and recall.

**Definition 8.** F-measure of the hypothesis is

$$F(h) = 2 \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

In classification tasks, we can define the percentage of classes that were classified correctly or incorrectly resulting into the Predictive Accuracy and Error Rate.

**Definition 9.** Predictive Accuracy is defined as

$$\text{PredictiveAccuracy} = \frac{\#(h(x_i) = f(x_i))}{n},$$

where  $n$  is the number of samples in the testing set. Error Rate is defined as

$$\text{ErrorRate} = 1 - \text{PredictiveAccuracy} = \frac{\#(h(x_i) \neq f(x_i))}{n},$$

where  $h$  is the hypothesis,  $f$  is the true function,  $n$  is the number of samples and  $\#$  denotes the number of cases for which the argument is True.

In the real world scenario, the reward is not often associated with every action. We often get a reward (or punishment) after a series of decisions. For example, if somebody wants to get from one place A to another place B, with every step taken he does not know whether the step was right. He can evaluate his decisions after he successfully gets to place B for example based on the time, money and energy needed to get to the destination. This concept is formalized as a third learning scenario – reinforcement learning:

**Definition 10.** The basic reinforcement learning model consists of:

1. A set of environment states  $S$ .
2. A set of actions  $A$ .
3. Rules of transitioning between states.
4. Rules that determine the scalar immediate reward of a transition.
5. Rules that describe what the agent observes.

## 2.3 Metalearning

This section describes the task of metalearning. Intuitively, it is a task of learning how to learn. We will discuss the metalearning task in general and then move to one particular subset of metalearning tasks – algorithm recommendation. We will create a taxonomy of recommendation based on properties we would like to recommend.

### 2.3.1 Metalearning Task

In the recent years, many algorithms were proposed to solve different machine learning tasks. Usually, every such algorithm has many parameters and its performance is heavily dependent on settings of those parameters. Even for an expert, it is sometimes difficult to choose the right algorithm and set up the parameters correctly. Furthermore, scientists from different fields than computer science face great challenges when trying to employ machine learning to solve their tasks. It would be useful if such a system was provided that would either solve the machine learning tasks directly or would assist in choosing the right algorithms and parameters. Such a tool should learn from a previous experience. The common idea behind such recommendation tool is based on the idea that algorithms perform similarly on similar datasets.

One could also improve based on the past experience in adjusting the parameters. For example, if one non-deterministic machine learning experiment did not achieve desired performance for hundreds of attempts with the same parameter settings, it could be perhaps a good idea to try different settings next time. Also, if some indicators would suggest that the algorithm was being stuck in some local optimum, the parameters of the algorithm could be adjusted ad hoc, so the local optima are avoided and algorithm hopefully converges to a global optimum. The authors of [120] review the dynamic control of parameters for many nature inspired algorithms.

In transfer learning, we try to transfer some learned properties to a new – similar – task. For example, authors of [126] argue that deep neural networks often learn a generally useful concept in the top layers and that such knowledge can be transferred to other tasks. The ability to transfer knowledge depends on the similarity of tasks.

These scenarios have some common patterns. In each case above, we are utilizing some extra knowledge – previous experience or notion of dataset similarity – usually referred to as *metaknowledge*. This continuous process of learning how to learn is often denoted as metalearning. In [16], we can find more formal definition of metalearning:

**Definition 11.** *Metalearning is the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes.*

In this thesis, we will focus on the first case – the *algorithm selection* – in which we use metaknowledge to recommend the best algorithms, parameters or both to some machine learning task. The problem was originally stated in 1976 by Rice [102] and received much attention from the research community ever since. Rice

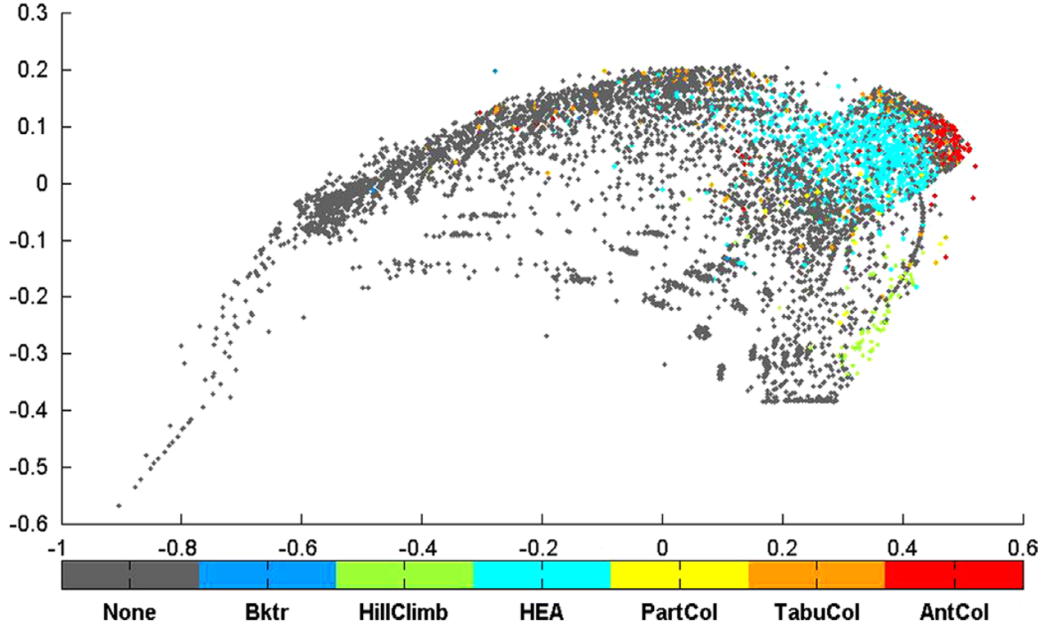


Figure 2.1: Space of optimisation problems was projected to two dimensions by the PCA algorithm and coloured according to the best algorithm in that area. Grey colour is for the instances that had multiple best algorithms (according to some small margin) [116].

devised a general framework to select the best algorithm for a problem at hand. He suggested to extract set of *metafeatures* out of a given problem and use these metafeatures to select an algorithm that maximizes the expected performance of the algorithm on the problem at hand.

One could naturally ask whether such recommendation is needed in the first place. Perhaps there exists an algorithm that is outperforming every other algorithm on every task. The answer is no. According to the *No Free Lunch* theorem stated in [124], the average performances of data mining algorithms on all data mining problems are equal. It can still be the case that there are some areas that are dominated by certain algorithms.

For instance, in [116], Smith-Miles et al. investigated different instances of optimization problems and features describing differences between these instances. The PCA (see Section 2.11) was used to map features into two dimensional space. Another instances were generated so the projected two dimensional space was better covered. The search algorithms were employed to find instances that are hard for each solver and provided a map of search space based on the expected foot-steps of different optimization algorithms. The projected space of instances were coloured according to colour of the dominating optimization technique on that space. This gave a nice overview of which part of two dimensional space computed by the PCA algorithm was dominated by which optimisation algorithm. This can be seen in Figure 2.1.

Another way to look at the No Free Lunch theorem is the fact that lots of problems from the set of all data mining problems may not be particularly interesting. One could argue that only problems related to some useful problem are worth considering. This argument is supported by the observation that usually

when one learns something, he expects that the function being learned has some nice properties. For instance, it is continuous and close points have the same or similar value. For example, a car does not stop being a car if it changes a colour. One would have to make multiple adjustments to destroy the car properties. This arguing gives the notion to vaguely defined *real world* problems. It follows that one should not be too concerned about the No Free Lunch theorem as it may still be the case that there is an algorithm outperforming every other algorithms on the real world problems.

In the rest of the thesis, we will solely focus on the algorithm selection problem.

### 2.3.2 Types of Recommendation

We can distinguish different types of algorithm selection [16] based on what exactly we want to predict – the *metatarget*:

1. Best in set.
2. Best subset.
3. Ranking.
4. Performance prediction.

The simplest case is choosing the best algorithm among the set of algorithms. This case has the advantage that it can be formulated as a classification problem. The major disadvantage is that if this algorithm produces unsatisfactory result, the recommendation system does not clue any steps to take further on. One may also choose to predict the best parameter for a certain algorithm.

More complicated case is when one is interested in some smaller subset of algorithms or parameters. The algorithms selected are those performing well – this is often described as performing well within a margin. In the case of classification, the margin can be defined as

$$\left[ e_{min}, e_{min} + k \sqrt{\frac{e_{min}(1 - e_{min})}{n}} \right), \quad (2.1)$$

where  $e_{min}$  is the error of the best algorithm,  $n$  is the number of examples and  $k$  is a user-defined parameter determining the size of the margin.

Other alternative it to carry out statistical testing. The selected algorithms will be those with not significantly worse performance than the best. Best subset address some flaws of the first type, however user does not have any guidance with the order of algorithms to try. Similarly, there is no guidance if all algorithms from the subset were tried.

In ranking, the goal is to rank (sort) algorithms or parameters according to their expected performance. In this case the exact performance value is not important, one is simply interested in the rank itself. One would naturally expect complete total ordering between algorithms or parameters. This does not have to be the case, sometimes this level of granularity is not desirable. For instance, one can decide that the algorithms with the similar expected performance will be inserted to a certain class and define the ordering just between the different

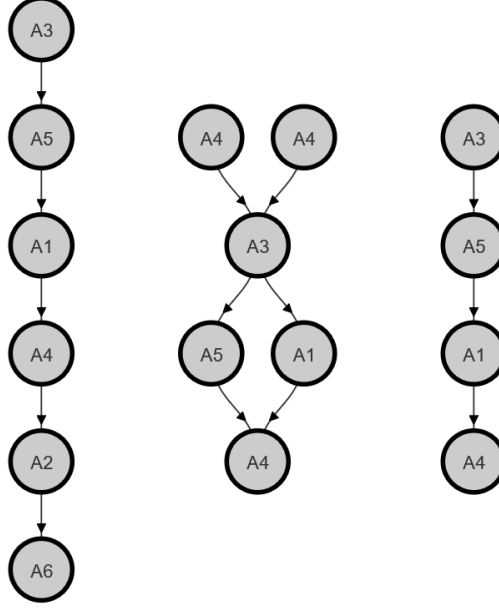


Figure 2.2: Types of ranking. The left one is linear complete, middle one is weak and complete and the ranking on the right is linear and incomplete. Adapted from [16].

classes of algorithms. The ordering between the algorithms of the same class could remain undefined.

Based on these, we can distinguish different types of ranking based on the level of granularity provided. We can recognize linear and weak rankings (whether ranking defines linear order on some set of algorithms) and complete and incomplete rankings (whether rankings include all algorithms or just a subset). Rankings can be represented by *Hasse diagrams* – examples are in Figure 2.2.

In some cases, one is not only interested in the ranking, but also in the performance estimation. This can be useful if there are some requirements for the minimal performance acceptable or if more detailed information about the expected outcome are required. It also makes sense if the estimated performance is the runtime of algorithm, as resources could be either limited or expensive, and algorithms that are expected to finish quickly are preferred. One could also expect that faster algorithms produce simpler hypothesis, thus arguing that on simple problems faster algorithms will have better generalization abilities by Occam’s razor. The performance estimations can be easily transformed to ranking by ordering the algorithms by their estimated performance.

Authors of [42] propose a practical model of *Evolutionary Program-induction Algorithms* (EPAs) including Genetic Programming (Section 7.2). The model corresponds to the following equation:

$$P(t) \approx a_0 + \sum_{p \in S} a_p d(p, t), \quad (2.2)$$

where  $a_i$  are coefficients,  $P(t)$  is a performance of an EPA on the target functionality  $t$ ,  $S$  is a subset of a program search space and  $d$  is a function of similarity between the output of the EPA and the target functionality  $t$ . The paper deals with the issue of determining the suitable coefficients and the suitable subset of

the search space. The model is tested on various tasks.

In [112, 113], we have predicted accuracy and expected time consumption for some algorithm on a new dataset using the data gathered by our recommender system Pikater (see Section 2.9). A unique non-propositional distance measure was used (see Algorithm 9). Estimation functions were evolved by Genetic Programming and were represented by two trees: one for estimating accuracy and another one for estimating time consumption. The GP algorithm included terminals representing distance and the performance result (either accuracy or duration depending on the tree being evolved) from one of the nearest dataset. Randomly initialized constant terminal specified which nearest dataset was used. The special terminal estimating complexity was proposed:

$$complexity = \sum_{a_i} \sigma(a_i) \log(n_{rows}), \quad (2.3)$$

where  $n_{rows}$  measures number of rows in the datasets,  $a_i$  is the  $i$ -th attribute and  $\sigma(a_i)$  estimates the attribute complexity based on its type:

$$\sigma(a_i) = \begin{cases} 1; & \text{if } a_i \text{ is a boolean or categorical attribute,} \\ 2; & \text{if } a_i \text{ is an integer attribute,} \\ 3; & \text{if } a_i \text{ is a real attribute.} \end{cases} \quad (2.4)$$

We argued that continuous and numerical attributes are usually hard to process, hence the increased coefficient. This complexity terminal proved especially useful for estimating the time needed to conduct the experiment. The example of an evolved recommendation tree is in Figure 2.3. The recommendation agent utilizing the evolved estimation trees was implemented into our custom metalearning system called Pikater (see Section 2.9).

## 2.4 Types of Metafeatures

As devised in [102], the extraction of metafeatures out of the problem at hand is needed for the algorithm selection problem. There are two main requirements for the metafeatures to extract – the efficiency of the extraction and the descriptive factor of the metafeatures. The efficiency requirement is clear. The exact performance of all algorithms could be extracted as a feature but that would not give us any time savings at all. The question is what is an acceptable complexity. For instance, article [92] even suggests to restrict the complexity to  $\mathcal{O}(n \log n)$  as bigger complexities are too expensive and it would be better to devote the time saved to already running some algorithms. We argue that the exact value depends on the algorithms in portfolio. With lots of algorithms and increased complexity of algorithms in portfolio, it still makes sense to compute even more expensive metafeatures. For example, if the portfolio contains NP-Complete problems, it should not be of concern to compute polynomial metafeatures. Furthermore, the suggestion is from the year 2000 and the significant increase of the CPU power and the ability to get more computational power instantly by using virtual servers in the cloud calls for not being so strict in limiting the complexity of computing the metafeatures.

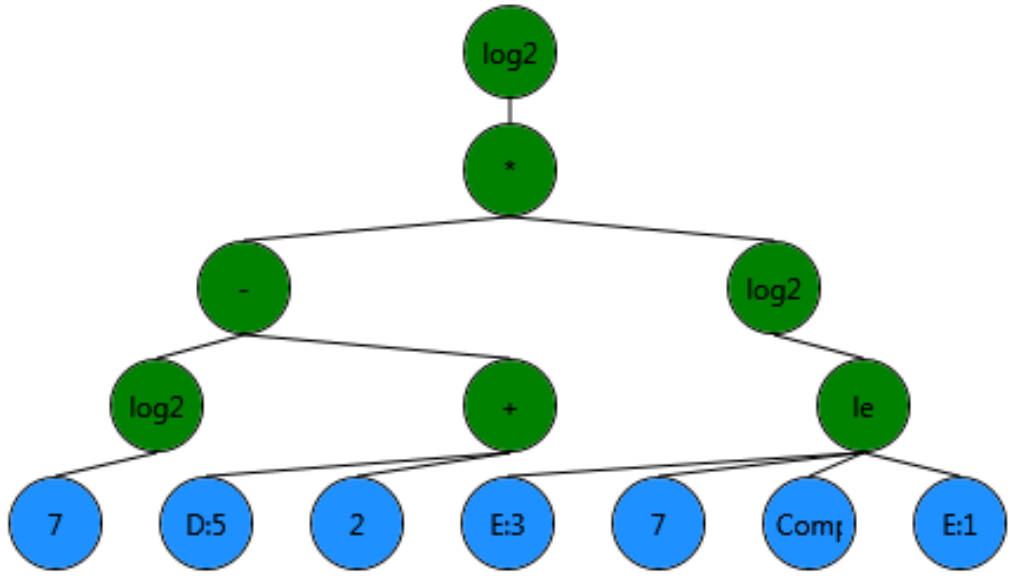


Figure 2.3: Example of the tree for estimating the algorithm duration on a new dataset. Terminals are blue, functions are green. The node labelled  $D:i$  represents distance from the  $i$ -th nearest dataset. Similarly, the node labelled  $E:i$  denotes error of the algorithm on the  $i$ -th nearest dataset.  $Com$  is the complexity terminal.

The second requirement also makes sense, as it is important to use the metafeatures that somehow help to distinguish between different algorithms. The rest is just noise and should be ignored, as even the computation of such noise can be costly.

We can distinguish metafeatures according to the nature of their computation.

The first category builds on statistic and information theory. Article [48] distinguishes three subcategories of the first category: *simple*, *statistical* and *information-theoretic* measures. The simple measures include only very basic ones, such as the number of rows, the number of attributes or the number of integer attributes. Among the statistical ones, the article ranks such metafeatures as skewness, correlation, and kurtosis. Information-theoretic measures are motivated by information theory and are mostly appropriate for discrete attributes. The example of information-theoretic feature is *entropy* of a discrete random variable:

$$H(X) = - \sum_i q_i \log_2 q_i, \quad (2.5)$$

where  $q_i$  is the probability that a variable  $X$  takes on the  $i$ -th value. Conventionally, the entropy is converted to  $\log_2$  as the measured value is in bits.

Authors of [16] recognize two additional types of metadata – *model-based* and *landmarkers*. The model-based metafeatures lies in training some sort of model on the data and take some properties of the model as metafeatures. For example, the decision tree [99] predicting the target is built. The number of nodes in the tree is introduces as a new metafeature. The landmarks are a quick estimate of algorithm performance on the dataset. There are two ways how to do this. Either a simplified version of some algorithm is used to build the model on the whole data. For instance, the decision tree only with the top node is created and its performance evaluated on the data. Alternatively, *subsampling landmarks* use the whole algorithm on the subset of the data, which gives again the estimate.

From now on, we will treat dataset, unless stated otherwise, as the entity described by its metafeatures. We will distinguish two types of metafeatures. Propositional metafeatures (fixed size vector of metafeatures describing dataset as a whole) will be referred to in this thesis as *global metafeatures*. The second type we will recognize are *attribute metafeatures* – that is the set of metafeatures extracted for each attribute in the dataset. As the number of attributes can vary per dataset, attribute metafeatures are non-propositional as we will have a vector of variable length of vectors of metafeatures. The vector of metafeatures does not have to be necessarily of a fixed size. Different types of attributes can have different types of metafeatures. We will address this in the later chapters. The set of metafeatures will be also called metadata throughout the thesis. It can be argued that reshuffling the attributes of the datasets does not change the dataset at all. Therefore, we will consider two datasets represented by their metadata equal if and only if:

1. All available global metadata are equal.
2. All available attribute metadata are equal, or there exists a permutation (reshuffling) of attributes of the first dataset such that attribute metadata are equal after the reshuffling.

## 2.5 Distance Based Ranking

In the previous sections, we have discussed the problem of algorithm ranking. The usual approach to the ranking is based on the idea that algorithms perform similarly on similar datasets. If we want to exploit this idea, we need two things – a notion of distance between datasets and a way of calculating a ranking from the previous results on datasets similar to a dataset at hand using the distance between datasets. In this section, we will focus on the latter and suppose that we already have the notion of distance. Throughout this thesis, when outlying the pseudocode of some algorithm, we will treat dataset distance measure as an interface taking two datasets and outputting a real number – the measured distance between the both input datasets. This is formalized in Algorithm 1.

---

**Algorithm 1:** *IDatasetDistance*: dataset distance interface

---

```
// Interface for measuring distance between two datasets.
input :  $a \leftarrow$  First dataset
input :  $b \leftarrow$  Second dataset
output:  $d \in \mathbb{R}, d$  is a distance between  $a, b$  ( $d = \Delta(a, b)$ )
```

---

This will enable us to plug different algorithms as the part of others. We will see lots of algorithms conforming to the *IDatasetDistance* interface throughout the thesis. To avoid confusion, we will always use the symbol  $\Delta$  when referring to the *IDatasetDistance* interface in equations or other algorithms.

To allow for obtaining the ranking, we will define another interface called *IRanking* – the one that just takes a dataset and returns the ranking of algorithms to the given dataset. The interface is intentionally very generic, so we can reuse it later even with the non-distance based ranking.

---

**Algorithm 2:** *IRanking*: Interface for ranking calculation

---

```
// Interface for calculating the ranking to a new dataset.
input :  $d \leftarrow$  dataset to rank
output: Ranking – ordering of algorithms
```

---

One could wonder how to implement *IRanking* interface using distance based ranking. The *IRanking* interface provides just a dataset on the input. But for the distance based ranking, at least the distance measure and other datasets need to be provided.

To solve this, we will use a concept from functional programming. *Functional programming* [40] is a programming paradigm – a style of building the structure and elements of computer programs – that treats computation as the evaluation of mathematical functions. In this thesis, we will be proposing generic layered architecture of algorithms. As we want to design generic interface but the sub-components will be initialized and parametrized differently, we will use specific functional programming construct know as partial application.

**Definition 12.** Partial application *refers to the process of fixing a number of arguments to a function, producing another function of smaller arity. Given an*

integer  $k$  and a function  $f : X_1 \times \dots \times X_k \times \dots \times X_n \rightarrow Y$ , the function  $\text{partial}_k$  is defined as a function  $X_1 \times \dots \times X_k \rightarrow (X_{k+1} \times \dots \times X_n \rightarrow Y)$ . The  $k$  is often omitted as it can be inferred from the number of input arguments.

This will enable us to preinitialise some parameters and the remaining not yet assigned parameters will define the generic interface.

Because of the *partial* application, we have an elegant way to solve the issue. We can define an interface taking all the necessary information to build up the distance based ranking and then just use *partial* application to fix all the arguments except the dataset one. The resulting function now conforms to the *IRanking* interface. More formally, the *IDistanceRanking* interface is defined in Algorithm 3.

---

**Algorithm 3:** *IDistanceRanking*: Interface for distance based ranking calculation

---

```
// Interface for calculating the ranking to a new dataset
// based on distance.
input :  $\Delta \leftarrow IDatasetDistance$  - Dataset distance measure
input :  $\text{exp} \leftarrow$  Previous results
input :  $d \leftarrow$  Dataset to obtain ranking for
output: Ranking – ordering of algorithms
```

---

And the transformation of *IDistanceRanking* interface to a more generic *IRanking* interface is outlined in Algorithm 4.

---

**Algorithm 4:** Distance Ranking Transformation: transforming a *IDistanceRanking* interface to generic *IRanking* interface

---

```
// Interface for transforming IDistanceRanking to IRanking.
input :  $\text{distanceRanking} \leftarrow IDistanceRanking$  interface
input :  $\Delta \leftarrow IDatasetDistance$  - Dataset distance measure
input :  $\text{exp} \leftarrow$  Previous results
output: IRanking interface
1 return  $\text{partial}(\text{distanceRanking}, \Delta, \text{exp});$ 
```

---

Now we can propose several algorithms that can be implemented to conform to the *IDistanceRanking* interface.

The *k-Nearest Neighbours* algorithm (*k*-NN) [27] is widely used for ranking [16]. For instance, Maratea et al. use *k*-NN in their system for solving *answer set programming task* (ASP) [80, 81]. ASP is a declarative approach towards hard (mainly NP) search problems. The *k*-NN algorithm for ranking is aggregating previous results on the *k* nearest neighbours identified by the distance measure to estimate the ranking. The whole method is outlined in Algorithm 5. The complexity, provided that a distance is already precomputed, is  $\mathcal{O}(n_d \log(n_d) + n_a \log(n_a))$ , where  $n_a$  is the number of algorithms and  $n_d$  the number of datasets. If the ranking is called multiple times for different datasets, the computation of the distance could be repeated, therefore it is often useful to compute the distance outside of this function. This was the reason why we

stated the complexity for already precomputed distance. This will be our case as we need lots of ranking computation to evaluate the quality of ranking algorithm. This is mainly a technicality, so for the sake of simplicity, we will not state explicitly in the algorithms that we are computing something outside of the algorithm when plugging the algorithms into one another.

---

**Algorithm 5:** *K*-NN Ranking: *k*-NN based implementation of *IRanking*.

---

```

// Implementation of IDistanceRanking using k-NN algorithm
input :  $k \leftarrow$  Number of neighbours
input :  $\Delta \leftarrow$  IDatasetDistance Dataset distance measure
input :  $\text{exp} \leftarrow$  Previous results
input :  $d \leftarrow$  Dataset to obtain ranking for
output: Ranking

1 distances  $\leftarrow []$ ;
2 foreach dataset  $\in$  exp.datasets do
3   | current_distance  $\leftarrow \Delta(d, \text{dataset})$ ;
4   | distances += [(dataset, current_distance)];
5 end
6 distances  $\leftarrow$  sort(distances, (dataset, distance):distance);
7 neighbours  $\leftarrow$  distances[:  $k$ ];
8 foreach algorithm  $\in$  exp.algorithms do
9   |  $r_{\text{algorithm}} \leftarrow 0$ ;
10 end
11 foreach neighbour  $\in$  neighbours do
12   | foreach algorithm  $\in$  exp.algorithms do
13     |  $r_{\text{algorithm}} += \text{exp.rank}(\text{neighbour}, \text{algorithms})$ ;
14   | end
15 end
16  $R \leftarrow [1, \dots, \text{len}(\text{exp.algorithms})]$ ;
17  $R \leftarrow \text{sort}(R, \text{algorithm: } r_{\text{algorithm}})$ ;
18 return  $R$ ;

```

---

The *k*-NN algorithm can be also modified in such a way to involve the weights into computation, so that closer neighbours have bigger influence on the final result [35]. The number of neighbours – *k* – is the crucial parameter of the whole algorithm. Setting of *k* to 1 or similarly low values usually results in overfitting and corresponding bad generalization abilities of the model. On the other hand, high values of *k* defeat the purpose of the local neighbourhood and tend to have lower performance as the information from the distant regions affect the decision process. To address this, there has been also some additional modifications proposed to the *k*-NN algorithm. The *G-means* algorithm [45] enhances the original algorithm by automatically setting the number of neighbours. The G-means is based on a statistical test for the hypothesis that a subset of data follows a Gaussian distribution. The G-means runs *k*-means with increasing *k* in a hierarchical fashion until the test accepts the hypothesis that the data assigned to each *k*-means centre are Gaussian. Two key advantages are that the hypoth-

esis test does not limit the covariance of the data and does not compute a full covariance matrix. The *G*-means algorithm was used for algorithm selection by authors of *Autofolio* [77]. The *G*-means is used together with other algorithms for automatic selection of parameters to solve some artificial intelligence problem at hand (e.g. *satisfiability* (SAT), *constraint satisfaction programming* (CSP) or *quantified boolean formula* (QBF)). In [58], the *G*-means is used in algorithm portfolio selection of the SAT solvers. Motivated by the observation that solvers have complementary strengths and therefore exhibit incomparable behaviour on different problem instances, algorithm portfolios run multiple solvers in parallel or select one solver, based on the features of a given instance.

In [67], we investigated the possibility of distance based ranking using clustering constructed from the training set. In our case, the clustering was a result of an *agglomerative clustering algorithm* [29]. The advantage of this clustering method is that it does not require creation of centroids as in the case of other methods. Compared to the *k*-NN algorithm, the neighbourhood does not have a fixed size but rather a variable size depending on the cluster method.

The bottom up method was chosen because it is faster than the top down method. The question arises which criterion of clusters' distance to use. We have adopted *Unweighted Pair Group Method with Arithmetic Mean* (UPGMA), or average linkage clustering [29], which is used in various applications [34].

In our experiments, ranking was constructed the same way as in the *k*-NN ranking algorithm with the exceptions that the nearest neighbours of the given dataset were selected according to the nearest cluster, given metric and UPGMA method. In the experiments, we have used the results of 8 *Weka* [44] algorithms on the 85 *UCI* (Dataset Repository of University of California, Irvine) [12] datasets. The datasets were divided into the training and testing set with the ratio of 2:1. We have tested various distance metric to estimate the algorithm performance. Results indicated that such hierarchical clustering can be successfully used for metalearning. The best dendrogram of the best cluster is shown in Figure 2.4.

## 2.6 Assessing the Ranking Quality

In this section, we describe techniques to assess the quality of some method for predicting the ranking of algorithms. Given some *IRanking* interface implementation and a new dataset *d*, the *Spearman's rank correlation coefficient* can be used to assess the accuracy of the ranking method:

$$r_s^d = 1 - \frac{6 \sum_{i=1}^n (R_i'^d - R_i^d)^2}{n^3 - n}, \quad (2.6)$$

where *n* is the number of models,  $R_i^d$  is the actual rank of model *i* on dataset *d* and  $R_i'^d$  is the predicted rank of model *i* on dataset *d*. The Spearman's rank correlation coefficient has some interesting properties. The range of the coefficient is normalized to the interval  $\langle -1, 1 \rangle$ , where 1 is a perfect match, -1 is a perfect mismatch and 0 indicates results as good as random guessing. Equation 2.6 gives us the means of measuring ranking quality for a single dataset. If the aim is to measure the quality over all datasets, it is possible to take Spearman's rank

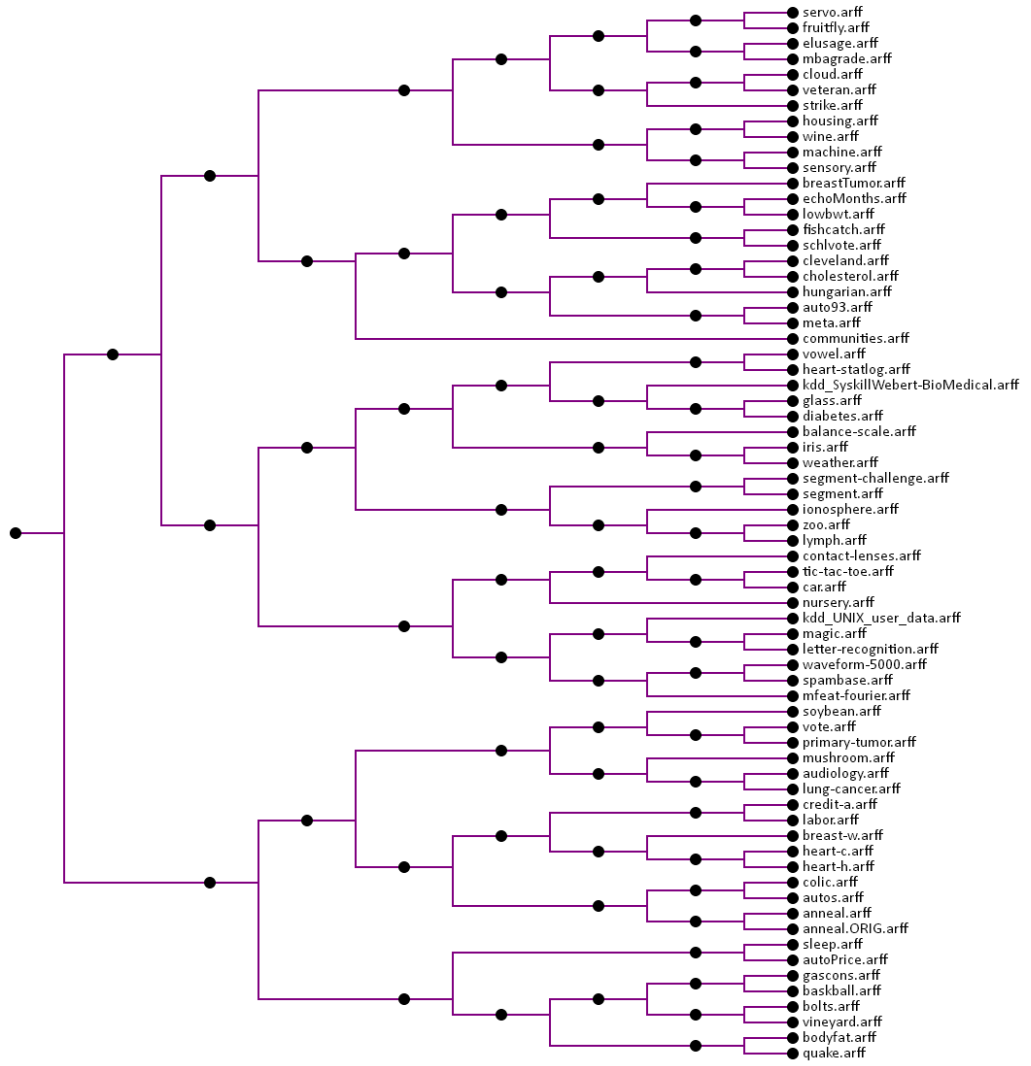


Figure 2.4: The dendrogram as a result of the agglomerative clustering [67]. Datasets are assigned to clusters according to their similarity.

correlation coefficient averaged over all datasets:

$$r_s = \frac{\sum_{d=1}^n r_s^d}{n}, \quad (2.7)$$

where  $n$  is the number of datasets and  $r_s^d$  is the Spearman's rank correlation coefficient computed for dataset  $d$ .

The whole ranking quality assessment is summarized in Algorithm 6.

---

**Algorithm 6:** Ranking Quality Assessment

---

```

// Pseudocode for ranking prediction quality assessment
using dataset distance function.
input : datasets  $\leftarrow$  List of datasets
input : predictor  $\leftarrow$  IRanking Ranking predictor
input : results  $\leftarrow$  Results of algorithms on datasets indexed by
        datasets
input : algorithms  $\leftarrow$  List of algorithms
output: Ranking Prediction Quality Assessment

1  $r_s \leftarrow 0$ ;
2  $n \leftarrow \text{len}(\text{algorithms})$ ;
3 foreach dataset  $\in$  datasets do
4   remaining  $\leftarrow$  datasets  $\setminus$  dataset;
5    $R^{\text{dataset}} \leftarrow \text{predictor}(\text{dataset}, \text{remaining}, \text{results}, \text{algorithms})$ ;
6    $r_s^{\text{dataset}} \leftarrow 1 - \frac{6 \sum_{i=1}^n (R_i^{\text{dataset}'} - R_i^{\text{dataset}})^2}{n^3 - n}$ ;
7    $r_s \leftarrow r_s + \frac{r_s^{\text{dataset}}}{n}$ ;
8 end
9 return  $r_s$ ;

```

---

Basically, the algorithm for each dataset estimates the ranking of algorithms by using *IRanking* interface. The quality of rankings is measured using Equation 2.7. Provided the predictions are already precomputed, we get a complexity of the quality evaluation:

$$\mathcal{O}(n_d n_a),$$

where  $n_d$  is the number of datasets and  $n_a$  is the number of algorithms.

## 2.7 Ranking Baseline

Algorithm 6 can give us a good estimate of how good our model is compared to random guessing. However, there may be trivial methods that are very easy to implement, yet they exhibit a relatively good performance. Such trivial algorithms are often referred to as *baseline* algorithms. Their output is usually based on some statistical knowledge about the data and is calculated using the most frequent values. Complex algorithms should outperform baseline algorithms, otherwise there is no need for the extra complexity. In the classification tasks, the

baseline usually outputs the most frequent class. In the regression task, the median or average of the target values is usually returned by the baseline algorithm. We can also define a baseline algorithm for the ranking task. The baseline algorithm can predict the ranking results based on algorithm’s average rank on all datasets present in the metaknowledge base. If one algorithm is the best in average on all datasets, baseline algorithm will just assign the first rank to this algorithm. Again, we will design the baseline algorithm with the *IRanking* interface in mind. To calculate the ranking, baseline needs the information about the average ranking. One could be tempted to use again the trick with the *partial* function. In this case this would be a mistake, as that would result in recalculating the average ranking for each call to the interface. Instead, we will create a function that takes previous results as an input, calculates average ranking, and then returns a function conforming to the *IRanking* interface. Such course is outlined in Algorithm 7. The complexity of resulting *IRanking* is constant. To build up the distance, one has to do

$$\mathcal{O}(n_d n_a + n_a \log(n_a))$$

steps, where  $n_d$  is the number of datasets and  $n_a$  is the number of algorithms. The term  $(n_d n_a)$  represents the cost of two inner loops, the term  $n_a \log(n_a)$  is for sorting the rankings.

## 2.8 Performance Indicators

In the previous sections, we have discussed the metatarget and ranking in particular. We did not discuss what should be the performance indicator defining the metatarget. It can be any of the performance measures mentioned in Section 2.2 or a combination of them. The time complexity can also be predicted or time can be included into the performance consideration. This is useful when good and fast learning algorithms are preferred. The time is especially crucial when the resources are scarce or expensive.

The time cost value of following some ranking strategy is captured by the *top-N* evaluation [18]. The  $N$  is a parameter that determines how many best algorithms will be tried on some dataset. Different settings of  $N$  can be consequently simulated, and we can observe how the accuracy of the best model improved compared to the time cost associated with the increase of  $N$ .

The *top-N* strategy has a disadvantage that it is unable to leverage what is learned from previous evaluations. Alternatively, we can tackle this issue with a strategy called *active testing* [74]. It proceeds in a tournament-style fashion, in each round selecting and testing the algorithm that is most likely to outperform the best algorithm of the previous round on the new dataset. The next contender is selected based on the concept of *relative landmarks* [41]. These landmarks estimate the relative probability that a particular algorithm will outperform the current best candidate. The *cross-validation* [104] of the new contender is performed. The result is added to the relative landmarks and the contender replaces the current best algorithm if the cross-validation result is better or ties the result of current best algorithm.

One approach of combining time and accuracy was proposed by Brazdil et al. [17]. The  $k$ -Nearest Neighbours algorithm [27] with a distance function based

---

**Algorithm 7: Ranking Baseline**

---

```
// Pseudocode for building the ranking baseline model.
input : datasets  $\leftarrow$  List of datasets
input : algorithms  $\leftarrow$  List of machine learning algorithms
input : results  $\leftarrow$  Results of algorithms on datasets indexed by datasets
output: IRanking interface

1 averageRanking  $\leftarrow$  [];
2 occurrences  $\leftarrow$  [];
3 foreach algorithm  $\in$  algorithms do
4   | averageRanking[algorithm]  $\leftarrow$  0;
5   | occurrences[algorithm]  $\leftarrow$  0;
6 end
7 foreach dataset  $\in$  datasets do
8   | resultsOnDataset  $\leftarrow$  results[dataset];
9   | foreach algorithm  $\in$  algorithms do
10  |   | if algorithm  $\in$  resultsOnDataset then
11  |   |   | occurrences[algorithm]++;
12  |   |   | currentRank  $\leftarrow \frac{\text{resultsOnDataset[algorithm]}}{\text{len(resultsOnDataset)}}$ ;
13  |   |   | averageRanking[algorithm] += currentRank;
14  |   | end
15  | end
16 end
17 averageRankingWithAlgorithm  $\leftarrow$  zip(averageRanking, algorithms);
18 averageRankingWithAlgorithm  $\leftarrow$ 
    map(averageRankingWithAlgorithm,  $\lambda(x, y) \rightarrow \frac{x}{\text{occurrences}[y]}, y$ );
19 averageRankingWithAlgorithm  $\leftarrow$ 
    sort(averageRankingWithAlgorithm,  $(x, y) : x$ );
20 return  $\lambda(x) \rightarrow$  averageRankingWithAlgorithm;
```

---

on a set of statistical, information theoretic, and other dataset characterization measures is employed in order to identify the set of similar already computed tasks. For the ranking phase, the adjusted ratio of ratios ranking method is presented, which processes performance information based on accuracy and time.

The relevance of the processed dataset  $d_i$  to the dataset  $d_j$  at hand is defined in terms of similarity between them, according to metafeatures. It is given by a metric (or a distance function):

$$\Delta(d_i, d_j) = \sum_x \sigma(v_{x,d_i}, v_{x,d_j}), \quad (2.8)$$

where  $d_i$  and  $d_j$  are datasets,  $v_{x,d_i}$  is the value of metafeature  $x$  for dataset  $d_i$ , and  $\sigma(v_{x,d_i}, v_{x,d_j})$  is the distance between the values of metafeature  $x$  for datasets  $d_i$  and  $d_j$ . All metafeatures are normalized.

The  $k$ -Nearest Neighbours algorithm is then used to identify the  $k$  cases nearest to the dataset at hand.

The *adjusted ratio of ratios* uses information about accuracy and execution time to rank the given classification algorithms. It is computed by means of the auxiliary term  $ARR_{a_p,a_q}^{d_i}$  defined as:

$$ARR_{a_p,a_q}^{d_i} = \frac{\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}}{1 + \frac{\log\left(\frac{T_{a_p}^{d_i}}{T_{a_q}^{d_i}}\right)}{K_T}}, \quad (2.9)$$

where  $SR_{a_p}^{d_i}$  and  $T_{a_p}^{d_i}$  are the success rate and duration of algorithm  $a_p$  on the dataset  $d_i$ , and  $K_T$  is a user-defined value that represents the amount of accuracy the user is willing to trade for a 10 times speed-up or slowdown. For example,  $K_T = 10\%$  means that the user is willing to trade 10% of accuracy for 10 times speed-up.

Finally, the overall mean adjusted ratio of ratios for each algorithm is derived:

$$ARR_{a_p} = \frac{1}{m-1} \left( \sum_{a_q} \frac{\sum_{d_i} ARR_{a_p,a_q}^{d_i}}{n} \right), \quad (2.10)$$

where  $m$  is the number of algorithms and  $n$  is the number of datasets. The ranking is based on this measure. Authors of [2] including one of original authors of *ARR* looks into the *ARR* measure and argue that *ARR* should be monotonically increasing. Higher success rate ratios should lead to higher values of *ARR* and, similarly, higher time ratios should lead to lower values of *ARR*. Experiments were proposed to verify this property on data. The  $\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}$  was fixed to 1, the time ratio was sampled from very small  $2^{-20}$  to very high values  $2^{20}$  and three different values of  $K_T$  were used (0.2, 0.3, 0.7). The resulting *ARR* function was not monotonic and was even approaching infinity at some point. In general, this can lead to incorrect rankings provided by the metalearner and can affect the evaluation results. Authors proposed a solution that addresses this issue by

changing the re-sampling used in *ARR*. The updated formula *A3R* was proposed:

$$A3R_{a_p, a_q}^{d_i} = \frac{\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}}{\sqrt[n]{\frac{T_{a_p}^{d_i}}{T_{a_q}^{d_i}}}}, \quad (2.11)$$

where  $n$  is a user defined constant representing the importance of time.

The *A3N* is used in [3] as a performance indicator. The paper proposes several modification to active testing strategy [74]. The first approach uses faster sample-based tests to identify competitive algorithms. The second modification lies in argument that full cross-validation test is not necessary to estimate the next candidate. Instead, the test is performed on a smaller sample of the new dataset. This is motivated by the fact that a sample-based test is much faster than a full cross-validation test. The full cross-validation test is carried out only if a candidate algorithm beats the currently best algorithm on the sample-based test.

## 2.9 Systems for Algorithm Recommendation

Having a viable framework for metalearning is only one side of a coin. The actual systems built for metalearning are called *recommendation system*. Their goal is to allow for a trade-off between human time and machine time. They can also provide the power of machine learning algorithms to the non-expert users, even to those with limited computer knowledge. There are many scenarios for such systems.

*Auto-weka* [118] is a system integrating into the Weka toolkit [44] and it is able to perform model selection, and autotune model parameters. It employs the *Sequential Model-based Algorithm Configuration algorithm* (SMAC) [52]. SMAC supports a variety of models of the type  $p(c|\lambda)$  to capture the dependence of the loss function  $c$  on hyper-parameters  $\lambda$ , including approximate Gaussian processes [101] and random forests [19].

Autofolio [77] combines SMAC with algorithm selection framework claspfolio2 [51]. *Claspfolio2* is a flexible framework that provides functionality to train and assess the performance of different algorithm selection techniques. The extensibility is the main advantage and it provides an unified framework for algorithm selection problem. The overview of all components is in Figure 2.5. The scheduling system is also integrated.

Our recommendation system called *Pikater* [91, 65, 66, 64] is implemented using *multi-agent systems* (MAS). To be precise, the multi-agent framework *JADE* [13] is used as a platform of the system. The extensibility is assured by the use of the structured ontology language and following the *Foundation for Intelligent Physical Agents* (FIPA) [96] international standards of agents' communication. The following basic scenarios have been considered.

- In the most simple case, the user knows which method and what parameters of that method they would like to use.

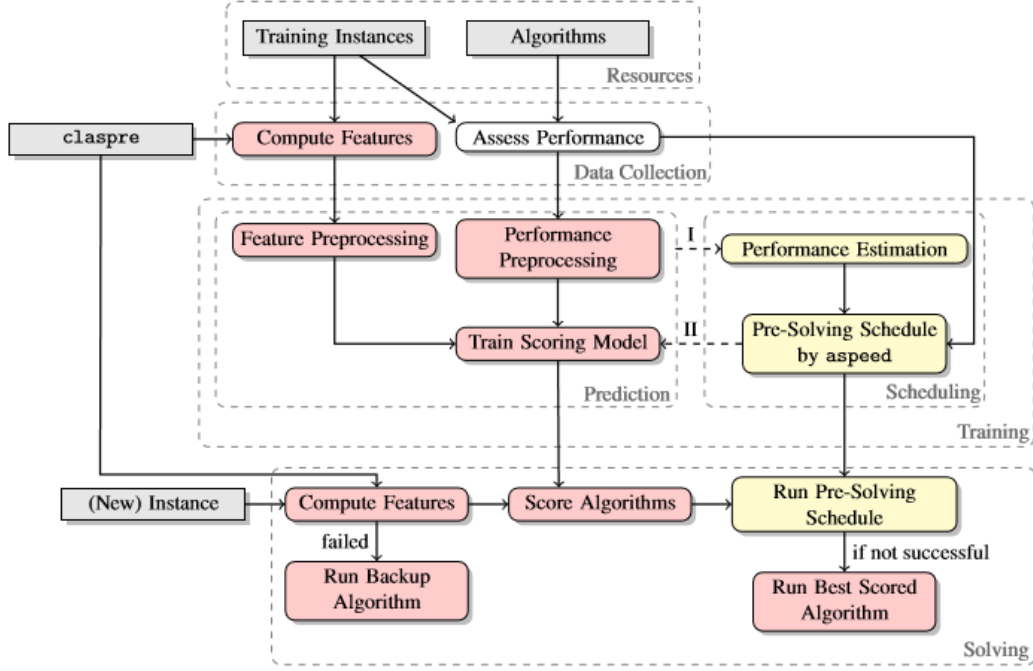


Figure 2.5: Overview of claspfolio2 framework [51].

- In the second scenario, the user knows what method to use but does not know how to set its parameters. The system is able to search the parameter space of the method and find a setting that provides good results.
- In the third case, the user does not even know what method to use and lets the system decide by itself. In this case, the system recommends the best possible method or provides a ranking of the methods based on predicted errors and duration.

These simple scenarios can be extended into more complex ones. For instance, it is also possible to combine the recommendation of the best method with parameter space search, when the system recommends an interval of the parameter’s values. As a positive side effect, the metaknowledge base for metalearning purposes is being built up by each experiment.

In order to effectively design our system, we have chosen the organization-centred formalism *AGR* (Agent-Group-Role) [38]. The role is a set of capabilities and responsibilities that the agent accepts by handling the role. Group — the building block of a MAS — is a set of agents with allowed roles and interactions, defined by a group structure. The multi-agent system then consists of multiple groups which can overlap, as agents can belong to more than one group. In this formalism, we abstract from the algorithmic details and inner logic of the agents in the MAS. Authors of [64] used the ontological formalism of *OWL-DL* [109] to describe the organizational model of Pikater. The following group structures were defined according to the aforementioned scenarios: administrative group structure, computational group structure, search group structure, recommendation group structure, data group structure, and data-management group structure. Our MAS is composed of groups that are instances of these group structures. The architecture is depicted in Figure 2.6.

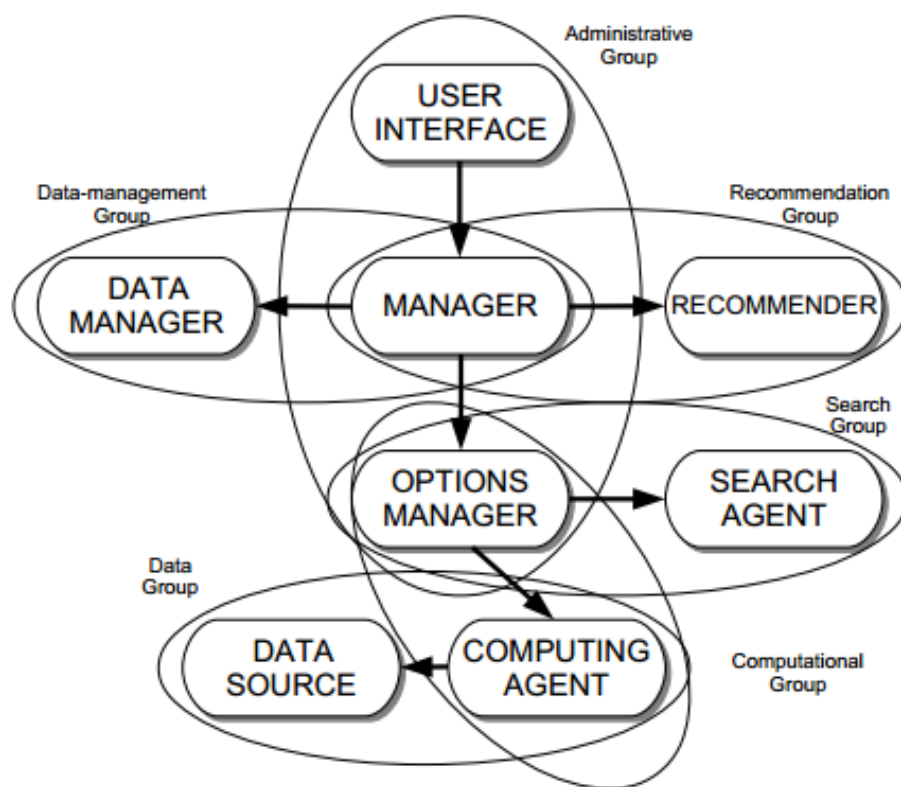


Figure 2.6: Overview of Agent-Group-Role model of our recommendation system Pikater [91].

The MAS-based solution allows for a flexibility in choice of the parameter space search algorithms, each of these is encapsulated in a search agent. General tabulation, random search, simulated annealing [104], or parallel methods, such as genetic algorithms, are implemented in our system. Another great benefit of the agent-based approach is the natural capability to parallelly execute multiple computations with various parameters. This significantly decreases the time needed for the execution of the parameter space search process. One of essential features of our MAS is its capability of recommending a suitable computational method for a new dataset, according to datasets similarity and previously gathered experience. The choice of the similar dataset(s) is based on various previously proposed metrics [29], which measure the similarity of their metadata. Our database contains over 600,000 records, that are used to suggest the proper method (including its parameters) and estimate its performance on a new dataset. The latest version of our MAS contains the following types of recommenders, which differ in the metric used and in the number of recommended methods they provide:

- Basic recommender chooses a method based on the single closest dataset using the unweighted metadata metric.
- Clustering Based Classification [67] chooses the whole cluster of similar datasets and the corresponding methods, using different sets of metadata features.
- Evolutionary Optimized Recommenders are similar to the two above described recommender types, using different weighted metrics, optimized by an evolutionary algorithm.
- Interval Recommender recommends intervals of suitable parameter values and leaves their fine-tuning to the parameter space search methods.

Another functionality of our system is a multi-objective optimization of data mining configurations. To test this, the search algorithm is employed in order to find beneficial combinations of pre-processing and machine learning methods to the presented data. The minimization is performed in error-rate as well as run-time criteria [63].

## 2.10 Related Work

In this section, we further review some of the literature related to metalearning. Authors of [62] define the distance between datasets based on the following metadata:

- *Number of attributes* in data,
- *Number of instances*,
- *Data type* (Relates to all values of all attributes in the dataset. Four categories of data are considered – categorical, integer, real, or multivariate – different attributes are of different types),

- *Default task type* (Set by the user, the most common types are classification and regression),
- *Missing values* (Flag whether data contains unknown or unspecified values).

The following metric is defined between two tasks based on their metadata:

$$\Delta(\text{dataset}_1, \text{dataset}_2) = \sum_{i=1}^n w_i \sigma_i(\text{dataset}_1[i], \text{dataset}_2[i]), \quad (2.12)$$

where  $\text{dataset}_j[i]$  is the  $i$ -th metadata of the  $j$ -th dataset,  $w_i$  is the weight of the  $i$ -th metafeature and  $\sigma_i$  is the distance on the  $i$ -th metafeature defined by the type of the  $i$ -th metafeature. For categorical or boolean type the  $d_i$  is 0 if the value matches, otherwise 1. For the numerical types of metadata the normalized difference between their value is taken as a distance.

There is also a work by Graff and Poli [43] based on evolutionary program induction. They use (among other techniques) genetic programming to predict the performance of various algorithms, such as neural networks, to solve given tasks.

Kordík and Černý [68] propose the evolution of so called *templates*. Templates specify the workflow to produce a model and are the collection of ensembling algorithms, modelling and classification algorithms combined in a hierarchical manner. These templates are evolved using genetic algorithms in order to produce the best results. The similarity of templates can be used as a landmarker metadata.

Misir and Sebag [87] used a completely different approach. They formulated a more general problem of algorithm selection as a *collaborative filtering problem* [73]. In this case, instead of talking about the selection of methods for given dataset, we can imagine that the various methods rate the datasets based on their performance. The methods prefer the datasets, on which they have better performance. Interestingly, such an approach does not require any metadata, it is possible to run a few of the methods, find their performance and use this information to recommend better methods. However, if some metadata are available, they can be used instead of running the methods.

## 2.11 Principal Component Analysis

In this section, the *Principal Component Analysis* (PCA) [56] method is described, as it will be used further in the thesis.

PCA is an orthogonal linear transformation to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Given some dataset, we begin the PCA by subtracting the mean of each dimension from the dimension itself. We then calculate the covariance matrix of the data. The next step is calculation of the eigenvectors and eigenvalues of the covariance matrix. This is possible since the covariance matrix is square. This gives us the components. If we order the eigenvectors by eigenvalues, we get the

ordering by significance. We can reduce the dimension by ignoring the components with less significance. Suppose we have decided to keep  $k$  components. Now we form the *FeatureVector* as follows:

$$FeatureVector = (eigenvector_1 \dots eigenvector_k).$$

The *FeatureVector* is a matrix with eigenvectors as columns. We get the data in the new coordinate system by the following operation:

$$NewCoordinates = FeatureVector^T \times DataAdjustedByMeans^T,$$

where *DataAdjustedByMeans* are the original data with the means subtracted.

It is also possible to calculate points back and forth between the original coordinate systems and the new one. This is useful when we add some new data (regardless of the encoding).

PCA is often used for visualizing multi-dimensional data, as it can be used to reduce the number of dimension to two or three dimensions.

# Chapter 3

## Global Distance

In the previous chapter, the concept of distances between datasets was introduced. In this chapter, we will discuss what are the properties of good distance measures in general. We will define a *metric* and *metric spaces*. We will also define a concept of *norms*, which can be intuitively used to form a metric. We give a few examples of norms – *p-norms* and their weighted counterparts, and we outline a recognized fact that *p-norms* are indeed a norms using the *Hölder's* and *Minkowski's* inequalities. We summarize a few well known theorems about metric spaces – that it is possible to rescale the metric space without violating the metrics axioms and that the sum of metric is also a metric. These facts will be useful in the chapters to follow. We move back to dataset distance measures and define a class of such distance measures using the global metafeatures. We will show that if the distance is based on weighted *p-norms*, then the resulting dataset distance is a metric.

### 3.1 Metric Spaces

In this section, we will define metric, metric spaces and norms. We will also present few examples of the metrics.

**Definition 13.** A metric on a set  $X$  is a function (called the distance function or simply distance)

$$d : X \times X \rightarrow [0, \infty), \quad (3.1)$$

and for all  $x, y, z$  in  $X$ , the following conditions are satisfied:

1.  $d(x, y) \geq 0$  (non-negativity).
2.  $d(x, y) = 0 \Leftrightarrow x = y$  (coincidence axiom).
3.  $d(x, y) = d(y, x)$  (symmetry).
4.  $d(x, z) \leq d(x, y) + d(y, z)$  (triangle inequality).

**Definition 14.** Metric space is a tuple  $(X, d)$ , where  $M$  is a set and  $d$  is a metric on  $M$ .

The first condition of Definition 13 is sometimes omitted as the following holds:

**Theorem 1.** *Metric conditions 2, 3 and 4 imply condition 1.*

*Proof.*

$$d(x, x) \leq d(x, y) + d(y, x) \quad |triangle\ inequality \quad (3.2)$$

$$d(x, x) \leq 2d(x, y) \quad |symmetry, 3.2 \quad (3.3)$$

$$0 \leq 2d(x, y) \quad |coincidence, 3.3 \quad (3.4)$$

$$d(x, y) \geq 0 \quad |3.4 \quad (3.5)$$

□

In this thesis, we will prefer to use all four conditions, as it will later enable us to relax the definition of a metric a little bit, and propose additional algorithms for solving the algorithm ranking task.

### 3.1.1 Metric Examples

In this section, we will list few metric examples that will be later reused further in the thesis.

**Theorem 2.** *Let  $M = \mathbb{R}^n$  and  $x, y \in M$ . Then function defined as*

$$d(x, y) = \begin{cases} 0; & \text{if } x = y, \\ 1; & \text{otherwise.} \end{cases} \quad (3.6)$$

*is a metric on  $M$ .*

**Definition 15.** *We will denote the metric from Theorem 2 as the discrete metric. It can be easily verified that the discrete metric is indeed a metric.*

**Definition 16.** *Let  $V$  be a vector space over  $\mathbb{F}$  (with  $\mathbb{F} = \mathbb{R}$  or  $\mathbb{F} = \mathbb{C}$ ) and  $N : V \rightarrow \mathbb{R}$  a map such that, writing  $N(u) = \|u\|$ , the following results hold:*

1.  $\forall u \in V : \|u\| \geq 0$  (non-negativity).
2.  $\forall u \in V : \|u\| = 0 \iff u = \vec{0}$  (separates points).
3.  $\forall \lambda \in \mathbb{F}, \forall u \in V : \|\lambda u\| = |\lambda| \|u\|$  (absolute scalability).
4.  $\forall u, v \in V : \|u\| + \|v\| \geq \|u + v\|$  (triangle inequality).

*Then we call  $\|\cdot\|$  a norm and say that  $(V, \|\cdot\|)$  is a normed vector space.*

Normed vector space can be easily transformed into a metric space:

**Theorem 3.** *Let  $(V, \|\cdot\|)$  be a normed vector space. Then function  $d : V^2 \rightarrow \mathbb{R}$  defined as:*

$$d(u, v) = \|u - v\| \quad (3.7)$$

*is a metric on  $V$ . Consequently, the tuple  $(V, d)$  is a metric space.*

*Proof.* We will split the proof according to the different metric axioms.

1. We will begin with proving the coincidence axiom:

$$0 = d(v, w) \iff \|v - w\| = 0 \iff v - w = \vec{0} \iff v = w.$$

2. Symmetry can be derived using the second axiom of the norm:

$$d(v, w) = \|v - w\| = |-1| \|w - v\| = \|v - w\| = d(w, v). \quad (3.8)$$

3. A proof of triangle inequality uses the fourth norm property:

$$d(v, w) = \|v - w\| = \|(v - u) + (u - w)\| \leq \quad (3.9)$$

$$\leq \|(v - u)\| + \|(u - w)\| = d(v, u) + d(u, w). \quad (3.10)$$

4. The first metric axiom is implied by Theorem 1.

As we have proven every metric axiom, we can conclude the proof.  $\square$

**Definition 17.** For  $p \geq 1$ , the  $p$ -norm of  $x \in \mathbb{R}^n$  is defined as

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{1/p}.$$

To show that the  $p$ -norm is formally a norm, we will first state two theorems – Hölder's Inequality and Minkowski's Inequality.

**Theorem 4** (Hölder's Inequality). Let  $a, b$  be vectors in  $\mathbb{R}^n$  and  $p, q > 1, p, q \in \mathbb{R}$  satisfy  $\frac{1}{p} + \frac{1}{q} = 1$ . Then

$$\sum_{i=1}^n |a_i b_i| \leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} \left( \sum_{i=1}^n |b_i|^q \right)^{1/q}. \quad (3.11)$$

*Proof.* We will divide the proof into two steps. In the first step, we will prove an auxiliary claim that will be used in the step 2 to finally prove the desired inequality.

• Step 1. We shall show that if  $x, y > 0, x, y \in \mathbb{R}$  and  $0 < \lambda < 1, \lambda \in \mathbb{R}$  then

$$x^\lambda y^{1-\lambda} \leq \lambda x + (1 - \lambda)y. \quad (3.12)$$

Set  $t = \frac{x}{y}$ . Then after dividing both sides of the equation by  $y$ , we get the equivalent equation to prove  $t^\lambda \leq \lambda t + (1 - \lambda)$ .

Set  $\phi(t) = \lambda t + (1 - \lambda) - t^\lambda$ . Then we need to show that  $\phi(t) \geq 0$ . We will investigate the first derivative of  $\phi$ :  $\phi'(t) = \lambda - \lambda t^{\lambda-1} = \lambda(1 - t^{\lambda-1})$ , so

$$\phi'(t) = \begin{cases} < 0; & \text{if } t < 1, \\ = 0; & \text{if } t = 1, \\ > 0; & \text{otherwise.} \end{cases} \quad (3.13)$$

Since  $\phi(1) = 0$ , according to the derivatives this must be a global minimum of the function and therefore the step 1 is concluded.

- Step 2. Let us define  $A_i$  and  $B_i$  as

$$A_i = \frac{|a_i|^p}{\sum_{i=1}^n |a_i|^p}, B_i = \frac{|b_i|^q}{\sum_{i=1}^n |b_i|^q}. \quad (3.14)$$

Let  $\lambda = \frac{1}{p}$ . Then, by Step 1,

$$A_i^{1/p} B_i^{1/q} \leq \frac{A_i}{p} + \frac{B_i}{q} \quad (3.15)$$

as  $\frac{1}{q} = 1 - \frac{1}{p}$ . By fully expanding  $A_i$  and  $B_i$ , we obtain

$$\frac{|a_i|}{(\sum_{i=1}^n |a_i|^p)^{1/p}} \frac{|b_i|}{(\sum_{i=1}^n |b_i|^q)^{1/q}} \leq \frac{1}{p} \frac{|a_i|^p}{\sum_{i=1}^n |a_i|^p} + \frac{1}{q} \frac{|b_i|^q}{\sum_{i=1}^n |b_i|^q}. \quad (3.16)$$

By summing above equation over  $1, \dots, n$  we obtain

$$\frac{\sum_{i=1}^n |a_i| |b_i|}{(\sum_{i=1}^n |a_i|^p)^{1/p} (\sum_{i=1}^n |b_i|^q)^{1/q}} \leq \frac{1}{p} + \frac{1}{q} = 1. \quad (3.17)$$

Multiplying both sides by  $(\sum_{i=1}^n |a_i|^p)^{1/p} (\sum_{i=1}^n |b_i|^q)^{1/q}$  concludes the proof of the Hölder's Inequality.

□

**Theorem 5** (Minkowski's Inequality). *Let  $a, b$  be vectors in  $\mathbb{R}^n$  and  $p \geq 1, p \in \mathbb{N}$ . Then*

$$\left( \sum_{i=1}^n |a_i + b_i|^p \right)^{1/p} \leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |b_i|^p \right)^{1/p}. \quad (3.18)$$

*Proof.* If  $p = 1$  we can see that the inequality holds using triangle inequality:

$$|a_i + b_i| \leq |a_i| + |b_i|.$$

By summing up we get

$$\sum_{i=1}^n |a_i + b_i| \leq \sum_{i=1}^n |a_i| + \sum_{i=1}^n |b_i|.$$

If the  $p > 1$  define  $q > 1$  so that  $\frac{1}{p} + \frac{1}{q} = 1$ :  $q = \frac{p}{p-1}$ . We have that

$$\begin{aligned} \sum_{i=1}^n |a_i + b_i|^p &= \sum_{i=1}^n |a_i + b_i| |a_i + b_i|^{p-1} \leq \sum_{i=1}^n |a_i| |a_i + b_i|^{p-1} + \sum_{i=1}^n |b_i| |a_i + b_i|^{p-1} \\ &\leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} \left( \sum_{i=1}^n |a_i + b_i|^{(p-1)q} \right)^{1/q} + \left( \sum_{i=1}^n |b_i|^p \right)^{1/p} \left( \sum_{i=1}^n |a_i + b_i|^{(p-1)q} \right)^{1/q} \end{aligned} \quad (3.19)$$

The last inequality follows from the Hölder's Inequality (Theorem 4). Since

$$(p-1)q = p,$$

we may rewrite the inequality in Equation 3.19 as

$$\sum_{i=1}^n |a_i + b_i|^p \leq \left( \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |b_i|^p \right)^{1/p} \right) \left( \sum_{i=1}^n |a_i + b_i|^p \right)^{1/q}. \quad (3.20)$$

Dividing by  $(\sum_{i=1}^n |a_i + b_i|^p)^{1/q}$  we get

$$\left( \sum_{i=1}^n |a_i + b_i|^p \right)^{1-1/q} \leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |b_i|^p \right)^{1/p}. \quad (3.21)$$

Since  $1 - 1/q = 1/p$  we get

$$\left( \sum_{i=1}^n |a_i + b_i|^p \right)^{1/p} \leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |b_i|^p \right)^{1/p}. \quad (3.22)$$

which is the Minkowski's inequality.  $\square$

**Theorem 6.** *P-norm is a norm.*

*Proof.* We will split the proof according to the different axioms of the norms.

1. We will begin with the non-negativity. *P*-norm is a root of sum of absolute values raised to power *p*, therefore it cannot be negative.
2. Root is 0 if and only if the argument is zero. In our case, if the sum of absolute values raised to power *p* is zero. That is if and only if all the absolute values are zero, that is if and only if *x* is a zero vector.
3. Let  $\lambda \in \mathbb{R}$ . The absolute scalability follows from:

$$\begin{aligned} \|\lambda x\| &= \left( \sum_{i=1}^n |\lambda x_i|^p \right)^{1/p} = \left( \sum_{i=1}^n |\lambda|^p |x_i|^p \right)^{1/p} = \\ &= \left( |\lambda|^p \sum_{i=1}^n |x_i|^p \right)^{1/p} = |\lambda| \left( \sum_{i=1}^n |x_i|^p \right)^{1/p} = |\lambda| \|x\|. \end{aligned}$$

4. The triangle inequality results from:

$$\|u + v\| = \left( \sum_{i=1}^n |u_i + v_i|^p \right)^{1/p},$$

which is by Minkowski's Inequality less or equal to

$$\left( \sum_{i=1}^n |u_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |v_i|^p \right)^{1/p} = \|u\| + \|v\|.$$

As we have proven every norm axiom, we can conclude the proof.  $\square$

**Theorem 7.** Let  $M = \mathbb{R}^n, p \geq 1, p \in \mathbb{R}$  and  $x, y \in M$ . Then the function  $d_p : M \rightarrow \mathbb{R}$  defined as

$$d_p(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} \quad (3.23)$$

is a metric on  $M$ .

*Proof.* Follows from the fact that  $p$ -norm is a norm (Theorem 6) and the fact that derived metric according to Theorem 3 is a metric. This metric corresponds to the function  $d_p$  in question.  $\square$

We will define metrics for special values of  $p$  from Theorem 7:

**Definition 18.** Let  $d_p$  be a metric from Theorem 7. Then:

1. Let us call  $d_1$  defined as  $d_1(x, y) = \sum_{i=1}^n |x_i - y_i|$  the taxicab or Manhattan distance.
2. Let us call  $d_2$  defined as  $d_2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$  the Euclidean distance.
3. Let us call  $d_\infty$  defined as  $\lim_{p \rightarrow \infty} d_p(x, y) = \max(|x_i - y_i|)$  the Chebyshev distance or maximum metric.

Furthermore, we can weight the  $p$ -norms.

**Definition 19.** For  $p \geq 1, w \in \mathbb{R}^n$ , the weighted  $p$ -norm of  $x \in \mathbb{C}^n$  is defined as  $\|x\|_p = \left( \sum_{i=1}^n |w_i x_i|^p \right)^{1/p}$ .

We will show that weighted  $p$ -norm is a norm as long as weights are strictly positive:

**Theorem 8.** Let  $w = (w_1, \dots, w_n) \in \mathbb{R}^n$  and  $\forall w_i \in w : w_i > 0$ . Then weighted  $p$ -norm is a norm.

*Proof.* We will split the proof according to the different axioms of the norms.

1. We begin the proof by validating the non-negativity axiom.  $\forall w_i, x_i$  is  $|w_i x_i| \geq 0$ , therefore the chain of exponentiation, summing and  $p$ -th root will also be non-negative.
2. To prove the second axiom we have to verify both directions of the equivalence:
  - $\Rightarrow$  If weighted  $p$ -norm is zero, then all elements in the summation must be zero, and because weights are non-negative, all  $x_i$  must have been zero.
  - $\Leftarrow$  If  $x$  is zero vector then all elements in the sum are zero.

3. Let  $\lambda \in \mathbb{R}$ . The absolute scalability follows from:

$$\begin{aligned} \|\lambda x w^T\| &= \left( \sum_{i=1}^n |\lambda x_i w_i|^p \right)^{1/p} = \left( \sum_{i=1}^n |\lambda|^p |x_i w_i|^p \right)^{1/p} = \\ &= \left( |\lambda|^p \sum_{i=1}^n |x_i w_i|^p \right)^{1/p} = |\lambda| \left( \sum_{i=1}^n |x_i w_i|^p \right)^{1/p} = |\lambda| \|x w^T\|. \end{aligned}$$

4. The triangle inequality results from:

$$\|(u + v)w^T\| = \left( \sum_{i=1}^n |(u_i + v_i)w_i|^p \right)^{1/p} = \left( \sum_{i=1}^n |u_i w_i + v_i w_i|^p \right)^{1/p}, \quad (3.24)$$

which is an instance of Minkowski's inequality (Theorem 5) where  $a = u_i w_i$  and  $b = v_i w_i$ . Therefore Equation 3.24 is less or equal to:

$$\left( \sum_{i=1}^n |u_i w_i|^p \right)^{1/p} + \left( \sum_{i=1}^n |v_i w_i|^p \right)^{1/p} = \|u w^T\| + \|v w^T\|.$$

As we have proven every norm axiom, we can conclude the proof.  $\square$

Again, we can obtain weighted metrics from weighted  $p$ -norms through Theorem 3.

We will also state a few theorems that will be useful when we will propose the new algorithms. First, we will show that we can rescale the metric without breaking the axioms:

**Theorem 9.** *Let  $(M, \sigma)$  be a metric space and  $k \in \mathbb{R}, k > 0$ . Then  $\sigma'$  defined as  $\sigma'(x, y) = k\sigma(x, y)$  is a metric on  $M$ .*

*Proof.* We will split the proof according to the different metric axioms.

1. The non-negativity follows from the fact that multiplying by positive number does not change the sign.
2. The coincidence axiom results from that multiplying by positive number returns zero if and only if the multiplied value is zero.
3. The fact that multiplying by positive number does not change the symmetry of the function is enough to prove the symmetry.
4. The triangle inequality remains to be proved:  $\sigma'(x, y) = k\sigma(x, y) \leq k\sigma(x, z) + k\sigma(z, y) = \sigma'(x, z) + \sigma'(z, y)$ .

As we have proven every metric axiom we can conclude the proof.  $\square$

The rescaling would not work by zero or a negative number, as we would immediately break non-negativity or coincidence axiom. The metric is also closed under addition:

**Theorem 10.** *Let  $(M, \alpha), (M, \beta)$  be metric spaces. Let  $\sigma = \alpha + \beta$ , then  $(M, \sigma)$  is also a metric space.*

*Proof.* We will split the proof according to the different metric axioms.

1. The non-negativity axiom follows from the non-negativity of  $\alpha$  and  $\beta$ . As they are both non-negative, their sum must be also non-negative.
2. For the coincidence axiom it is enough to realize that both metrics are equal to zero if and only if  $x = y$ . Otherwise both metrics are positive.
3. The symmetry axiom follows from:

$$\begin{aligned}\sigma(x, y) &= \alpha(x, y) + \beta(x, y) = \\ &= \alpha(y, x) + \beta(y, x) = \sigma(y, x).\end{aligned}$$

4. The triangle inequality results from:

$$\begin{aligned}\sigma(x, y) &= \alpha(x, y) + \beta(x, y) \leq \alpha(x, z) + \beta(x, z) + \alpha(z, y) + \beta(z, y) = \\ &= \sigma(x, z) + \sigma(z, y).\end{aligned}$$

As we have proven every metric axiom, we can conclude the proof.  $\square$

The closure under addition together with the rescaling of the metric gives us the following corollary:

**Corollary 1.** *Let  $\{(M, \alpha_1), \dots, (M, \alpha_n)\}$  be a set of metric spaces and  $K = \{k_1, \dots, k_n\} \subset \mathbb{R}_{>0}^n$  be the set of weights. Let  $\sigma = \sum_{i=1}^n k_i \alpha_i$ , then  $(M, \sigma)$  is also a metric space.*

*Proof.* First we rescale each  $\alpha_i$  by  $k_i$ . This rescaling results into another metric by Theorem 9. Now we iteratively merge the rescaled metric using that the metric is closed under addition (Theorem 10).  $\square$

This enables us to construct a metric using weighted sum of other metrics. Note that in this case negative values of some  $k_i$  do not necessary break the metric if some metric would produce always higher values. However, for our purpose Corollary 1 is sufficient.

## 3.2 Distance Using Global Metadata

In this section, we discuss group of several algorithms to measure distance between two datasets using global attributes only. Given a distance measure between global metadata  $\sigma$ , an algorithm for measuring the distance between datasets  $\Delta$  using the  $\sigma$  can be naturally designed, as per Algorithm 8.

As `global_metafeatures` property returns fixed sized real-valued vector, we can intuitively use any metric on  $\mathbb{R}^n$ , and Algorithm 8 will produce a metric on the dataset space. Therefore, we can use any metric defined in this chapter including the weighted  $p$ -norms. This one is particularly interesting, as it allows for testing different settings of weights and see how this change affects our metalearning framework.

---

**Algorithm 8:** Distance  $\Delta$  using global metadata: *IDatasetDistance*

---

// Pseudocode for measuring dataset distance measure using  
global attributes.

**input** :  $\sigma \leftarrow$  Global metadata distance measure

**input** :  $x \leftarrow$  First dataset

**input** :  $y \leftarrow$  Second dataset

**output:** Distance between two datasets

1  $\text{global}_x \leftarrow x.\text{global\_metafeatures};$

2  $\text{global}_y \leftarrow y.\text{global\_metafeatures};$

3  $\text{distance} \leftarrow \sigma(\text{global}_x, \text{global}_y);$

4 **return** distance;

---

# Chapter 4

## Attribute Assignment

In Chapter 2, we defined the problem of metalearning, particularly the problem of algorithm selection and ranking. We have presented a general workflow that – given some distance measure between attributes and a new dataset – can rank algorithms based on previous experience. We also presented a distance measure based on the vector of global metadata. We have shown that this distance measure can be a metric on the space of datasets if the distance between the vector of global metafeatures is a metric. The room for improvement lies in the fact that for each dataset with arbitrary number of attributes, rows, and attribute domains, the number of global metadata extracted is always the same for each dataset. The ability to deal with such unstructured data has been recognized as a difficult and important task [16, 59].

The main contribution of this thesis lies in the proposal and analysis of algorithms that can handle non-propositional representation of datasets without losing information that can occur when extracting fixed amount information from the common structure of the datasets. We start by discussing the importance of dealing with unstructured data and by presenting unstructured domains and algorithms that are able to tackle associated issues. Namely, we will discuss vector embeddings on the space of strings together with kernel methods on arbitrary spaces. We will explain why such techniques cannot be applied directly to the space of datasets. We will review recent attempts of how to handle non-propositional representations in the metalearning domain. Finally, we propose concept of attribute assignment given some attribute distance measure. If the datasets have different number of attributes, dummy attributes are added to the dataset with less attributes. Dataset distance is computed based on sum of attribute distances given by the assignment. We will present several algorithms derived from this concept. The first supports only simple attribute distance measures but can be computed quickly. The second is based on the Hungarian algorithm and can handle arbitrary attribute distance measure. We will present some examples to give the reader a clear idea about the algorithms.

Then, theoretical properties of the proposed algorithms are discussed. It turns out that if the distance on the space of attributes extended by the dummy attribute is a metric then the resulting distance measure between datasets must also be a metric. We will also show that the same holds for the other direction. Yet we will discuss that the former direction is somewhat stronger if we only care about a metric on some subsets of dataset and attribute space. This can be

useful if we optimize metric properties on some training data. These theorems also mean that we can use attribute metrics based on  $p$ -norms and their weighted counterparts and we get a metric on dataset space as well. We will discuss conditions that are necessary for the resulting dataset distance to be a metric. We will especially discuss the addition of dummy attribute followed by the discussion on what is the best way how to add such dummy attributes into the attribute distance. We also show that if we normalized the resulting dataset distance by the number of attributes, we could break the metric properties.

## 4.1 Dealing with Unstructured Data

During last decades, many machine learning and data mining techniques emerged. Almost all of them were designed to handle propositional data, usually encoded as a vector of fixed size. However, not all data have this nice structure and there are lots of real world problems that are defined on unstructured data. *Natural language processing* (NLP) tasks are defined on the space of strings (words and texts) and many traditional methods are therefore not applicable. Another example of such unstructured space is the space of graphs. Many things such as social network connections can be naturally described using graphs. Therefore, it is important to investigate means of modifying machine learning tools to handle various non-propositional representations. In this section, we shall present two notable approaches to deal with unstructured data.

### 4.1.1 Word Embeddings

Word embeddings is the set of natural language processing techniques where words or phrases are mapped to the real-valued vectors. A well known example of word embedding is *word2vec* [84, 85]. Word2vec uses two different types of models to learn the vectors. The first one – *Continuous Bag-of-Word Model* or CBOW – tries to predict the target word given the surrounding words (so called context). The CBOW uses the probabilistic feedforward neural network [14] to estimate the probabilities of current word. The desired vector is the weight vector between the hidden layer and the neuron in the output neuron corresponding to the word in question. The *Skip-Gram Model* is the opposite of CBOW. The goal now is to guess the context given the word in the middle.

The interesting part is that the trained vectors capture many linguistic regularities [86]. For example the vector operation *CzechRepublic* – *Prague* should yield vector similar to the result of *Japan* – *Tokyo*. Similarly, the operation *King* – *Man* + *Woman* should resemble the vector of *Queen*. This is illustrated in Figure 4.1.

A similar algorithm for learning vectors is *GloVe* (Global Vectors for Word Representation) [90]. GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. The authors of [108] shows that GloVe and Skip-Gram model of word2vec, one explicitly factorizing a co-occurrence matrix and one implicitly factorizing a shifted-PMI matrix, are actually sharing similar objectives, though not completely the same.

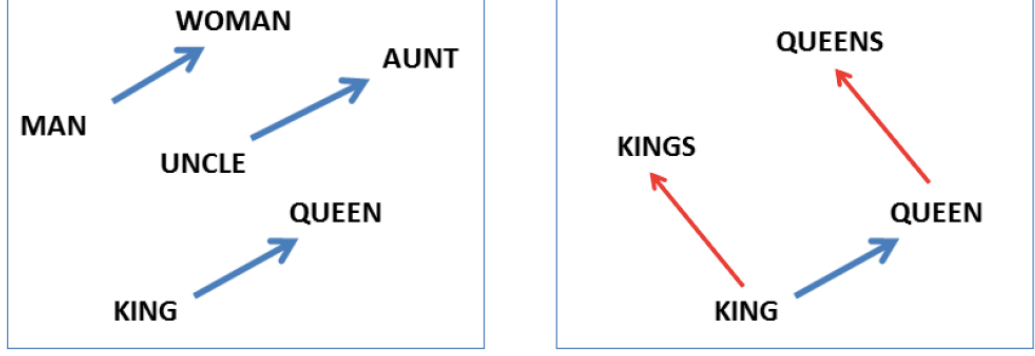


Figure 4.1: Linguistic regularities preserved by the word2vec algorithm [84, 85].

It is important to note that the word-embedding techniques are using the fact that the order of words in text is important. This is not applicable for the non-target attributes within the datasets as the order is not important. We could easily permute all the attributes and still argue that the information required to learn the features is still the same. In comparison, the order of words usually matters in the texts written by humans – which is one of the reasons why the vector embeddings show such nice results.

The key factor in the performance of word2vec and similar methods is the size of the corpus. Common choice is up to one billion of words [84].

There has also been ongoing research to use the vector embeddings on different domains than just natural languages. For example, *protein-vectors* (ProtVec) [9] uses word2vec ideas. The paper uses Skip gram neural network to build a dense distributed representation for biological sequences. The method was evaluated by classifying protein sequences obtained from *Swiss-Prot* [24] and outperformed existing classification methods. Furthermore, the method is applicable to other bioinformatics problem such as protein visualization, structure prediction, domain extraction, and interaction prediction.

### 4.1.2 Kernels

The kernel approach tackles the problem by looking at the similarities between objects. In general, kernel [49] is a function taking two objects and returning a measurement:

**Definition 20.** Let  $\Omega$  be a set. Then kernel  $K$  is a real-valued function of two variables from  $\Omega$ , i.e.,

$$\Omega \times \Omega \rightarrow \mathbb{R} \quad (4.1)$$

To get a similarity, only the so called dot-product kernels are usually considered:

**Definition 21.** Let  $\Omega$  be a set. Then dot-product kernel  $K$  is a real-valued function of two variables from  $\Omega$  satisfying

$$K(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathbb{V}}, \quad (4.2)$$

where  $\phi$  is a feature map  $X \rightarrow \mathbb{V}$  and  $\langle \cdot, \cdot \rangle_{\mathbb{V}}$  is an inner product on  $\mathbb{V}$ .

If  $\Omega$  is a space of real vectors, the dot product is a commonly considered inner product, and we can easily get so called linear kernel just by using the dot product:

$$K(x, y) = x^t y, \forall x, y \in \mathbb{R}^n. \quad (4.3)$$

It is possible to use  $\phi$  for rescaling and increase of dimensions. For instance, the polynomial kernel defined as

$$K(x, y) = (x^t y + b)^n, x, y \in \mathbb{R}^n, r > 0, n \in \mathbb{N}, n > 0. \quad (4.4)$$

computes the product in  $\binom{d+2}{2}$ -dimensional feature space where  $d$  is a dimension of input vectors and  $b$  is a parameter describing the influence of higher order terms versus lower order terms. For example, for  $n = 2$  the product is given by the (implicitly defined) mapping

$$\begin{aligned} \phi(\langle x_1, \dots, x_n \rangle) = \\ \langle x_1^2, \dots, x_n^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_{n-1}x_n, \sqrt{2b}x_1, \dots, \sqrt{2b}x_n, c \rangle. \end{aligned}$$

Other popular kernel is the so called *radial basis function* (RBF) kernel:

$$K(x, y) = e^{-\gamma \|x-y\|^2},$$

where  $\gamma$  is a parameter of the kernel.

What makes the kernel approach particularly interesting is that  $\Omega$  can be an arbitrary set, not just real numbers. If we can define a mapping  $\Omega$  to some space with inner product, we can have a kernel even for objects with variable structure.

For instance, strings over some alphabet have variable structure as they can be of arbitrary length. One of the kernels defined over strings is string subsequence kernel [78], which defined the similarity of two strings by the number of their common subsequences (not-necessarily contiguous):

**Definition 22.** Let  $\mathbb{A}$  be an alphabet and  $x, y \in \mathbb{A}$ . The string subsequence kernel is defined as:

$$K(x, y) = \sum_{u \in \mathbb{A}^n} \sum_{i: u=x[i]} \sum_{j: u=y[j]} \lambda^{l(i)+l(j)},$$

where  $\mathbb{A}^n$  is the set of all strings of length  $n$ ,  $s[i]$  is a subsequence of  $s$  given by some set of indices  $i$ ,  $\lambda$  is a decay factor and  $l(i)$  denote the total length of  $s[i]$  in  $s$  – the biggest index in  $i$  minus the smallest index in  $i$  plus one.

While this computation appears very expensive, recursive computation can be reduced to  $\mathcal{O}(n|x||y|)$  [78].

Another example of the string kernel is a spectrum kernel [75], which bases the similarity on common substrings of some predefined length.

**Definition 23.** Given a number  $k \geq 1$ , the  $k$ -spectrum of an input sequence is the set of all the  $k$ -length (contiguous) subsequences that it contains.

We defined a kernel with a feature map indexed by all possible subsequences  $a$  of length  $k$  from alphabet  $\mathbb{A}$ .

**Definition 24.** *K-spectrum kernel is defined as:*

$$K(x, y) = \langle (\phi_a(x))_{a \in \mathbb{A}^k}, \phi_a(y)_{a \in \mathbb{A}^k} \rangle,$$

where  $\phi_a(x)$  denotes number of times  $a$  occurs in  $x$ .

A very efficient method for computing spectral kernel is to build a suffix tree for the collection of  $k$ -length subsequences of  $x$  and  $y$ , obtained by moving a  $k$ -length sliding window across each of  $x$  and  $y$ . At each depth- $k$  leaf node of the suffix tree, store two counts, one representing the number of times a  $k$ -length subsequence of  $x$  ends at the leaf, the other representing a similar count for  $y$ . Note that this suffix tree has  $\mathcal{O}(kn)$  nodes. Using a linear time construction algorithm for the suffix tree [119], we can build and annotate the suffix tree in  $\mathcal{O}(kn)$  time. Now we calculate the kernel value by traversing the suffix tree and computing the sum of the products of the counts stored at the depth- $k$  nodes. The overall cost is thus  $\mathcal{O}(kn)$ .

We would like to mention another string kernel called local alignment as it directly influenced our work.

**Definition 25.** *An alignment (with gaps)  $\pi$  of  $p \geq 0$  positions between two sequences  $x, y$  is a pair of  $p$ -tuples:*

$$\pi = ((\pi_1(1), \dots, \pi_1(p), \pi_2(1), \dots, \pi_2(p))) \in \mathbb{N}^{2p}$$

that satisfies

$$1 \leq \pi_1(1) < \pi_1(2) < \dots < \pi_1(n) \leq |x|$$

$$1 \leq \pi_2(1) < \pi_2(2) < \dots < \pi_2(n) \leq |y|$$

We can score the alignments as follows:

**Definition 26.** *The local alignment score of an alignment  $\pi$  is equal to*

$$s_{S,g}(\pi) = \sum_{i=1}^{|\pi|} S(x_{\pi_1(i)}, y_{\pi_2(i)}) - \sum_{i=1}^{|\pi|-1} [g(\pi_1(i+1) - \pi_1(i)) + g(\pi_2(i+1) - \pi_2(i))],$$

where  $S$  is a substitution matrix encoding score for aligning letters with other letters and  $g$  is a gap penalty function.

The *Smith-Waterman* (SW) score is a local alignment score of the best alignment:

$$SW_{S,g} = \max_{\pi \in \Pi(x,y)} s_{S,g}(\pi).$$

The SW score can be calculated in  $\mathcal{O}(|x||y|)$  by dynamic programming with Smith-Waterman algorithm [115].

$$SW(i, j) = \max \begin{cases} 0, \\ SW(i-1, j) + S(x_i, y_j), \\ \max_k SW(i-k, j) - g(k), \\ \max_l SW(i, j-l) - g(l). \end{cases} \quad (4.5)$$

$SW(i, j)$  stands for the similarity of two segments ending in  $x_i$  and  $y_j$  respectively.

Unfortunately, SW score does not have to be a valid inner-product kernel [106]. However, a convolution kernel can be defined as follows:

$$K_{LA}^{(\beta)}(x, y) = \sum_{\pi \in \Pi(x, y)} e^{\beta s_{S, g}(\pi)}, \quad (4.6)$$

where  $\beta$  is a parameter. It can be shown that the SW score is approached by the limit:

$$\lim_{\beta \rightarrow \infty} \frac{1}{\beta} \ln(K_{LA}^{(\beta)}(x, y)) = SW(x, y). \quad (4.7)$$

Furthermore, it can be shown that if there is no gap penalty a SW score is a kernel [105].

Another domains where kernels were applied is the space of graphs [122] and images [46]. Again, we cannot apply string or graph kernels directly to the dataset space as the order of character or vertices matters.

Having an inner product kernel has many advantages. Computation of the inner product kernel can be carried out without explicitly using the  $\phi$  function and computing the inner product. Mapping to the feature space can be very expensive or not feasible at all. For example, the RBF kernel with  $\gamma = 1$  is actually computing the inner product in an infinite dimensional feature space:

$$e^{-\|x-y\|^2} = \sum_{j=0}^{\infty} \frac{(x^T y)^j}{j!} e^{(-\frac{1}{2}\|x\|^2)} e^{(-\frac{1}{2}\|y\|^2)}.$$

Such computation of inner product in the feature space using only values in the input space is sometimes being referred to as a *kernel trick*.

Many methods benefit from the kernel trick or can be extended to do so. For example, *support vector machine* (SVM) [26] can use the kernel to separate non-linearly separable sets by performing the separability in the feature space instead. The ability of kernels to be used in such a way is demonstrated in Figure 4.2 by using the scikit-learn library [89]. First, two categories of points are generated – blue points distributed over the circle with some small noise and red circles distributed around the circle with bigger radius. Clearly, the blue set is not linearly separable from the red set. Using another method enhanceable by the kernel trick – *kernelized PCA* with the RBF kernel [107] – we can achieve the linear separability.

The kernelized PCA opens many options as it can map arbitrary space with kernel to the vector of the desired length. Reducing the dimensions to two or three is especially useful as it can be visualized easily.

Metalearning is also used to aid the methods with kernels as choosing a right kernel can be crucial to solve the problem at hand. Authors of [8] use metalearning to select the right kernel for SVM by measuring the problem characteristics using classical, distance and distribution-based statistical information. In [7] the kernel is automatically selected based on knowledge transferred from the results on related data.

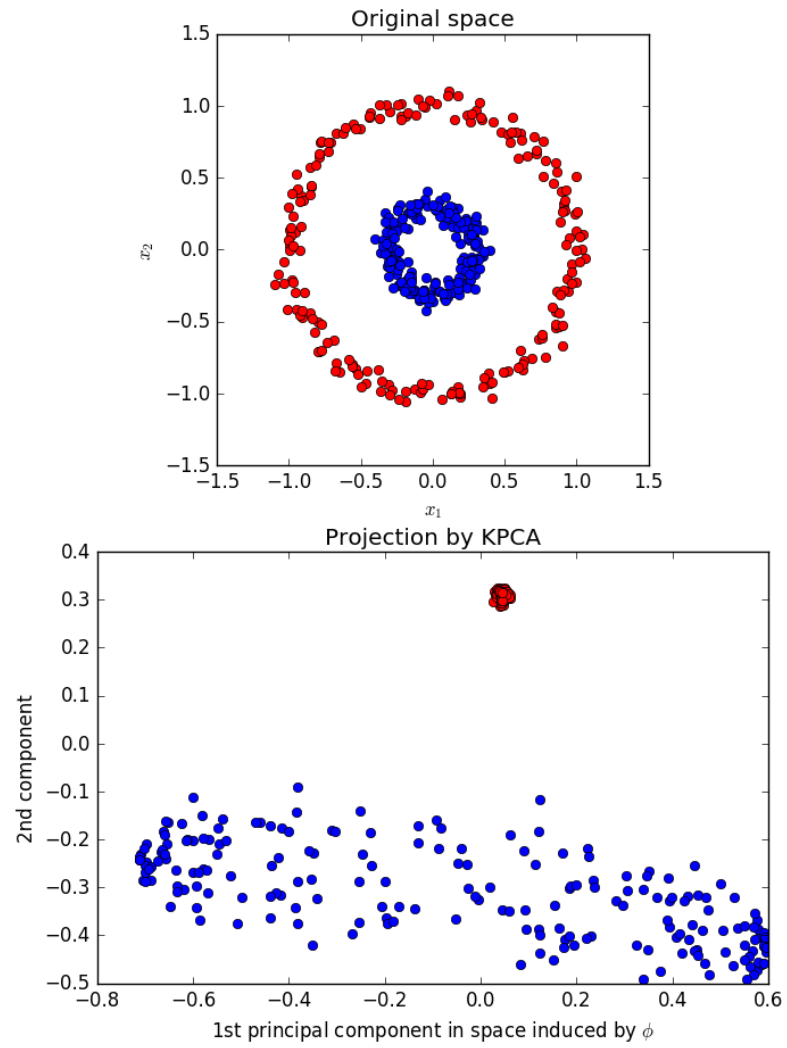


Figure 4.2: Example of using the kernelized PCA in the scikit-learn library [89] to make the data linearly separable.

## 4.2 Non-propositional Approach to Metalearning

All of the methods for algorithm selection problem discussed so far used the propositional approach – every dataset was represented by the metadata vector of fixed size. However, the problem is highly non-propositional as the number of attributes varies throughout the datasets. According to the authors of [16], the most common approach to deal with this problem is to do some form of aggregation, for example use mean skewness as an aggregation of every attribute with skewness. It is expected that important information will be lost in the process. In [61], we partially addressed the problem by adding the special attribute metadata. First, we identified the most significant attributes. The significance of attribute was identified by the *joint entropy* of a target class and attribute:

$$H(A, C) = - \sum_{v \in Val(A)} \frac{n(A(v))}{n} \sum_{t \in Val(C)} \frac{n_t(A(v))}{n(A(v))} \log_2 \frac{n_t(A(v))}{n(A(v))}, \quad (4.8)$$

where  $n$  is the number of observations,  $n(A(v))$  is the number of observation with the value  $v$  of the attribute and  $n_t(A(v))$  is the number of observations with the value  $v$  of the attribute and with the value  $t$  of the target attribute. For the continuous attributes (for them the direct entropy computation is not desirable) the discretization of the values was used instead. We then chose a number  $k$  and the attribute metadata of  $k$  most significant attributes were added to the propositional representation of the datasets. We specifically used entropy (see Equation 2.5) of the  $i$ -th most significant attribute. It is however easy to extend the approach with the arbitrary attribute metadata.

Alternatively, in [60] two measures of association are used, one for continuous attributes and one for discrete ones. For continuous attributes, the absolute value of the correlation coefficient is used, for discrete attributes, the *Goodman and Kruskal's tau* [6] coefficient also known as the concentration coefficient is used. The problem of the various number of attributes is dealt with by introducing a histogram of these attributes with the fixed number of bins.

Both the approaches above are merely pushing the problem further. As we have chosen fixed number of most significant attributes or bins, what happens if we have to deal with datasets with billion times more attributes than the number of bins?

Kalousis et al. [59] solve the problem of varying attributes by defining a distance measure on the attribute space. The dataset distance is then similar to the distances used in clustering – either *Single Linkage Based Similarity* (similarity taken is the maximum similarity between the attributes) or *Average Linkage Based Similarity* is used. Attribute similarity is calculated as 1 minus the Manhattan distance defined on the attribute metadata. This can handle the non-propositional approach and the authors demonstrated that such distance function is useful in metalearning. However, we can still imagine datasets that are very different but whose Single Linkage Based Similarity is high (one common attribute, many very different attributes) and similar datasets whose Average Linkage Based Similarity is low (many similar attributes but distant from others). A possible explanation is that the Single Linkage Based Distance does not necessarily produce a metric:

**Observation 1.** *Given arbitrary metric  $\delta$  on the attribute space whose cardinality is at least two there exist datasets  $a, b, c$  such that the distance between datasets  $\Delta$  induced by Single Linkage Clustering*

$$\Delta(a, b) = \min_{i, e_i \in a, j, e_j \in b} (\delta(e_i, e_j))$$

*is not a metric on the dataset space.*

*Proof.* The attribute space has at least two distinct elements, we will call them  $e_1, e_2$ . Let  $a = \{e_1\}$ ,  $b = \{e_2\}$  and  $c = \{e_1, e_2\}$ . Then

$$\Delta(a, c) + \Delta(b, c) = 0 + 0 < \Delta(a, b),$$

which breaks the triangle inequality.  $\square$

We can also find the similar counterexample for the Average Linkage Based Distance:

**Observation 2.** *Given arbitrary metric  $\delta$  on the attribute space whose cardinality is at least two there exist datasets  $a, b$  such that the distance between datasets  $\Delta$  induced by the Average Linkage Clustering*

$$\Delta(a, b) = \frac{1}{|a||b|} \sum_{i, e_i \in a} \sum_{j, e_j \in b} \delta(e_i, e_j)$$

*is not a metric on the dataset space.*

*Proof.* The attribute space has at least two distinct elements, we will call them  $e_1, e_2$ . Let  $a = b = \{e_1, e_2\}$ . Then

$$\delta(a, b) = \frac{1}{4}(0 + \delta(e_1, e_2) + 0 + \delta(e_2, e_1)) = \frac{1}{2}\delta(e_1, e_2) > 0,$$

which breaks the coincidence axiom.  $\square$

## 4.3 Distance Using Attributes

As discussed in the beginning of this chapter, ability to process unstructured data can significantly improve the tasks defined on those data as we can use bigger palette of approaches. Unfortunately, the methods proposed on the strings and graph spaces are not out of the box applicable to the space of datasets but rather can serve as an inspiration. For example, the Smith-Waterman algorithm looks interesting, but it was designed with the fact that the order of letter matters in mind. With the datasets, it should not matter if we permute the attribute. The target function should be learnable all the same as we can preprocess the attributes by inverse permutation. The non-propositional approaches reviewed in the previous section are either losing information or lacking properties usually important for the distance functions.

Authors of [125] review possible ways of using the distance measures defined on  $\mathbb{X}$  in order to define distance measures on the power set  $2^{\mathbb{X}}$  of  $\mathbb{X}$ . The most promising is the distance measure defined as follows:

$$D_M(A, B) = \min_{R_i \in R} \left( \sum_{(a_k, b_j) \in R_i} \delta(a_k, b_j) + (|B \setminus R_i(A)| + |A \setminus R_i^{-1}(B)|) \frac{M}{2} \right), \quad (4.9)$$

where  $\delta$  is a distance between elements of some space  $\mathbb{S}$ ,  $A, B \subseteq \mathbb{S}$ ,  $R_i$  is a matching (each element at most once) on  $A \times B$ , and  $M$  is a maximum distance of  $\delta$  on  $\mathbb{S}$ .  $D_M$  can be also viewed as a distance that finds optimal mapping between elements of  $A, B$  and can decide to omit some elements by getting  $\frac{M}{2}$  penalty. Authors of [100] prove that the  $D_M$  is a metric if  $\delta$  is a metric. Furthermore,  $D_M$  is computable in the polynomial time. We can treat datasets as composed by attributes and use  $D_M$  by defining a distance  $\delta$  on attribute space. However, there are still reasons that render this approach impractical for metalearning. The distance measure  $D_M$  requires  $M$ . The computation of  $M$  may not be feasible or not known in advance, especially if the attribute distance was not normalized. Additionally, some attributes may be more significant than the others. Not matching some insignificant attributes because of fewer attributes in the second dataset (for example with low entropy for predicting the target attribute) will still result in  $\frac{M}{2}$  distance. Also, algorithm can decide not to match some attributes with high difference and get only  $\frac{M}{2}$  penalty. For example, when comparing two datasets with single attributes, the results will be the same if the attributes are really distant ( $M$ ) or somewhat distant ( $\frac{M}{2}$ ). In both cases the  $D_M$  will return  $\frac{M}{2}$ .

In this section, we propose several approaches that can handle the non-propositional dataset representations without any trade-offs. We have already published their descriptions and experiments validating their asset in [112, 113, 110, 114, 111]. Each one is based on the idea of attribute aligning where the order of attributes is not important. By supplying a function measuring the distance between individual attributes (like in [59]), we could try to align the attributes in the way that minimizes the sum of distances between aligned attributes. To avoid confusion, we will always denote attribute distance measure as  $\delta$ . The name was not chosen randomly. The upper case  $\Delta$  – as always – will denote the final dataset distance measure. As we will see, we will piece  $\Delta$  out of smaller  $\delta$ s, hence the name.

Although we are aiming to handle the non-propositional approach, we will start with propositional situation to describe the approach, and extend it later to handle varying amount of attributes. For now let us suppose that there is the same number of attributes in every dataset (and hence the same number of attribute metafeatures).

**Definition 27.** *Given a distance function between attributes  $\delta$ , two datasets  $a, b$  and a bijection  $f$  between the attributes of  $a, b$  we define the dataset distance induced by the bijection between the datasets as:*

$$\Delta_f(a, b) = \sum_{k=1}^n (\delta(a_k, f(a_k))), \quad (4.10)$$

where  $a_k$  is the  $k$ -th attribute of  $a$  and  $f(a_k)$  corresponding attribute in dataset  $b$  given by the bijection  $f$ . We will sometimes refer to the  $\Delta_f$  as the cost of  $f$ .

We would like to match the attributes as best as possible to get the lowest possible distance  $\Delta_f$ , so optimally we are looking for  $f^*$ :

$$f^* = \operatorname{argmin}_f (\Delta_f). \quad (4.11)$$

We will denote  $f^*$  as an optimal alignment.

From this, the general distance between datasets can be derived:

$$\Delta(a, b) = \Delta_{f^*}(a, b). \quad (4.12)$$

Now suppose that the number of attributes is different. We transform this case to the previous one by adding dummy attributes into the dataset with less attributes. We can think of a dummy attribute in a similar way as of a gap penalty in the Smith-Waterman algorithm. There are two approaches how to do this. Suppose  $\mathbb{A}$  is a space of attributes. Either  $dummy \in \mathbb{A}$  or  $dummy \notin \mathbb{A}$ . In the former case, nothing is needed, as the distance function between attributes is already defined if the *dummy* attribute is on the input. In the latter case, it is needed to extend the distance function by defining the distance between a regular attribute and the *dummy* one. In the further text, we will refer to the attribute space with the *dummy* attribute as to the *extended attribute space*. If it is clear from the context, we will sometimes refer to extended attribute space simply as attribute space. *Dummy* attribute will be referred to as *dummy* or specifically to  $dummy_a$  if the dummy attribute is already in the attribute space and  $dummy_n$  if the attribute is newly created. We will discuss the pros and cons of these approaches as well as how to choose the right attribute for the *dummy* one, and how to define the distance between a regular attribute and a *dummy* one respectively in Section 4.5.

From now on, we can suppose that if we are aligning two datasets that they have the same number of attributes, and that one dataset was to some extent enriched with some number of *dummy* attributes, so the amount of attributes matches.

There are  $n!$  bijections from  $a$  to  $b$ . It is costly to enumerate them, so it is desired to eliminate some of the bijections in advance. In this work, we came up with two approaches that vary in the generality of the attribute distance and computational complexity.

**Definition 28.** Evaluation function  $\sigma$  is a function mapping attributes to  $\mathbb{R}$ .

In our first approach, we suppose we have  $\sigma$  (for example, number of distinct values), and the attribute distance is defined as follows:

$$\delta(a_k, f(a_k)) = |\sigma(a_k) - \sigma(f(a_k))|. \quad (4.13)$$

**Theorem 11.** Given datasets  $a, b$  and a bijection  $f$ , if we sort  $a$  and  $b$  by  $\sigma$  in the ascending order obtaining  $a', b'$  we can find  $f'$  such that

$$\Delta_{f'}(a', b') = \Delta_f(a, b). \quad (4.14)$$

*Proof.* Let  $\pi_a, \pi_b$  be the permutations used to sort  $a, b$ . Define  $f'$  as follows:

$$f'(a'_i) = \pi_b(f(\pi_a^{-1}(a'_i))). \quad (4.15)$$

We have to prove that for every  $k$  there is  $j$  such that  $|\sigma(a_k) - \sigma(f(a_k))| = |\sigma(a'_j) - \sigma(f'(a'_j))|$ . As a candidate for  $j$  we take such  $j$  that  $a'_j = \pi_a(a_k)$ . Such  $j$  exists and it is unique as  $\pi_a$  is a permutation. Observe that sorting permutation does

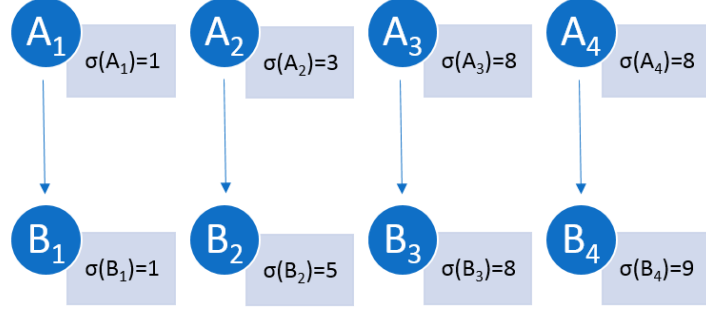


Figure 4.3: Example of the identity alignment. If the attributes are sorted by  $\sigma$ , each attribute is aligned to the attribute with the same order.

not affect  $\sigma$ . Therefore  $\sigma(a'_j) = \sigma(\pi_a(a_k)) = \sigma(a_k)$ . Directly from the observation the following equation can be derived:

$$\sigma(f'(a'_j)) = \sigma(\pi_b(f(\pi_a^{-1}(a'_j)))) = \sigma(f(\pi_a^{-1}(a'_j))) = \sigma(f(a_k)), \quad (4.16)$$

which concludes the proof as the  $j$  is unique and it always exists.  $\square$

**Corollary 2.** *We can exclusively use this canonical representation and without the loss of generality, suppose that  $a, b$  are sorted by  $\sigma$ .*

**Definition 29.** *We say that the bijection  $f$  is the identity alignment if  $\forall i, a_i \in a : f(a_i) = b_i$ .*

The example of the identity alignment is shown in Figure 4.3.

**Theorem 12.** *Identity alignment is optimal.*

*Proof.* Given the bijection  $f$  that is not an identity alignment and is optimal, we will show that it can be either transformed to identity alignment or it is not optimal, which leads to a contradiction. Find the lowest  $i$  such that  $f(a_i) \neq b_i$ . Such  $i$  exists, because  $f$  is not an identity alignment. We mark the index of attribute  $f(a_i)$  as  $z$ . Because  $f$  is a bijection, we can find  $j$  such that  $f^{-1}(b_i) = a_j$ . Take a bijection  $f'$  that is the same as  $f$  with the exception of arguments  $i$  and  $z$ :

$$\begin{aligned} f'(a_i) &= b_i, \\ f'(a_j) &= b_z. \end{aligned} \quad (4.17)$$

In other words,  $f'$  is more similar to identity alignment than  $f$ . The whole transformation is shown in Figure 4.4.

We have to verify that:

$$d_f \geq d_{f'}. \quad (4.18)$$

$$\sum_{k=1}^n (|\sigma(a_k) - \sigma(f(a_k))|) \geq \sum_{k=1}^n (|\sigma(a_k) - \sigma(f'(a_k))|). \quad (4.19)$$

The  $d_f$  and  $d_{f'}$  differs only in positions  $i$  and  $j$ . Thus we can simplify the equation to:

$$\sum_{k \in i, j} (|\sigma(a_k) - \sigma(f(a_k))|) \geq \sum_{k \in i, j} (|\sigma(a_k) - \sigma(f'(a_k))|). \quad (4.20)$$

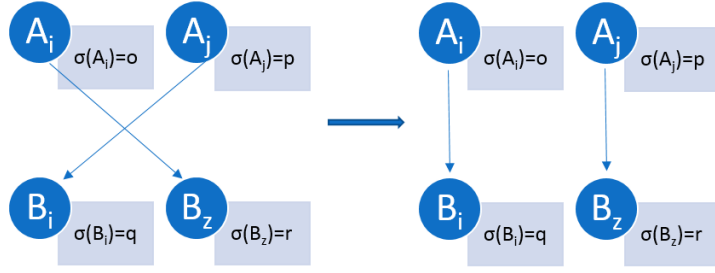


Figure 4.4: Example of transformation. Find attributes that falsify identity alignment and switch them. At least one less pair of attributes now falsifies the identity alignment.

$$|\sigma(a_i) - \sigma(b_z)| + |\sigma(a_j) - \sigma(b_i)| \geq |\sigma(a_i) - \sigma(b_i)| + |\sigma(a_j) - \sigma(b_z)|. \quad (4.21)$$

If we denote values  $\sigma(a_i), \sigma(a_j), \sigma(b_i), \sigma(b_z)$  as  $o, p, q, r$ , we can simplify the equation further:

$$|o - r| + |p - q| \geq |o - q| + |p - r|. \quad (4.22)$$

Because we use sorted canonical representation and because of the fact we have taken the lowest  $i$  possible we know that:

$$o \leq p, \quad (4.23)$$

$$q \leq r. \quad (4.24)$$

There are 6 possible orderings of  $o, p, q, r$  that fulfils these two equations. The enumeration of these 6 cases is in Table 4.1.

Table 4.1: Equation 4.22 holds for every possible case.

ORDER	$ r - o  +  q - p $	$ q - o  +  r - p $	$\geq$
opqr	$ q - r  + 2 *  q - p  +  p - o $	$ q - r  + 2 *  q - p  +  p - o $	$=$
oqpr	$ r - p  + 2 *  q - p  +  q - o $	$ q - o  +  r - p $	$\geq$
oqrp	$2 *  q - r  +  q - o  +  r - p $	$ q - o  +  r - p $	$\geq$
qopr	$ r - p  +  q - o  + 2 *  p - o $	$ q - o  +  r - p $	$\geq$
qorp	$2 *  r - o  +  q - o  +  r - p $	$ q - o  +  r - p $	$\geq$
qrop	$2 *  r - o  +  q - r  +  p - o $	$2 *  r - o  +  q - r  +  p - o $	$=$

In every presented case, the  $f'$  is at least as good as  $f$ , and the position  $i$  no longer falsifies the identity alignment. If  $f'$  is still not the identity alignment, we can repeat the previous step until we get an identity alignment because the number of attributes is finite.  $\square$

**Corollary 3.** *Based on Theorem 12, a more efficient algorithm for the attribute alignment can be derived with the complexity of alignment equal to  $\mathcal{O}(n \log(n))$ , where  $n$  is the number of attributes.*

The pseudocode is outlined in Algorithm 9.

From the algorithm, the total computational complexity can be inferred. The sorting can be done in  $\mathcal{O}(n \log(n))$ , where  $n$  is the number of attributes. The enumeration can be done in  $\mathcal{O}(n)$ , the evaluation of  $\sigma$  for every attribute takes

---

**Algorithm 9:** Attribute alignment

---

// Pseudocode for an attribute alignment algorithm with  
constrained attribute distance function running in  
 $\mathcal{O}(n \log(n))$ .

**input** :  $a \leftarrow$  List of attributes

**input** :  $b \leftarrow$  List of attributes

**input** :  $\sigma \leftarrow$  Attribute evaluation function:  $\mathbb{A} \rightarrow \mathbb{R}$

**output**: Distance between  $a$  and  $b$

```
1 Add dummy attributes into the list with less attributes;
2 Sort both list of attributes by  $\sigma$ ;
3 totalDistance  $\leftarrow$  0;
4 for  $i \leftarrow 0$  to  $i < \text{len}(a)$  do
5   | totalDistance  $\leftarrow |\sigma(a[i]) - \sigma(b[i])|$ ;
6 end
7 return totalDistance;
```

---

$\mathcal{O}(nc(\sigma))$  steps, where  $c(\sigma)$  is a cost of calling the evaluation function for a single attribute. Therefore, the total complexity of Algorithm 9 is  $\mathcal{O}(n(\log(n) + c(\sigma)))$ .

In our second approach, we allow arbitrary function  $\delta$  measuring the distance between two attributes. Given two datasets  $a, b$  and attribute distance function  $\delta$ , a distance matrix  $M$  can be easily built up:

$$M_{i,j} = \delta(a_i, b_j). \quad (4.25)$$

We can see the distance matrix  $M$  as the cost function and the aligning of attributes as an assignment, which leads to an assignment problem. The assignment problem is well known. The polynomial algorithm – in the graph theory known as the Hungarian algorithm – solving the assignment problem in  $\mathcal{O}(n^4)$  was described in [70]. The  $\mathcal{O}(n^3)$  implementation of the Hungarian algorithm was later published in [36]. Before the algorithm itself we will state few definitions needed. Given a graph  $G = (V, E)$ :

**Definition 30.** A matching is a subset  $M \subseteq E$  such that  $\forall v \in V$  at most one edge in  $M$  is incident upon  $V$ .

**Definition 31.** A perfect matching is an  $M$  in which every vertex is adjacent to some edge in  $M$ .

**Definition 32.** A vertex labelling is a function  $l : V \mapsto \mathbb{R}$ .

**Definition 33.** Vertex  $v$  is matched if it is an endpoint of edge in  $M$ , otherwise it is free.

**Definition 34.** The equality graph with the respect to the labelling  $l$  is  $G = (V, E_l)$ , where  $E_l = \{(x, y) | x, y \in V, l(x) + l(y) = w(x, y)\}$ .

**Definition 35.** A path is alternating if its edges alternate between  $M$  and  $E \setminus M$ .

**Definition 36.** An alternating path is augmenting if both endpoints are free.

**Definition 37.** Define neighbour of  $u \in V$  and set  $S \subseteq V$  to be:

$$N_l(u) = v : (u, v) \in E_l, N_l(S) = \cup_{u \in S} N_l(u). \quad (4.26)$$

Given the definitions above, the pseudocode for the Hungarian algorithm can be outlined, as per Algorithm 10.

---

**Algorithm 10:** Hungarian algorithm

---

```

// Pseudocode of the Hungarian Algorithm solving assignment
  problem in  $\mathcal{O}(n^3)$ .
input :  $(X \cup Y, E) \leftarrow$  Weighted bipartite graph
output: Minimal assignment

1 Generate initial labelling  $l$  and matching  $M$  in  $E_l$ . ;
2 while  $M$  is not perfect do
3    $u \leftarrow$  pick free vertex  $\in X$ ;
4    $S \leftarrow \{u\}$ ;
5    $T \leftarrow \{\}$ ;
6   if  $N_l(S) = T$  then
7     update labels (forcing  $N_l(S) \neq T$ ) ;
8      $a_l \leftarrow \min_{s \in S, y \notin T} (l(x) + l(y) - w(x, y))$ ;
9      $l'(v) \leftarrow \begin{cases} l(v) - a_l; & \text{if } v \in S, \\ l(v) + a_l; & \text{if } v \in T, \\ l(v); & \text{otherwise.} \end{cases}$ 
10  end
11  if  $N_l(S) \neq T$  then
12    pick  $y \in N_l(S) - T$ ;
13    If  $y$  free,  $u - y$  is augmenting path. Augment  $M$  and go to 2;
14    If  $y$  matched, say to  $z$ , extend alternating tree:
15     $S \leftarrow S \cup z, T \leftarrow T \cup y$ . Go to 6;
16  end
17 end
18 return minimalAssignment;

```

---

We can use this algorithm to find the best assignment of the attributes. From the assignment the total distance between attributes can be computed as the sum of individual distances defined by the alignment as already seen in Algorithm 9. If  $\delta$  was defined in the same manner as in the first approach, we would get the same result (Algorithm 9 is a special case of Algorithm 12). The difference is that this version allows for an arbitrary function measuring distance between attributes as an input.

For the sake of generalization, we will define *IAttributeDistance* interface that will represent such attribute distance. This interface is outlined in Algorithm 11. The total complexity of the algorithm depends on the complexity of this function and is  $\mathcal{O}(n^3 + n^2\sigma(n))$ , where  $n$  is the number of attributes and  $\sigma(n)$  is the complexity of the attribute distance function. The whole algorithm that uses the Hungarian Algorithm and *IAttributeDistance* is outlined in Algorithm 12.

It is arguable whether we should come with the distance function between attributes that covers all cases including the distance between categorical and

---

**Algorithm 11:** *IAttributeDistance*: Dataset distance interface

---

```
// Interface for measuring distance between two attributes.  
input :  $a \leftarrow$  First attribute  
input :  $b \leftarrow$  Second attribute  
output:  $d \in \mathbb{R}$ ,  $d$  is an attribute distance  $\delta$  between  $a, b$ .
```

---

---

**Algorithm 12:** Attribute assignment

---

```
// Pseudocode for an attribute alignment algorithm using  
    Hungarian Algorithm for the alignment.  
input :  $a \leftarrow$  First list of attributes  
input :  $b \leftarrow$  Second list of attributes  
input :  $\delta \leftarrow$  IAttributeDistance Function  
output: Distance between  $a$  and  $b$   
  
1 Add dummy attributes into the list with less attributes;  
2  $M[i, j] \leftarrow \delta(a[i], b[j]);$   
3  $\text{assignments} \leftarrow \text{HungarianAlgorithm}(M);$   
4  $\text{totalDistance} \leftarrow 0;$   
5 for  $i \leftarrow 0$  to  $i < \text{len}(a)$  do  
6 |  $\text{totalDistance} \leftarrow M[i][\text{assignments}[i]];$   
7 end  
8 return  $\text{totalDistance};$ 
```

---

numerical attribute. We suppose that the attribute metafeatures of categorical and numerical attributes will vary, thus making it difficult to specify the distance. To solve this problem, the distance could be split into two parts: the distance between numerical attributes and the distance between categorical attributes. The final distance would be then the total of the sub-distances. To generalize this idea, we could go a step further and define selectors over the list of attributes. The selector would be a function accepting a list of attributes and returning a subset of the list. The selector interface is described in the *ISelectorInterface* (Algorithm 13). Specific examples conforming to this *ISelectorInterface* are Algorithms 14 (numerical selector) and 15 (categorical selector). An attribute distance function could then be provided for each selector. Optionally, weights could be defined for each selector describing the importance of such selector (e.g. categorical attributes could be weighted more than numerical). This is outlined in Algorithm 16. To illustrate how to invoke this algorithm for the example above (sum of distances of categorical and numerical attributes), we would set the selectors to just defined ones:

$[NumericalAttributesSelector, CategoricalAttributesSelector].$

The attribute distance between categorical attributes and numerical attributes would be needed to invoke the whole algorithm. Also, note that Algorithm 16 is a generalization of Algorithm 12. To obtain the equal results we have to invoke Algorithm 16 with the single distance, selector that filters nothing and weight 1. This will allow us to reuse some theorems that are valid for Algorithm 10. We can also use the idea with selectors for the Attribute Alignment algorithm

(Algorithm 9). We will call the modified Attribute Alignment with selectors as Combined Attribute Alignment. However, we will not explicitly provide the outline of this algorithm as it is just Algorithm 16 with the Attribute Alignment algorithms instead of Attribute Assignment.

The last thing remaining is to make the assignments algorithms compatible with the *IDatasetDistance*. The first accepts two lists of attributes, the latter two datasets. As the transformation of dataset is trivial and more of a technicality we will treat the dataset to be implicitly convertible to the list of attributes (but not vice-versa) by just extracting all the attributes out of a dataset. This will make all assignment type algorithms compatible with *IDatasetDistance* interface.

---

**Algorithm 13:** *ISelectorInterface*: Interface for selecting subset of attributes.

---

// Interface for selecting subset of attributes.

**input** :  $a \leftarrow$  List of attributes

**output:**  $a', a' \subseteq a$

---



---

**Algorithm 14:** *NumericalAttributesSelector*: *ISelectorInterface* for selecting numerical attributes

---

// Selector for selecting numerical attributes.

**input** :  $a \leftarrow$  List of attributes

**output:**  $a', a' \subseteq a$ .

1 **return**  $a.\text{where}(x : x \text{ is Numerical});$

---



---

**Algorithm 15:** *CategoricalAttributesSelector*: *ISelectorInterface* for selecting numerical attributes

---

// Selector for selecting categorical attributes.

**input** :  $a \leftarrow$  List of attributes

**output:**  $a', a' \subseteq a$ .

1 **return**  $a.\text{where}(x : x \text{ is Categorical});$

---

## 4.4 Examples

In this section, we will demonstrate the use of our algorithms. Let us start with Algorithm 9. Suppose that we have two datasets  $a$  and  $b$ . The possible values of individual attributes are shown in Tables 4.2 and 4.3.

Let us define  $\sigma$  as the number of categories in each attribute. We will extend the attribute space using *dummy<sub>n</sub>*. To allow that, we need to define distance between a regular attribute and *dummy<sub>n</sub>*. It suffices to define  $\sigma$  of *dummy<sub>n</sub>* attribute. In our case we will set this value to 0. Add one *dummy<sub>n</sub>* attribute to the dataset  $b$ . Now the number of attributes is the same. Sort both datasets by

---

**Algorithm 16:** Combined Attribute Assignment

---

```
// Pseudocode for distance measure combining multiple
// attribute assignments for each selectors.
input :  $a \leftarrow$  First list of attributes
input :  $b \leftarrow$  Second list of attributes
input : selectors  $\leftarrow$  List of selectors
input :  $w \leftarrow$  List of weights
input : distanceMeasures  $\leftarrow$  List of IAtributeDistance functions
output: Distance between  $a$  and  $b$ 

1 totalDistance  $\leftarrow$  0;
2 for  $i \leftarrow 0$  to  $i < \text{len}(\text{selectors})$  do
3    $a' \leftarrow \text{selectors}[i](a)$ ;
4    $b' \leftarrow \text{selectors}[i](b)$ ;
5    $\delta \leftarrow \text{distanceMeasures}[i]$ ;
6   Add dummy attributes into  $a'$  or  $b'$  - whichever has less attributes;
7   totalDistance  $\leftarrow w[i] * \text{attributeAssignment}(a', b', \delta)$ ;
8 end
9 return totalDistance;
```

---

Table 4.2: Possible values of dataset  $a$ .

<i>Att1</i>	<i>Att2</i>	<i>Att3</i>
Blue	Small	Common
White	Medium	Rare
Red	Huge	
Pink		

Table 4.3: Possible values of dataset  $b$ .

<i>Att1</i>	<i>Att2</i>
Wool	Slow
Cotton	Fast
Straw	Faster than light
Bamboo	
Seaweed	

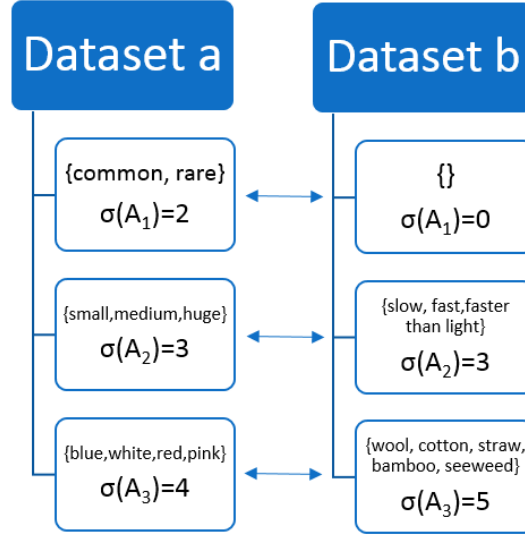


Figure 4.5: Method one - attributes of datasets  $a$  and  $b$  were aligned according to their  $\sigma$ .

$\sigma$ . The results are shown in Figure 4.5. Now enumerate every aligned pair and sum up the differences between sigma values of each pair. The total distance is  $|2 - 0| + |3 - 3| + |4 - 5| = 2 + 0 + 1 = 3$ .

We will use the same distance measure and the same datasets to demonstrate Algorithm 10. Let us define *AttributeDistance* function by building a distance matrix. The values of the distance matrix are in Table 4.4.

Table 4.4: Distance matrix of the attributes of datasets  $a$  and  $b$ . The matrix will serve as an input of the Hungarian algorithm.

	$a - Att1$	$a - Att2$	$a - Att3$
$b - Att1$	1	2	3
$b - Att2$	1	0	1
$b - Att3$ ( <i>dummy</i> )	4	3	2

By applying the Hungarian algorithm, we obtain the optimal alignment. The optimal alignment can be found in Table 4.5.

Algorithm 16 adds selectors to the process. Imagine we have dataset  $c$  defined in Table 4.6. Suppose we have two selectors – one for numerical and one for categorical attributes. New datasets emerge from the input dataset according to the selectors. In the case of dataset  $c$  processed by the selectors, we will get two datasets. The first consisting solely of  $Att1$  for the numerical selector and

Table 4.5: Results of application of the Hungarian algorithm. The optimal alignment is coloured.

	$a - Att1$	$a - Att2$	$a - Att3$
$b - Att1$	1	2	3
$b - Att2$	1	0	1
$b - Att3$ ( <i>dummy</i> )	4	3	2

the second consisting solely of *Att2* for the categorical selector. This gives us a new assignment task for every selector. Such tasks are processed by attribute assignment algorithm (Algorithm 10) demonstrated above. The algorithm finishes by weighting the inputs for each assignment result according to the given weights.

Table 4.6: Possible values of dataset *c*.

<i>Att1</i>	<i>Att2</i>
1	Good
4.5	Better
3.7	Best
5	
2	

### Complexity Concerns

The complexity of the assignment may be of a concern – even though Hungarian algorithm is polynomial, its power can be too high. We do not expect this to be the case very often, since the model training usually takes quite a long time (for example, the problem of training the Neural Network with 3 hidden neurons is an NP-Complete problem [15]). Polynomial complexity for the recommendation is still just a negligible part of the whole process that can significantly reduce time of the training phase. Still, we would like to address the issue in case the complexity would be of concern. The complexities above are related to quality assessment of some algorithm. This is relevant when training new models, as the model assessment is part of the training. In the case of recommendation system being in production and new dataset arrives, the dataset distance is not needed for every pair of datasets but rather for the input and every other dataset. In case of the attribute assignment, the main burden lies in the Hungarian Algorithm. This can be mimicked by using approximate algorithms to solve the assignment problem:

**Definition 38.** Let  $w(M)$  denote the weight of a matching in  $G$ , and  $M^*$  a minimum-cost perfect matching in  $G$ . We call a perfect matching  $M$   $c$ -approximate, for  $c \geq 1$ , if  $w(M) \leq cw(M^*)$ .

In case the underlying matrix for assignment is a metric, we can use results of [5]. For any  $\sigma > 0$ , the authors present an algorithm that, in

$$\mathcal{O}(n^{2+\sigma} \log n \log^2(1/\sigma))$$

time, computes a  $\mathcal{O}(1/\sigma^\alpha)$ -approximate matching of  $G$ , where  $\alpha = \log_3 2 \approx 0.631$ .

## 4.5 Theoretical Properties

In this section, we will explore interesting properties of the algorithms proposed in the previous section. We will show how the Assignments algorithm preserves metric properties, whether the same holds in opposite directions, and also we will

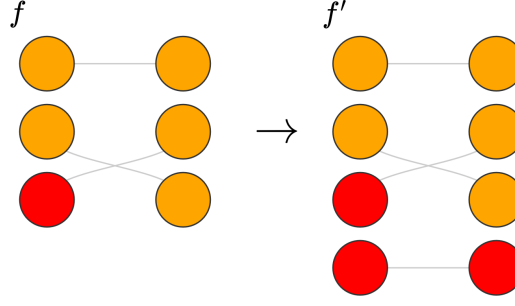


Figure 4.6: Extending optimal assignment by row of dummy attributes without changing the cost.

be arguing about the best way of extending an attribute space with a *dummy* attribute.

Before we proceed to exploring the theoretical properties, we have to address a technicality. Consider the following situation. If the  $dummy_a$  is taken directly from the attribute space, it may theoretically happen that  $dummy_a$  will be compared to the equal attribute that was not dummy. However, to have at least a hope of obtaining a metric, we have to consider the case where the dataset with some attributes equal to  $dummy_a$  will be compared to almost the same dataset except that it will be missing the  $dummy_a$  attributes. The metric would require the distance between these datasets to be a non-zero. We have two ways to overcome this. We can also consider datasets equal if their attribute metadata are equal except any number of  $dummy_a$  like attributes. Another possibility is to modify the algorithm to distinguish between an artificial  $dummy_a$  attribute and equal regular one and output some small non-negative number  $\epsilon$  instead. At the same time, we would return  $\epsilon + \delta(x, y)$  for non-matching input. In here we will use the former approach, as it will not clutter the text. However, we do not expect that this situation will happen often (or happen at all) as the attribute space will usually be of infinite size.

**Theorem 13.** *Let  $a, b$  be lists of attributes,  $f$  optimal alignment between  $a$  and  $b$ . Attribute distance measure  $\delta$  is a metric on the extended space of attributes. Let  $a' = a \cup \{dummy_1\}$  and  $b' = b \cup \{dummy_2\}$ , then  $f'$  defined as*

$$f'(x) = \begin{cases} f(x); & \text{if } x \in a, \\ dummy_2; & \text{otherwise.} \end{cases}$$

*is an optimal alignment between  $a'$  and  $b'$  with the same cost as  $f$ . This assignment extension is depicted in Figure 4.6.*

*Proof.* Suppose  $f'$  is not an optimal assignment between  $a'$  and  $b'$  and some  $f^*$  is. Observe that cost of  $f'$  is the same as cost of  $f$  because  $\delta(dummy_1, dummy_2) = 0$ . Choose  $z$  so that  $f^*(dummy_1) = z$ . If  $z$  is a dummy attribute, we can modify  $f^*$  to  $f'^*$  by swapping  $z$  and  $dummy_2$ . Therefore  $f'^*(dummy_1) = dummy_2$ . As both  $z$  and  $dummy_2$  are dummy attributes, this will not affect the cost of  $f^*$ . If  $f'^*$  has lower cost than  $f'$  (and thus consequently  $f$ ), we can improve original  $f$ , which is a contradiction. If it has the same cost then  $f'$  must have been optimal which is also the contradiction.

The remaining case is that  $z$  is not dummy. Let  $k$  be  $f^{*-1}(dummy_2)$ . Modify  $f^*$  as follows:

$$f'^*(x) = \begin{cases} z; & \text{if } x = k, \\ dummy_2; & \text{if } x = dummy_1, \\ f^*(x); & \text{otherwise.} \end{cases}$$

Let us examine cost of  $f'^*$ . We want to prove that this modification of  $f^*$  to  $f'^*$  did not increase the cost. As everything is the same except  $dummy_1$  and  $k$ , we need to investigate only those. We want to prove that

$$\delta(k, z) + \delta(dummy_1, dummy_2) \leq \delta(k, dummy_2) + \delta(dummy_1, z).$$

Using the coincidence axiom the above is equivalent to

$$\delta(k, z) \leq \delta(k, dummy_2) + \delta(dummy_1, z)$$

as  $\delta(dummy_1, dummy_2) = 0$ . This is of course true because it is an instance of triangle inequality.

Now we can make the same argument we did with the  $z$  being a dummy attribute: If  $f'^*$  has lower cost than  $f'$  (and thus consequently  $f$ ), we can improve original  $f$ , which is a contradiction. If it has the same cost then  $f'$  must have been optimal, which is also the contradiction. In all cases,  $f'$  must have been optimal alignment.  $\square$

By applying this theorem multiple times, we can add an arbitrary number of *dummy* attributes without changing the optimality of alignment.

**Theorem 14.** *Let  $a, b$  are list of attributes,  $\delta$  is a metric on the extended space of attributes. Then Algorithm 12 preserves all metric axioms and resulting distance  $\Delta$  on the dataset space is a metric.*

*Proof.* We will split the proof according to the different metric axioms.

1. We will begin with proving the non-negativity axiom. Since  $\delta$  is a metric, it satisfies non negativity axiom. The cost of the minimal assignment must be non-negative as well.
2. For the coincidence axiom we have to prove both direction of the equivalence:
  - $\Rightarrow$ : If  $\Delta(x, y) = 0$ , the cost of minimal assignment was 0. As  $\delta$  satisfies non-negativity, it must be the case that all attributes were equal since  $\delta$  also satisfies coincidence axiom. As all attributes were equal, the  $x$  and  $y$  must also be equal (up to permutation).
  - $\Leftarrow$ : If  $x = y$ , all attributes must be equal (up to a permutation). Optimal solution is for every attribute assign the equal attribute. As  $\delta$  satisfies coincidence and non-negativity, this optimal solution cost is 0.

3. The proof of the symmetry axiom is as follows: given two datasets  $a$  and  $b$ , the  $a$  and  $b$  are either of the same cardinality or the dataset with fewer attributes is extended by the appropriate number of *dummy* attributes, so the cardinalities of the datasets match. This is done regardless of the order of the arguments. Therefore, for the rest of this part, we can assume that the dataset have the same number of attributes. Hungarian algorithm would find an optimal alignment  $f$  – a bijection from  $a$  to  $b$ . If we would switch the arguments, algorithm would find the optimal bijection  $g$  from  $b$  to  $a$ . However, in both cases the algorithm would optimize the same thing, therefore the  $\text{cost}(f) = \text{cost}(g)$ , which would be the output of the algorithms regardless of their order.
4. The remaining axiom to prove is the triangle inequality axiom. Let  $x, y$  and  $z$  be arbitrary datasets,  $\delta$  metric on the attribute space and  $\Delta$  dataset distance measure produced by the algorithm. We want to prove that  $\Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$ . The algorithm would produce optimal assignments  $f, g$  and  $h$  between  $x', y', x, z$  and  $z, y$  respectively. We can reformulate our goal to proving that  $\text{cost}(f) \leq \text{cost}(g) + \text{cost}(h)$ . If the cardinality of the datasets would not match the corresponding amount of *dummy* attributes would be added to the datasets for the sake of the assignments. Now we will take maximum cardinality of the domains of the assignments:

$$\text{max}_{\text{card}} = \max(\mathbf{card}(\text{domain}(f)), \mathbf{card}(\text{domain}(g)), \mathbf{card}(\text{domain}(h))).$$

We will now extend the datasets by adding extra *dummy* attributes so the number of attributes matches the  $\text{max}_{\text{card}}$ . We get datasets  $x', y'$  and  $z'$ . Corresponding new optimal assignments would be  $f', g'$  and  $h'$ . We will argue that we did not change the costs of the assignments. If needed, algorithm would first add the *dummy* attributes to get the original assignments. If the domain does not match the  $\text{max}_{\text{card}}$  we could use Theorem 13. With its help we could add the desired amount of *dummy* attributes finally reaching to extended datasets and another optimal assignment with the same cost as the original one. Note that the following now holds:

$$\mathbf{card}(\text{domain}(f')) = \mathbf{card}(\text{domain}(g')) = \mathbf{card}(\text{domain}(h')).$$

We proceed with creating the suboptimal assignment  $f^\circ$  from  $\text{domain}(f')$  to  $\text{range}(f')$  by function composition of  $g'$  and  $h'$  –  $f^\circ = h' \circ g'$ . We can do that as  $\text{domain}(f') = \text{domain}(g')$  and  $\text{range}(f') = \text{range}(h')$ . As  $f'$  is optimal we get

$$\text{cost}(f) = \text{cost}(f') \leq \text{cost}(f^\circ) = \sum_{i=1}^{\text{len}(x')} \delta(x[i], f^\circ(x[i])).$$

According to the fact that  $\delta$  satisfies triangle inequality and the fact that  $f^\circ = h' \circ g'$  we get

$$\sum_{i=1}^{\text{len}(x')} \delta(x[i], f^\circ(x[i])) \leq \sum_{i=1}^{\text{len}(x')} (\delta(x[i], g'(x[i])) + \delta(g'(x[i]), h'(g'(x[i])))).$$

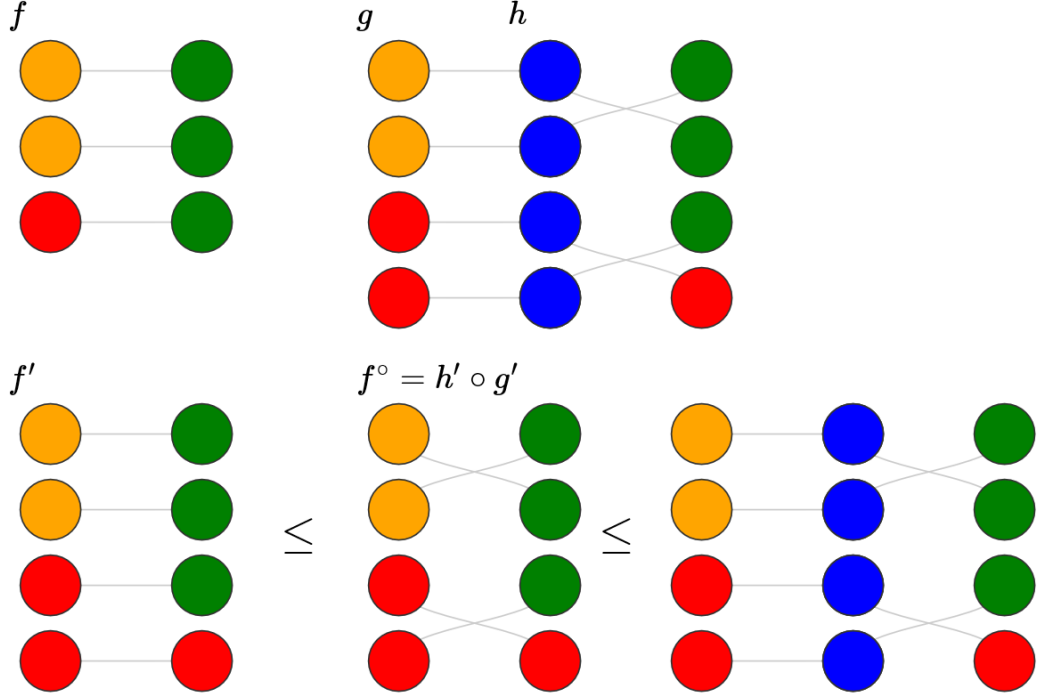


Figure 4.7: Proof of Theorem 14 – metric axiom 4 is preserved. Alignment  $f$  is extended and then reshuffled so the triangle inequality can be applied. Yellow nodes are original attributes of dataset  $x$ , green nodes are attributes of dataset  $y$  and blue of dataset  $z$ . Red nodes are *dummy* attributes used for extending the datasets so the corresponding assignments are of the same cardinality.

As  $\text{range}(g') = \text{domain}(h')$  and every assignment function is a bijection, summing over elements of  $x'$  is the same as summing over the permutation of elements of  $x'$  given by assignments, we can conclude the proof:

$$\begin{aligned} \sum_{i=1}^{\text{len}(x')} (\delta(x[i], g'(x[i])) + \delta(g'(x[i]), h'(g'(x[i]))) &= \text{cost}(g') + \text{cost}(h') \\ &= \text{cost}(g) + \text{cost}(h). \end{aligned}$$

The whole proof of the triangle inequality is illustrated in Figure 4.7.

As all metric axioms are proven, we can conclude the proof.  $\square$

The previous theorem also holds for Algorithm 16:

**Corollary 4.** *Let  $a, b$  be lists of attributes,  $\{\delta_1, \dots, \delta_n\}$  are metrics on the extended space of attributes,  $\{s_1, \dots, s_n\}$  are selectors,  $\{w_1, \dots, w_n\}$  are positive weights of each selector. Then Algorithm 16 preserves all metric axioms and resulting distance on the dataset space is a metric.*

*Proof.* Follows from the previous theorem and the fact that sum of metrics with positive weights is also a metric (Corollary 1).  $\square$

It may be interesting to see how to extend attribute space by an artificial *dummy* attribute. A following theorem gives us some clue:

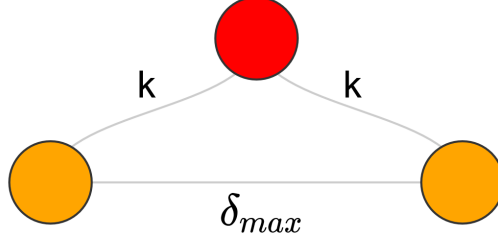


Figure 4.8: Part of proof of Theorem 15. Constant  $k$  representing a distance between a regular and an artificial  $dummy_n$  attribute needs to be big enough in order not to create the shortcut between the most distant points.

**Theorem 15.** *Let  $\delta'$  be a metric on a non-empty attribute space  $\mathbb{A}$ . Let  $\delta$  be a distance derived from  $\delta'$  by extending  $\mathbb{A}$  by artificial  $dummy_n$  attribute. We set the  $\delta(dummy_n, x)$  and  $\delta(x, dummy_n)$  to some constant  $k \forall x \in \mathbb{A}$ . Finally, we set  $\delta(dummy_n, dummy_n) = 0$ . Let  $\delta_{max} = \max_{x, y \in \mathbb{A}} \delta(x, y)$ . If  $\delta$  is a metric on the extended attribute space then  $0 < k$  and  $\frac{\delta_{max}}{2} \leq k$ .*

*Proof.* If  $k < 0$  then  $\delta(dummy_n, x) < 0$ , which would contradict non-negativity. If  $k = 0$  then  $\delta(dummy_n, x) = 0$  and for every  $x \in \mathbb{A}$  is  $x \neq dummy_n$ , which contradicts coincidence axiom. The idea of the remaining part is as follows: we cannot short-cut our way when going on the longest way in the space by going through  $dummy_n$  attribute which has the constant distance from every other point in the space – this is depicted in Figure 4.8. Find  $x_0, y_0$  so that  $\delta(x_0, y_0) = \delta_{max}$ . From triangle inequality we get  $\forall x, y \in \mathbb{A} : \delta(x, y) \leq \delta(x, dummy_n) + \delta(dummy_n, y)$ . All the more true that  $\delta(x_0, y_0) \leq \delta(x_0, dummy_n) + \delta(dummy_n, y_0)$ . As  $\forall z \in \mathbb{A} : \delta(dummy_n, z) = \delta(z, dummy_n) = k$  we get  $\delta_{max} \leq 2k$  which is equivalent to  $\frac{\delta_{max}}{2} \leq k$ .  $\square$

If we want to have metric attribute distance, and at the same time, a constant distance from artificial  $dummy_n$  attributes, we have to choose this constant from quite large values. Doing this can create unwanted side affects. The penalty for adding  $dummy_n$  attributes can become the most significant part of the distance, especially if the number of attributes varies by large amounts. It can be argued that if some attributes are not very relevant for the prediction of the target (for example attributes with low entropy), the two datasets should not differ too much if one dataset is missing such not very relevant attributes. Furthermore, in some cases it may be impossible to compute  $\delta_{max}$  or to learn the value in advance.

We get no such problems if we use the element of the attribute space itself, if we already made sure that the attribute distance is a metric. Using the  $dummy_a$  attribute from the space will not break the metric axioms (if the attribute distance  $\delta$  is a metric).

Theorem 14 is also valid in the opposite direction, as shown in the following theorem.

**Theorem 16.** *Let  $a, b$  are list of attributes,  $\delta$  is a distance between attributes. If  $\Delta$  is a distance over dataset space produced by Algorithm 12 and  $\Delta$  is a metric on the dataset space, then  $\delta$  is a metric on the extended attribute space  $\mathbb{A}$ .*

*Proof.* Again, we will split the proof according to the different metric axioms.

1. We will begin with proving the non-negativity axiom. For the sake of contradiction, let us suppose that  $\delta$  does not satisfy non-negativity axiom. It must be the case that  $\exists x, y \in \mathbb{A}$  so  $\delta(x, y) < 0$ . We define datasets  $X$  and  $Y$  consisting solely of  $x$  and  $y$  respectively. There is only a single assignment  $f$  available – mapping  $x$  to  $y$  and is therefore optimal. Algorithm 12 will output the cost( $f$ ):  $\Delta(X, Y) \sum_{i=1}^1 \delta(X[i], f(X[i])) = \delta(x, y)$ . We get a contradiction as  $\Delta(X, Y) \geq 0$  and  $\delta(x, y) < 0$ .
2. For the coincidence axiom we have to prove both direction of the equivalence:
  - $\Rightarrow$ : If  $\delta(x, y) = 0 \wedge x \neq y$ , we define datasets  $X$  and  $Y$  consisting solely of  $x$  and  $y$  respectively.  $X \neq Y$  and as  $\Delta$  is a metric we get  $\Delta(X, Y) > 0$ . As  $\mathbf{card}(X) = \mathbf{card}(Y) = 1$  there is only a single assignment  $f$  available and must be the optimal alignment returned by Algorithm 10. The cost outputted by Algorithm 12 is  $\Delta(X, Y) \sum_{i=1}^1 \delta(X[i], f(X[i])) = \delta(x, y) = 0$ , which is a contradiction.
  - $\Leftarrow$ : If  $x = y \wedge \delta(x, y) > 0$  we define datasets  $X$  and  $Y$  consisting solely of  $x$ .  $X = Y$  and as  $\Delta$  is a metric we get  $\Delta(X, Y) = 0$ . As  $\mathbf{card}(X) = \mathbf{card}(Y) = 1$  there is only a single assignment  $f$  available and must be the optimal alignment used by Algorithm 12. The cost outputted by the algorithm is  $\Delta(X, Y) \sum_{i=1}^1 \delta(X[i], f(X[i])) = \delta(x, y) > 0$ , which is a contradiction.
3. The proof of the symmetry axiom is as follows. For the sake of contradiction, let us assume that  $\delta$  does not satisfy symmetry axiom. It must be the case that  $\exists x, y \in \mathbb{A}$  so  $\delta(x, y) \neq \delta(y, x)$ . We will define datasets  $X$  and  $Y$  consisting solely of  $x$  and  $y$  respectively. There is only a single assignment  $f$  from  $X$  to  $Y$  available – mapping  $x$  to  $y$  and is therefore optimal. Similarly, there is only a single assignment  $g$  from  $Y$  to  $X$  available – mapping  $y$  to  $x$  and is therefore optimal. Algorithm 12 will output the cost( $f$ ):

$$\Delta(X, Y) \sum_{i=1}^1 \delta(X[i], f(X[i])) = \delta(x, y).$$

The cost( $g$ ) is computed analogically:

$$\Delta(Y, X) \sum_{i=1}^1 \delta(Y[i], g(Y[i])) = \delta(y, x).$$

We get a contradiction. As  $\Delta$  satisfies symmetry axiom we get  $\Delta(X, Y) = \Delta(Y, X)$ . However, at the same time we have  $\Delta(X, Y) \neq \Delta(Y, X)$  from assumptions.

4. The remaining axiom to prove is the triangle inequality axiom. For the sake of contradiction, let us suppose that  $\delta$  does not satisfy triangle inequality. It must be the case that  $\exists x, y, z \in \mathbb{A}$  so  $\delta(x, y) > \delta(x, z) + \delta(z, x)$ . We will define datasets  $X$ ,  $Y$  and  $Z$  consisting solely of  $x, y$  and  $z$  respectively. There is only a single assignment  $f$  from  $X$  to  $Y$  available – mapping  $x$  to

$y$  and is therefore optimal. Similarly, there is only a single assignment  $g$  from  $X$  to  $Z$  available and only a single assignment  $h$  from  $Z$  to  $Y$  – both are also optimal. Algorithm 12 will output the  $\text{cost}(f)$ :

$$\Delta(X, Y) \sum_{i=1}^1 \delta(X[i], f(X[i])) = \delta(x, y).$$

Again, the  $\text{cost}(g)$  and  $\text{cost}(h)$  are computed analogically:

$$\Delta(X, Z) \sum_{i=1}^1 \delta(X[i], g(X[i])) = \delta(x, z),$$

$$\Delta(Z, Y) \sum_{i=1}^1 \delta(Z[i], h(Z[i])) = \delta(z, y).$$

As  $\Delta$  satisfies triangle inequality we have  $\Delta(X, Y) \leq \Delta(X, Z) + \Delta(Z, Y)$ . At the same time we have  $\Delta(X, Y) > \Delta(X, Z) + \Delta(Z, Y)$  from assumptions, which is a contradiction.

As all metric axioms are proven, we can conclude the proof.  $\square$

Similar theorem holds for Algorithm 16 with addition that all selectors must be distinct:

**Corollary 5.** *Let  $a, b$  are list of attributes,  $\{\delta_1, \dots, \delta_n\}$  are the distances on the extended space of attributes,  $[s_1, \dots, s_n]$  are selectors,  $[w_1, \dots, w_n]$  are weights of each selector. If  $\forall k, j \in \{1, \dots, n\}, \forall \mathbb{A}' \subseteq \mathbb{A} : s_k(\mathbb{A}') \cap s_j(\mathbb{A}') = \emptyset$  and distance  $\Delta$  produced by Algorithm 16 is a metric on the dataset space then  $\{\delta_1, \dots, \delta_n\}$  are metrics on the extended attribute space defined by the corresponding selector.*

*Proof.* We can prove the corollary by following the proof of the previous theorem. When creating dataset of a single element, we now do this for every  $\delta_i$  that does not satisfy axiom in question. As selectors are distinct, other selectors will return  $\emptyset$  from single element datasets and therefore will output 0 for corresponding selector distance, as the resulting distance is the sum over all attributes returned by the selector.  $\square$

It is intuitive to have each selector distinct, if we think of a selector in the sense we have introduced them: selector selects specific attributes (e.g. categorical or numerical) so more fine grained distance functions can be applied. In this sense the selectors will be distinct as the subset of categorical attributes is clearly disjunct to subset of numerical attributes.

We can wonder whether Corollary 5 holds even without the constraint for distinct selectors.

**Observation 3.** *The requirement for distinct selectors is an essential part of Corollary 5.*

*Proof.* We will show a counterexample: let attribute space be the set of two elements:  $\mathbb{A} = a_1, a_2$ . Let  $s_1, s_2$  be two selectors defined as  $s_1(\mathbb{X}) = s_2(\mathbb{X}) = \mathbb{X}$ . Let  $\delta_1$  be a metric and  $\delta_2$  be a distance function defined as  $\delta_2(x, y) = -0.1\delta_1(x, y)$ .

$\delta_2$  is not a metric on the attribute space as it violates non-negativity –  $\delta_2(a_1, a_2) = -0.1\delta_1(a_1, a_2) < 0$ . It still satisfies symmetry and coincidence though. We can combine  $\delta_1$  and  $\delta_2$  to form  $\delta_3$ :  $\delta_3(x, y) = \delta_1(x, y) + \delta_2(x, y) = 0.9\delta_1(x, y)$ .  $\Delta$  induced by  $\delta_1$ ,  $\delta_2$ , and  $s_1 = s_2$  is the same as  $\Delta'$  induced by  $\delta_3$ .  $\delta_3$  is a metric on the attribute space according to Theorem 9 as it is positively rescaled  $\delta_1$ . Therefore, it does not matter if we use the combination of  $\delta_1, \delta_2$  or just  $\delta_3$  alone. Resulting  $\Delta$  is a metric on the dataset space according to Theorem 14 even-though  $\delta_2$  is not a metric on the dataset space.  $\square$

Theorems 14, 16 and Corollaries 4 and 5 are useful as they state that if we can get a metric on the attribute or dataset space, we get other metric on the other space for free when using attribute aligning. During training, we could define a function measuring metric similarity on the attribute or dataset samples. As the samples would usually be just small finite subsets of attribute or datasets space, we could be wondering whether by optimizing metric on some dataset or attribute subset, we would get also metric on the other subset as Theorems 14, 16 and Corollaries 4, 5 are valid only for the whole spaces. We will try to formalize this motion:

**Definition 39.** Let  $\mathbb{A}$  be attribute space, let  $A$  be the subset of  $\mathbb{A}$ . Let  $\mathbb{D}$  be a dataset space. Then we will denote dataset subspace  $D$  as supported by attribute subspace  $A$  if  $D \subseteq \mathbb{D}$  and if  $\forall d \in D, \forall att \in d : att \in A$ .

In other words, datasets in  $D$  are only composed of elements in  $A$ . This definition allow us to investigate metric properties in just the subset of attribute spaces and conclude properties in supported subspaces of datasets (like training and testing subset) and vice-versa.

**Definition 40.** Let  $\mathbb{A}$  be attribute space, let  $\mathbb{D}$  be a dataset space and  $D$  its subset. We will call the subset  $A$  of  $\mathbb{A}$  as the source of  $D$  if  $\forall att \in \mathbb{A} : att \in A \iff \exists d \in D : att \in d$ .

In other words, elements of  $A$  are just enough in order to create all elements in  $D$ . Similarly to the remark to Definition 39, we can use this definition to reason about whether properties that are valid for some distance function on some set of datasets are also valid for the attribute source of these datasets.

If we optimize metric on subset of attributes, we also get metric on all datasets that can be combined by this subset of attributes when using Algorithm 12:

**Theorem 17.** Let  $\mathbb{A}$  be extended space of attributes and  $A$  its extended subset,  $\delta$  is a metric on  $A$ ,  $\mathbb{D}$  space of datasets. Then  $\forall D \subseteq \mathbb{D}$ ,  $D$  supported by  $A$ : Algorithm 12 preserves all metric axioms and resulting distance  $\Delta$  on the  $D$  is a metric.

*Proof.* From following the proof of Theorem 14 as the proof does not require elements outside of  $D$  or  $A$  and we can replace the whole  $\mathbb{D}$  by  $D$  and  $\mathbb{A}$  by  $A$ .  $\square$

Intuitively, the same works for Algorithm 16.

This enables us to define our algorithms in such a way, that if we can create a metric on all the attributes in the training and testing samples, we can also

Table 4.7: Counterexample that metric on some subspace of datasets does not imply metric on its source attribute subspace

	$a_1$	$a_2$	$a_3$
$a_1$	0	50	50
$a_2$	50	-5	50
$a_3$	50	50	5

guarantee the resulting metric on all supported subsets of the dataset space. The training and testing datasets must be among them, as they are supported by the testing and training attributes.

Note that Theorem 17 is valid because there is nothing in the proof of Theorem 14 that would require elements (datasets) outside of the subset  $D$  and  $A$ . As for the other direction, when observing the proof of Theorem 16, we use the trick that we artificially create datasets with a single element. However, such datasets can be outside of  $D$ . We can then wonder whether Theorem 17 is valid also in the other direction considering Algorithm 12. It is not according to Observation 4.

**Observation 4.** *Let  $\mathbb{A}$  be space of attributes and  $\mathbb{D}$  space of datasets. Let  $D$  be a subset of  $\mathbb{D}$  and  $\Delta$  a metric on  $D$ . Let  $A$  be the source of  $D$  and  $\delta$  be a distance measure on  $A$ , such that Algorithm 12 induces  $\Delta$  using  $\delta$ . Then  $\delta$  is not necessarily a metric on  $A$ .*

*Proof.* We will show a counterexample. Let  $\mathbb{A}$  be the space of attributes,  $D$  be the subspace of datasets consisting of single dataset  $d = \{a_1, a_2, a_3\}$ . Let  $A = \{a_1, a_2, a_3\} \subseteq \mathbb{A}$  the source of  $D$ . Let  $\Delta$  be a metric on  $D$ . Since  $D$  consists of only one element, in order  $d$  to be a metric all we need to do is set  $\Delta(d, d) = 0$ . Let  $\delta$  be attribute distance function on  $A$ . We will show that even though  $\Delta$  is a metric,  $\delta$  does not have necessarily to be a metric. We can define  $\delta$  as shown in Table 4.7. As there are some negative values, the attribute distance  $\delta$  is not a metric. The optimal alignment  $f$  between  $d$  and  $d$  is coloured and is defined as  $f(a_i) = a_i$ . The cost( $f$ ) =  $\delta(a_1, a_1) + \delta(a_2, a_2) + \delta(a_3, a_3) = 0 + 5 - 5 = 0 = \Delta(d, d)$ .  $\square$

Same counterexample can be found when using Algorithm 16 as this algorithm is a generalization of Algorithm 12, therefore the same argument can be applied. This implies that we cannot guarantee metric on the subspace of attributes appearing in the training and testing datasets even if we can guarantee resulting distance to be a metric on the training and testing dataset subspace.

Another argument favouring a metric for the attributes is that specialized algorithms can be used for the assignment such as [5].

## 4.6 Distance Using Attribute Metadata

In Section 3.2, we discussed distance based on a vector of global metafeatures. We can analogically define a distance on the attribute space or subspace defined by some selector. This can be then the *IAtributeDistance* input for Algorithms 12 and 16.

*Attribute\_metafeatures* is a property returning real value vector of attribute metafeatures for a given attribute. This property does not have to be necessarily

---

**Algorithm 17:** Vectorized Attribute Distance: *IAttributeDistance*

---

// Pseudocode for measuring distance between attribute  
treating attributes as real valued vector.

**input** :  $\sigma \leftarrow$  Vector distance measure

**input** :  $x \leftarrow$  First attribute

**input** :  $y \leftarrow$  Second attribute

**output:** Distance between two attributes

```
1 vectorx  $\leftarrow$  xattribute_metafeatures;  
2 vectory  $\leftarrow$  yattribute_metafeatures;  
3 distance  $\leftarrow$   $\sigma(\text{vector}_x, \text{vector}_y)$ ;  
4 return distance;
```

---

defined on the whole attribute space. For example, in the case of Algorithm 16 it must be defined only on the subspace of attributes defined by the corresponding selector, e.g. it can return vector of features specific for numerical attributes in the case of numerical selector.

Again, note that both variables  $\text{vector}_x, \text{vector}_y$  are real valued vectors of the same length  $n$ . This allows us to use any metric defined on  $\mathbb{R}^n$  including all the metrics based on  $p$ -norms (Theorem 6) and their weighted counterparts (Theorem 8) and the resulting attribute measure will also be a metric. Using such metric in Algorithms 12 and 16 will preserve the metric and the resulting dataset distance will be also a metric according to Theorems 14 and 17, and Corollary 4 and its selector counterpart.

## 4.7 Combining the Distances

We have dedicated lots of effort to utilize extra information from the attributes. We have the whole workflow that builds the distance on the datasets from the attribute distance through aligning and selectors. In this section, we would like to address the fact that global attributes store useful information as well – this was proven in the literature. Even though the attribute alignment was competitive to global dataset distances, it does not necessarily mean that we have to use them separately from each other. In fact, it could be useful to combine their powers to get even better distance measure between datasets. In order to achieve this we have proposed Algorithm 18.

This algorithm takes list of dataset distance measures and weights their results. If the underlying datasets measures are metric and weights are positive, the algorithm will also produces a metric on the datasets space according to the fact that weighted sum of metrics is also a metric, if the weights are positive (Corollary 1). As usual, using the *partial* application we can conform to the *IDatasetDistance* interface, if we fix all the arguments except the last two.

---

**Algorithm 18:** Dataset Distance Aggregation: *IDatasetDistance*

---

```
// Pseudocode for combining multiple dataset distance measure
// and producing their weighted combination.
input : distances  $\leftarrow$  List of dataset distance measures  $\Delta$ 
input : weights  $\leftarrow$  List of weights
input :  $x \leftarrow$  First dataset
input :  $y \leftarrow$  Second dataset
output: Distance between two datasets

1 distance  $\leftarrow$  0;
2 for  $i$  in  $\{1, \dots, \text{len}(\text{distances})\}$  do
3   | distance  $\leftarrow$  distance + weights[ $i$ ] $\text{distances}[i](x, y)$ ;
4 end
5 return distance;
```

---

## 4.8 Normalization Based on the Number of Attributes

When using Algorithms 9, 12 and 16, after getting the total distance defined by the optimal alignment, we could think about normalizing this distance by the number of attributes. This would amend the algorithms. Algorithm 12 would return  $\frac{\text{TotalDistance}}{\text{NumberOfAttributes}}$  instead. Algorithm 16 would do this amendment for each selector (and number of attributes would be based on the number of attributes selected by the selector). Such amendments would normalize this distance to a count independent on the attribute number. This may or may not be beneficial, but in this section we show that there are theoretical reasons against it.

There are two natural ways how to normalize by the number of attributes. Given two datasets  $a$  and  $b$  with the number of attributes  $|a|$  and  $|b|$ , we can either divide the total distance by  $\min(|a|, |b|)$  or by  $\max(|a|, |b|)$ . We will start by exploring the latter case –  $\max(|a|, |b|)$ . Without the normalization, if the underlying attribute distance is a metric, the metric preservation to the dataset distance is ensured by Theorem 14. We will show that this is not necessary true if we do the normalization according to the  $\max(|a|, |b|)$ .

**Observation 5.** *Normalizing the distance by  $\max(|a|, |b|)$  can violate metric axioms.*

*Proof.* Let  $\delta$  be a metric on the extended attribute space  $\mathbb{A}$ ,  $dummy \in \mathbb{A}$ ,  $a = \{a_1\}$ ,  $b = \{b_1\}$  are datasets. Let us define the  $\delta(a_1, b_1)$  as 1 and set  $\delta(a_1, dummy) = \delta(b_1, dummy) = 10$ . The optimal alignment is the only one possible,  $\max(|a|, |b|)$  is 1, therefore distance  $\Delta$  returned by the normalized Algorithm 12 is  $\frac{1}{1}$ . We will show that the triangle inequality does not have to be preserved. We will define dataset  $z$  as  $\{a_1, dummy\}$ . For the triangle inequality to hold, it must be the case that  $\Delta(a, b) \leq \Delta(a, z) + \Delta(z, b)$ . Let us see what  $\Delta(a, z)$  is. One *dummy* attribute will be added, as  $\delta(dummy, dummy) = 0$ , we get optimal alignment of the cost of zero as it will align  $a_1$  to  $a_1$  and *dummy* to *dummy*. The zero will be divided by  $\max(|a|, |z|)$  resulting in  $\Delta(a, z) = 0$ . As for the  $\Delta(z, b)$ , one *dummy* attribute is again added to  $b$ , the optimal alignment is either  $a_1 \rightarrow b_1, dummy \rightarrow dummy$

or  $a_1 \rightarrow dummy, dummy \rightarrow b_1$ . The cost of the former is 1, the cost of the latter is  $2 \times 10 = 20$ . Therefore, the former is optimal. As  $\max(|b|, |z|) = 2$  we get  $\Delta(z, b) = 1/2$ . Finally we get  $1 = \Delta(a, b) \leq \Delta(a, z) + \Delta(z, b) = 1/2$  which is not valid and distance on the dataset space  $\Delta$  is not a metric, as the triangle inequality was broken.  $\square$

The same holds for the former case –  $\min(|a|, |b|)$ .

**Observation 6.** *Normalizing the distance by  $\min(|a|, |b|)$  can violate metric axioms.*

*Proof.* Again, let  $\delta$  be a metric on the extended attribute space  $\mathbb{A}$ ,  $dummy \in \mathbb{A}$ . But this time we define  $a = \{a_1, a_2\}, b = \{b_1\}$ . Let us define  $\delta(a_1, b_1) = \delta(a_2, dummy) = 1$  and set  $\delta(a_2, b_1) = 10$ . The optimal alignment is – after adding one dummy attribute  $dummy$  to  $b$  –  $a_1 \rightarrow b_1$  and  $a_2 \rightarrow dummy$ . The cost of this alignment is 2 and as  $\min(|a|, |b|) = 1$  the resulting  $\Delta(a, b) = 2/1 = 2$ . Again, we break the triangle inequality. We will define dataset  $z$  as  $\{b_1, dummy\}$ . For the triangle inequality to hold, it must be the case that  $\Delta(a, b) \leq \Delta(a, z) + \Delta(z, b)$ . The  $\Delta(a, z)$  is calculated according to the optimal alignment. As  $z$  is the same as  $b$  with one added  $dummy$  attribute, the optimal alignment must be the same as the optimal alignment from  $a$  to  $b$ . The cost is therefore also 2. But this time  $\min(|a|, |z|) = 2$  and we get  $\Delta(a, z) = 2/2 = 1$ . As for the  $\Delta(z, b)$  we use the same argument –  $z$  is the same as  $b$  with added  $dummy$ . During the alignment, one  $dummy$  will be truly added to  $b$  and consequently the cost of the optimal alignment will be zero.  $\Delta(z, b)$  is therefore 0. Finally we get  $2 = \Delta(a, b) \leq \Delta(a, z) + \Delta(z, b) = 1$ , which again breaks the triangle inequality and therefore  $\Delta$  is not a metric.  $\square$

We have given the counterexamples for Algorithm 12, however the same is valid for Algorithm 16. As already stated in Section 4.3, Algorithm 12 is a special case of Algorithm 16 and every counterexample is therefore valid even for more generic algorithm.

According the theoretical results stated in this section, we will not use this normalization in the rest of the thesis as this modification could violate metric axioms of the resulting dataset distance.

# Chapter 5

## Obtaining the Data

To train the models, as proposed in the previous chapters, and to validate their ranking qualities, data has to be obtained beforehand. We also need to know what metadata will be available before specifying global and attribute distances. Therefore, we need to gather datasets, metafeatures of those datasets, machine learning algorithms and experiment results prior to conducting the experiments with the ranking algorithms. The potential sources of such data and what metafeatures to use is the topic of this chapter. We will begin by defining the common format for storing datasets. Then we will discuss machine learning repositories and discuss their pros and cons. The differences lie mainly in the types of data available. We will discuss one particular machine learning repository – OpenML – in more details as it will be used as the main datasource. We then describe the dump we downloaded from the OpenML, filters we used to clean the dump, and the total amount of data we had after the filtering. This will include the decision to perform the ranking on the classification algorithms, therefore including the filter on classification tasks and algorithms only. We list the global metadata provided by the OpenML and discuss the subset that will be used for the experiments. As OpenML does not provide attribute metadata, we then review the types of attribute metadata we extracted. We look into the distribution of attribute metadata in more detail and identify few potential problems in the distribution of attribute metadata. To tackle them, we introduce another attribute metafeatures that are calculated based on the previous metadata but do not suffer from the same problems.

### 5.1 ARFF Format

In Section 2.1, we introduced the concept of datasets as a relation. That was however a mere theoretical description. The specification of mapping this theoretical description to a file is still needed. In this section, we describe the popular format called *ARFF* (Attribute-Relation File Format) for storing datasets and therefore also classification and regression tasks. Its importance lies in that almost every machine learning tool and library supports ARFF and large amount of public datasets is distributed using this very format.

ARFF is a file format usually recognizable by the ".arff" file extension. An ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes [1]. The file consists of two sections. The first one – called Header

– contains descriptions, name of the relation and the list of attributes and their types. Description lines start with ”%” and can contain arbitrary information.

The relation name is defined as the first line in the ARFF file. The format is: ”@relation \$relation-name”, where \$relation-name is a string. The string must be quoted if the name includes spaces. The format for the @attribute statement is: ”@attribute \$attribute-name \$datatype”, where the \$attribute-name must start with an alphabetic character. If spaces are to be included in the name, then the entire name must be quoted.

The \$datatype can be any of the following:

- numeric
- integer
- nominal
- string
- date

The header can contain arbitrary number of attributes.

The second section – called Data – starts with the @data declaration on a single line followed by lines of instances – one data instance per line. Every instance consists of the list of values of the attributes in the same order. Missing values are encoded by ”?”.

The example of the ARFF format describes header and few instances of the Iris dataset [39] and is as follows:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
% (a) Creator: R.A. Fisher
% (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
% (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength NUMERIC
@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petallength NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE class Iris-setosa,Iris-versicolor,Iris-virginica

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

```
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

All declarations are case insensitive.

The advantage of this file format is its simplicity. No special tools or libraries are required to parse ARFF files. This advantage inevitably comes with a big disadvantage – there is no guarantee that the information stored in the file is consistent, especially entries stored in the Header section are often corrupted, or do not follow the format completely.

## 5.2 Machine Learning Repositories

In this section, we discuss machine learning repositories that provide machine learning capabilities that may include datasets, algorithms, results and metadata. Therefore they make ideal candidates for the potential source of data for conducting machine learning experiments.

In the past, perhaps the most popular machine learning repository was the repository of University of California, Irvine abbreviated only as UCI [12]. The sole purpose of the repository was to provide public datasets (mostly in the ARFF format) for machine learning and metalearning experiments. Many works reviewed in this thesis used UCI as the source of data (including some of our works). The major drawback is that the repository does not provide any additional data besides datasets. Even if we used the UCI repository for datasets, additional data would still be needed. In our previous experiments, we combined the repository with our recommendation system Pikater (see Section 2.9) to gather the rest of the data. We used the search agents in Pikater for hyperparameter tuning of machine learning algorithms over UCI datasets. This resulted in a huge database containing 600,000 machine learning experiments results, which we used for our previous experiments with metalearning [110]. The purpose of building such experiments results was mainly in finding the best settings of hyperparameters for some machine learning algorithm on some datasets. Therefore, in terms of datasets, only 85 unique UCI datasets and 8 WEKA models were in these 600,000 results. This is a common phenomenon. Despite the fact that many machine learning experiments are conducted every day, the number of public datasets used for the experiments is small. We wanted to conduct the experiments in the thesis on the bigger dataset. One option was to use the system to create another batch of experiments.

Another option was to use the OpenML [121] machine learning repository that emerged in the recent years. It is a repository of datasets, tasks, machine learning algorithms, and experiment results called runs. Most of UCI datasets are already present in OpenML, although many more datasets are also in the repository. OpenML user can also specify whether the dataset is private or public, however major number of datasets is public. When a new dataset is uploaded to a repository, OpenML automatically extracts metafeatures. There are in total

106 different metafeatures that can be extracted, however not every metafeature is extracted for each dataset. An OpenML task defines experiment over some dataset. It specifies the goal (e.g. supervised classification over the target attribute), estimation procedure (e.g. 10-fold cross-validation) and evaluation measures used (RMSE, PredictiveAccuracy). An OpenML run is the result of some machine learning algorithm on some task.

As the experiment results are standardized, it is easy for researchers to compare the results of machine learning and metalearning methods. Furthermore, as some metafeatures are automatically computed by OpenML, it makes implementation of other metalearning approaches faster, thus speeding up the research. OpenML also exposes data via its REST API. Specialized connector packages for communication with this interface are available for R, Java, .Net (which was created by us) and Python. However, it is possible to write a custom connector, as almost all languages support REST API. Another benefit is that the whole project is under active development and there is a growing community around it.

Not even OpenML is without drawbacks. All datasets are visible including the testing data. With many experiments, there is no guarantee that users will not eventually carry information out of testing datasets into the training by looking at the results of previous experiments on the testing datasets (the phenomenon referred to as *peeking* by authors of [104]). The potential solution can be the gamification used for example in the *Kaggle* site [11]. The testing dataset is not available and researchers have only a limited amount of time or submissions to submit their models. Therefore, the model cannot be built infinitely to reach the best testing result.

With the pros and cons in mind, OpenML was chosen as the source of data mainly because of its public availability, many experiments and datasets including those from UCI and global metadata autocomputed by the repository.

The choice of the data is very important for the quality of experiments. Results can be affected by many factors – amount of errors in the data, whether the data are general or domain specific only, etc. It should be taken into consideration that OpenML repository is very general and public. Therefore, it may contain errors or noise, and our algorithms have to handle every type of dataset.

## 5.3 OpenML Dump

In the previous section, we discussed the potential source of data and have chosen OpenML as the main data source. Originally, our OpenML dump contained 791 public datasets. We have placed extra requirements on the datasets in order to fulfil following two requirements:

- Keep the computation of alignment reasonable for all pairs of datasets in the chosen subset. This will speed up the evaluation of quality of alignment predictor. The computation cost depends on the number of attributes, the goal is to find a good compromise between the amount of datasets that we can use and the cost of alignment of their attributes.
- In order to be able to easily compare metalearning approaches, it is desirable that every dataset has the same types of metafeatures extracted.

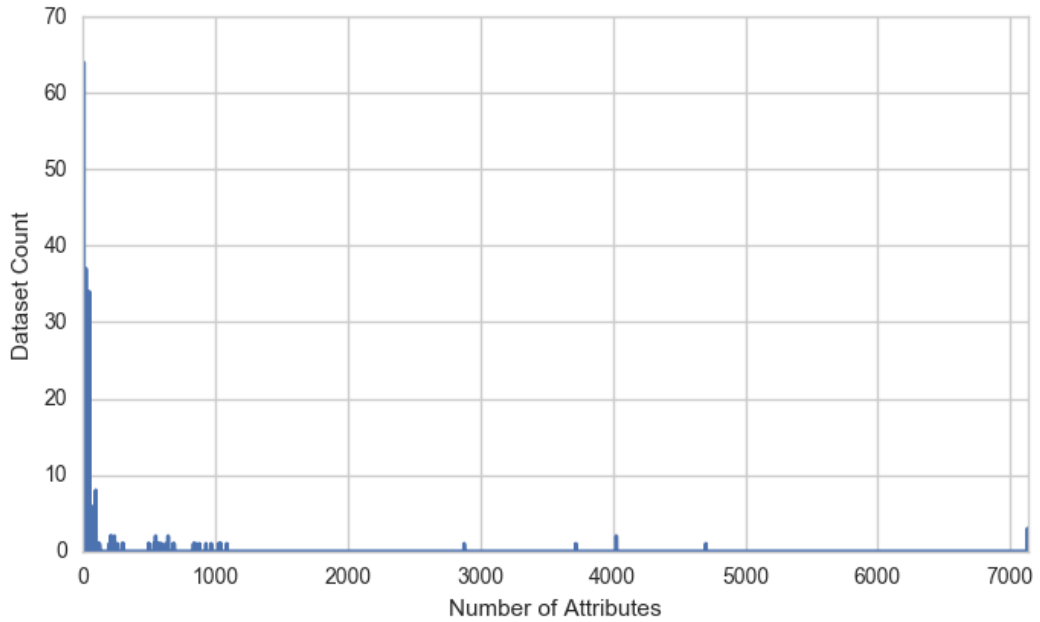


Figure 5.1: Distribution of number of attributes among datasets in the OpenML dump.

This may not be possible in every case, as some global metafeatures are only computable for classification/regression tasks. Because of this, some compromises may be needed. Similarly, some attribute metafeatures may be computable only for classification/numerical attributes but this does not concern us too much, as we will be using selectors (see Algorithm 16) to handle different types of attributes).

To find out the optimal compromise between the amount of datasets and attributes, we have plotted the distribution of number of attributes among datasets. The distribution is shown in Figure 5.1. It should be clear from the figure that only a very small number of datasets have large number of attributes. The alignment of this minority of datasets would take the majority of time. We have decided to filter the datasets with more than 50 attributes.

The remaining datasets were examined for the distribution of the OpenML global metafeatures. The distribution is shown in Figure 5.2. Not all global metafeatures are computed for each dataset. Luckily for us, in this case few metafeatures have useful property ensuring that if this metafeature was computed, then all others are computed (example of such metafeature was `kNN_3NKappa`). To fulfil the second criterion, it was then sufficient enough just to create a filter that one of these metafeatures cannot be null. After application of this filter, only classification datasets remained.

This data preprocessing included only dataset specific view. However, in our experiments the experiment results are also needed, which – in the case of OpenML – are covered by OpenML tasks and runs. First requirement was clear – we can include only those datasets with the results available. Before stating another requirements, we have to choose some sort of performance criterion that will be used. We have already made the choice of restricting the datasets to classification datasets only. There are lots of performance measures possible for

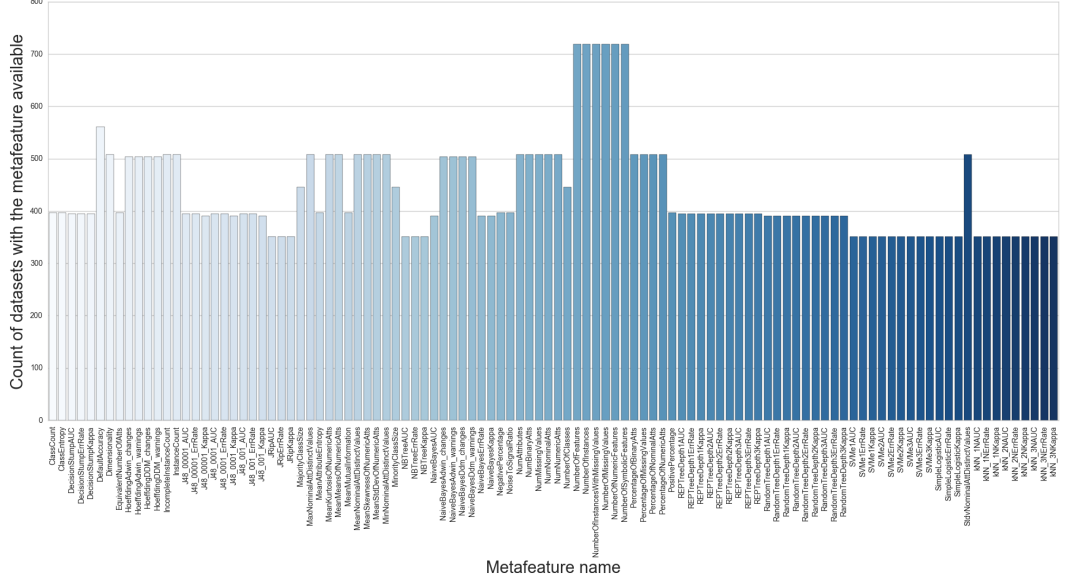


Figure 5.2: Distribution of global metafeatures computed by OpenML.

the classification task. When choosing a suitable measure, we wanted to include the criterion that is often used. This would minimize the filtering of datasets as this performance measure will be often available for each classification task. Based on this reasoning, we have chosen Predictive Accuracy (see Definition 9). There could still be multiple classification tasks defined with Predictive Accuracy as the performance measure – they could still differ in the estimation procedure (simple training/testing split of the dataset, cross-validation, etc.). In order not to filter the data further, we have decided to neglect this difference, as the performance indicator is well defined regardless of the estimation procedure. Given the above, we can state the second requirement: we will include only those datasets that have some experiments results available with the Predictive Accuracy as one of the evaluation measures.

There could be multiple results for a pair of algorithm, dataset. The question was which result to choose when arguing about the actual ranking of the algorithms on datasets. We argued that we want to assess the maximal potential of some algorithm. The best result also usually corresponds with a solution found by a hyper-parameter optimizer. Therefore, we have used the result with the best Predictive Accuracy.

Finally, the results of ensemble algorithms like bagging, boosting, stacking and rotation forests [103] were omitted. These ensembles are encompassing different number of other machine algorithms and can take advantage of such combined power. Such composite behaviour resulted in heavily outperforming every other algorithm in the database, thus changing the results significantly. Furthermore, their performance relies heavily on the parameters that specify what algorithms should be used in the ensemble and as such, every parameter setting should be better treated as a separate algorithm. More careful examination is thus necessary before including ensemble algorithms into our experiments.

Application of the filters resulted in 351 datasets with 20,719 rows of the best results with Predictive Accuracy for some pair of algorithm and dataset. That included 115 unique algorithms.

These amounts may seem low for machine learning experiments, but they are still much bigger than in our previous experiments, and are very high also compared to the rest of the metalearning experiments found in the literature.

Unfortunately, not every algorithm was run over every dataset. Therefore, results of some algorithms did not have to appear in the neighbourhood considered by the  $k$ -NN algorithm. We decided to calculate the ranking only for the algorithms that were available in the neighbourhood. This could affect the quality of ranking evaluation, however, such negative consequences can be minimized by choosing high enough  $k$ , so the neighbourhood is more likely to contain more algorithms. Furthermore, high values returned by ranking quality evaluator still means high quality ranking, although not on every dataset. This can be seen as a noise or as missing values presented in the data.

The final question was splitting the data into the training and testing datasets. As the space was not very dense, and the  $k$ -NN that we were going to use is very dependent on the space density, we decided to split the datasets with 1:1 ratio. This enabled better estimation of the overfitting. If we used a different ratio, even the model with good generalization abilities could have much different results on the testing dataset, because of the space with much different density.

## 5.4 Metadata

This section will discuss which metadata were available or computed. First, we are going to discuss global data. Second, attribute metadata are described. This section also discusses normalization of both the global and attribute metadata. Note that different algorithms use different metadata, this will be elaborated when discussing experiment details.

### 5.4.1 Global Metadata

As discussed in the previous section, we have filtered the OpenML dump in such a way that every metadata is available for each dataset. We have argued whether we should include all the types of global metadata or just the simple, statistical, and information-theoretic ones. There are two reasons for this. First, we will base the attribute metafeatures on the simple, statistical, and information-theoretic types. Including the model based and landmarker features would make it harder to see whether the aggregation of attribute metafeatures indeed lose important information. Second reason is that model based and landmarkers metafeatures can contain information about performance on the the training and testing set. We would have to filter the training and testing set for such use to mitigate this risk. For these reasons, we have decided to use only the simple, statistical, and information-theoretic global metafeatures. In the production use, we would however include metadata of every type.

OpenML machine learning repository contained the following simple, statistic, and information theoretic global metadata:

- `ClassEntropy` – entropy of the target attribute.
- `DefaultAccuracy` – default accuracy obtained by the baseline algorithm. In this case by predicting always the majority class.

- Dimensionality – number of attributes divided by the number of instances.
- InstanceCount – number of instances (rows) of the dataset.
- MajorityClassSize – number of instances with the most frequent class of the target attribute.
- MaxNominalAttDistinctValues – the maximum number of distinct values among attributes of the nominal type.
- MinNominalAttDistinctValues – the minimal number of distinct values among attributes of the nominal type.
- MinorityClassSize – number of instances with the least frequent class of the target attribute.
- NumAttributes – number of attributes (columns) of the dataset.
- NumBinaryAtts – count of binary attributes.
- NumNominalAtts – count of nominal attributes.
- NumNumericAtts – count of categorical attributes.
- PercentageOfBinaryAtts – percentage of binary attributes.
- PercentageOfNominalAtts – percentage of nominal attributes.
- PercentageOfNumericAtts – percentage of numerical attributes.
- NumberOfClasses – number of classes for the classification task.
- IncompleteInstanceCount – number of instances with at least one value missing.
- NumMissingValues – number of missing values in the dataset.
- PercentageOfMissingValues – percentage of missing values.
- PositivePercentage – percentage of rows with the class with the highest assigned index. In the case of binary classification this is equal to percentage of positive instances.
- NegativePercentage – percentage of rows with the class with the lowest assigned index. In the case of binary classification this is equal to percentage of negative instances.
- MeanAttributeEntropy – mean of entropy among attributes.
- MeanKurtosisOfNumericAtts – mean kurtosis among attributes of the numeric type.
- MeanMeansOfNumericAtts – mean of means among attributes of the numeric type.

- `MeanMutualInformation` – mean of mutual information between the nominal attributes and the target attribute.
- `MeanNominalAttDistinctValues` – mean of number of distinct values among the attributes of the nominal type.
- `MeanSkewnessOfNumericAtts` – mean skewness among attributes of the numeric type.
- `MeanStdDevOfNumericAtts` – mean standard deviation of attributes of the numeric type.
- `NoiseToSignalRatio` – `ClassEntropy` divided by `MeanMutualInformation`. Returns -1 if `MeanMutualInformation` is zero.
- `EquivalentNumberOfAtts` – The difference of (`MeanAttributeEntropy` - `MeanMutualInformation`) divided by `MeanMutualInformation`. Returns -1 if `MeanMutualInformation` is zero.
- `StdvNominalAttDistinctValues` – standard deviation of the number of distinct values among nominal attributes.

We have also identified some duplicates among the metafeatures:

`NumberOfMissingValues` as a duplicate of `NumMissingValues`, `ClassCount` as a duplicate of `NumberOfClasses`, `NumberOfInstances` as a duplicate of `InstanceCount`, `NumberOfFeatures` as a duplicate of `NumAttributes`, `NumberOfNumericFeatures` as a duplicate of `NumNumericAtts`, `NumberOfSymbolicFeatures` as a duplicate of `NumNominalAtts` and `NumberOfInstancesWithMissingValues` as a duplicate of `IncompleteInstanceCount` metafeature. We have removed the duplicates out of a set of metafeatures used.

The last few attributes beginning with the `MeanAttributeEntropy` represent exactly those attributes where some important information may be lost during the aggregation, as discussed in Section 4.2.

The excluded metadata (of model based or landmarker type) are listed in Table 5.1. We will omit their description, as they are no longer relevant.

To sum up, we had 31 global metafeatures available for each dataset.

## 5.4.2 Attribute Metadata

There is no such public repository as OpenML that would have attribute metadata available for each dataset. Some information could be available in the header of the ARFF files (see Section 5.1), but as already discussed, there is no guarantee that the information really corresponds to values in the data. Indeed, we have encountered datasets that were corrupted this way. Furthermore, this information is optional, thus not available in every header. Based on these facts, we decided to build our own ARFF analyser. The analyser was able to read the input ARFF file and extract various attribute metadata. In the rest of this section, we describe the attribute metafeatures extracted. As our algorithms are capable of handling type-specific attribute metadata using selectors (see Algorithm 16), we will list them according to the type of attribute they were extracted for.

Table 5.1: Excluded global metafeatures.

RandomTreeDepth1AUC	REPTreeDepth3AUC	J480001ErrRate
RandomTreeDepth1ErrRate	REPTreeDepth3ErrRate	J48001ErrRate
RandomTreeDepth1Kappa	REPTreeDepth3Kappa	JRipErrRate
RandomTreeDepth2AUC	DecisionStumpAUC	NBTreeErrRate
RandomTreeDepth2ErrRate	DecisionStumpErrRate	SVMe2ErrRate
RandomTreeDepth2Kappa	DecisionStumpKappa	kNN1NErrRate
RandomTreeDepth3AUC	SimpleLogisticAUC	NBTreeAUC
RandomTreeDepth3ErrRate	SimpleLogisticErrRate	kNN2NErrRate
RandomTreeDepth3Kappa	SimpleLogisticKappa	J480001AUC
HoeffdingAdwinChanges	NaiveBayesAUC	JRipKappa
HoeffdingAdwinWarnings	NaiveBayesErrRate	SVMe3ErrRate
HoeffdingDDMChanges	NaiveBayesKappa	SVMe1AUC
HoeffdingDDMWarnings	SVMe1Kappa	SVMe2AUC
NaiveBayesAdwinChanges	SVMe2Kappa	SVMe3AUC
NaiveBayesAdwinWarnings	SVMe3Kappa	J48001AUC
NaiveBayesDdmChanges	kNN3NErrRate	kNN1NAUC
NaiveBayesDdmWarnings	kNN1NKappa	kNN2NAUC
REPTreeDepth1AUC	J480001Kappa	J48001Kappa
REPTreeDepth1ErrRate	NBTreeKappa	SVMe1ErrRate
REPTreeDepth1Kappa	J4800001AUC	J4800001ErrRate
REPTreeDepth2AUC	kNN2NKappa	kNN3NAUC
REPTreeDepth2ErrRate	JRipAUC	J4800001Kappa
REPTreeDepth2Kappa	kNN3NKappa	

In order to be able to compute some metafeatures for the categorical attributes, a conversion to the integers was made. Distinct number was assigned to each category. The number was chosen based on the order of appearance beginning with one.

Measures common for both the categorical and numerical metafeatures:

- ForRegression – whether the target was of numerical type (this was not used as we have used only classification tasks).
- ValuesCount – number of values.
- NonMissingValuesCount – number of non missing values.
- MissingValuesCount – number of missing values.
- Distinct – number of distinct values (classed).
- AverageClassCount – average count of occurrences among different classes.
- Entropy – entropy of the values.
- MostFrequentClassCount – count of the most probable class.
- LeastFrequentClassCount – count of the least probable class.
- ModeClassCount – mode of the number of distinct values.
- MedianClassCount – median of the number of distinct values.
- PearsonCorrelationCoefficient – Pearson Correlation Coefficient of the values and the values of target attribute.
- SpearmanCorrelationCoefficient – Spearman Correlation Coefficient of the values and the values of target attribute.
- CovarianceWithTarget – covariance of the values with the values of the target attribute.

Numerical metadata:

- IsUniform – whether statistical test did not reject that the attribute values corresponds to a uniform distribution.
- IntegersOnly – whether attribute values contained only integers.
- Min – minimal value of the attribute values.
- Max – maximal value of the attribute values.
- Kurtosis – kurtosis of the values.
- Mean – mean of the values.
- Skewness – skewness of the values.
- StandardDeviation – standard deviation of the values.

- Variance – variance of the values.
- Mode – mode of the values.
- Median – median of the values.
- ValueRange – difference between maximum and minimum of the values.
- LowerOuterFence – lower outer fence of the values.
- HigherOuterFence – higher outer fence of the values.
- LowerQuartile – lower quartile.
- HigherQuartile – higher quartile of the values.
- HigherConfidence – higher confidence interval of the values.
- LowerConfidence – lower confidence interval of the values.
- PositiveCount – number of positive values.
- NegativeCount – number of negative values.

Categorical metadata:

- Uniform Discrete – result of Pearson’s chi-squared test for discrete uniform distribution.
- $\chi^2$  Statistic – statistic value for the Pearson’s chi-squared test.
- Ratio of attribute values that after sub-setting the dataset to that attribute value lead to different distribution of the target as indicated by the following statistical test:
  - Kolmogorov-Smirnoff test (continuous target only),
  - Mann-Whitney U-test (continuous target only),
  - $\chi^2$ -test (categorical target only).

To sum up, we have extracted 15 types of attribute metadata available for both the numerical and categorical attributes, another 20 types for numerical attributes and 3 types for categorical attributes only.

### 5.4.3 Normalization

Our algorithms were designed with no prior distinction of the metafeatures. Some metafeatures put on much bigger values than the others. If we used the vector of constant weights, the  $p$ -norm distance could yield distance mainly derived out of big valued attributes. This would mitigate the influence of the attributes with small absolute values even though those could contain important information. To be precise, some metafeatures in our database have values as high as 445694751099523.38 (*variance* metafeature) and as low as -142020048 (*minimum*

metafeature), other metafeatures have by definition values constrained to some interval (for example (Spearman's Correlation Coefficient with the range of  $\langle -1, 1 \rangle$ ).

For this reason, we have normalized most metafeatures into the interval  $\langle 0, 1 \rangle$ . Attributes already naturally constrained to that or similar interval (Spearman's Correlation Coefficient) were the exception. The normalization was performed regardless of whether the metafeature was global or attribute specific. We have used the *min* – *max* normalization given by the following equation:

$$x'_i = \frac{x_i - \min_{x \in \mathbb{X}}}{\max_{x \in \mathbb{X}} - \min_{x \in \mathbb{X}}},$$

where  $\min_{x \in \mathbb{X}}$  and  $\max_{x \in \mathbb{X}}$  are minimal and maximal values of the given metafeature  $\mathbb{X}$ ,  $x_i$  is specific value of the  $i$ -th metafeature before rescaling, and  $x'_i$  the value of that metafeature after rescaling.

This solved the original problem, but sometimes another problem emerged. In some cases the majority of values lied in some small subinterval of  $\langle 0, 1 \rangle$ . Even when the metafeature had been assigned with high weight, the distance on this single metafeature would still be around zero between most of the metafeature values and very high between few outliers and the rest of the values. This would neglect the usefulness of such metafeatures including one that could be expected to bring high discriminative factor into the distinguishing of attributes, such as maximum and minimum values.

The box plots after min-max normalization are plotted for categorical metafeatures in Figure 5.3 and for the numerical attributes in Figure 5.4. The metafeatures suffering from such problem are those whose boxplot is small compared to the interval. In some cases, the boxplot quartiles are blending together and such cases are the most problematic ones. To partially mitigate the problem, we have introduced virtual metafeatures, that means metafeatures computed given different features that were independent of the values and size of the dataset - percentage or boolean values calculated out of metafeatures. Percentage or boolean are naturally normalized between  $\langle 0, 1 \rangle$  and should not have some outliers as in the case of metafeatures based on counts. The virtual metafeatures added were the following:

Measures common for both the categorical and numerical metafeatures:

- MissingValues – Boolean whether count of missing values is greater than 0.
- AveragePercentageOfClass – Percentage of the occurrences among classes, calculated by AverageClassCount/Values count.
- PercentageOfMissing – Percentage of missing values in the attribute, calculated by MissingValuesCount/Values count.
- PercentageOfNonMissing – Percentage of non missing values in the attribute, calculated by 1 - Percentage of missing.
- PercentageOfMostFrequentClass – Percentage of the most frequent class calculated as Most Frequent Class Count/Values count.
- PercentageOfLeastFrequentClass – Percentage of the least frequent class calculated as Least Frequent Class Count/Values count.

- ModeClassPercentage – Percentage of mode of class count calculated as Mode Frequent Class Count / Values count.
- MedianClassPercentage – Percentage of median of class count calculated as Median Frequent Class Count / Values count.

Numerical metafeatures:

- PositivePercentage – Percentage of positive values calculated as  $\frac{PositiveCount}{ValuesCount}$ .
- NegativePercentage – Percentage of negative values calculated as  $\frac{NegativeCount}{ValuesCount}$ .
- HasPositiveValues – Boolean whether attribute values contain positive values. Determined as the result of  $PositiveCount > 0$ .
- HasNegativeValues – Boolean whether attribute values contain negative values. Determined as the result of  $NegativeCount > 0$ .

Adding the virtual attributes does not mean that the rest of the problematic features should be thrown away. They could still be valuable for detecting outliers in the data, thus useful for identifying distant attributes (or datasets).

In total we have added 8 virtual metafeatures available regardless of attribute type and 4 virtual attributes for the numerical attributes. With the number of original attributes, it made 23 attributes for all types, 24 for numerical attributes, and additional 3 for categorical attributes.

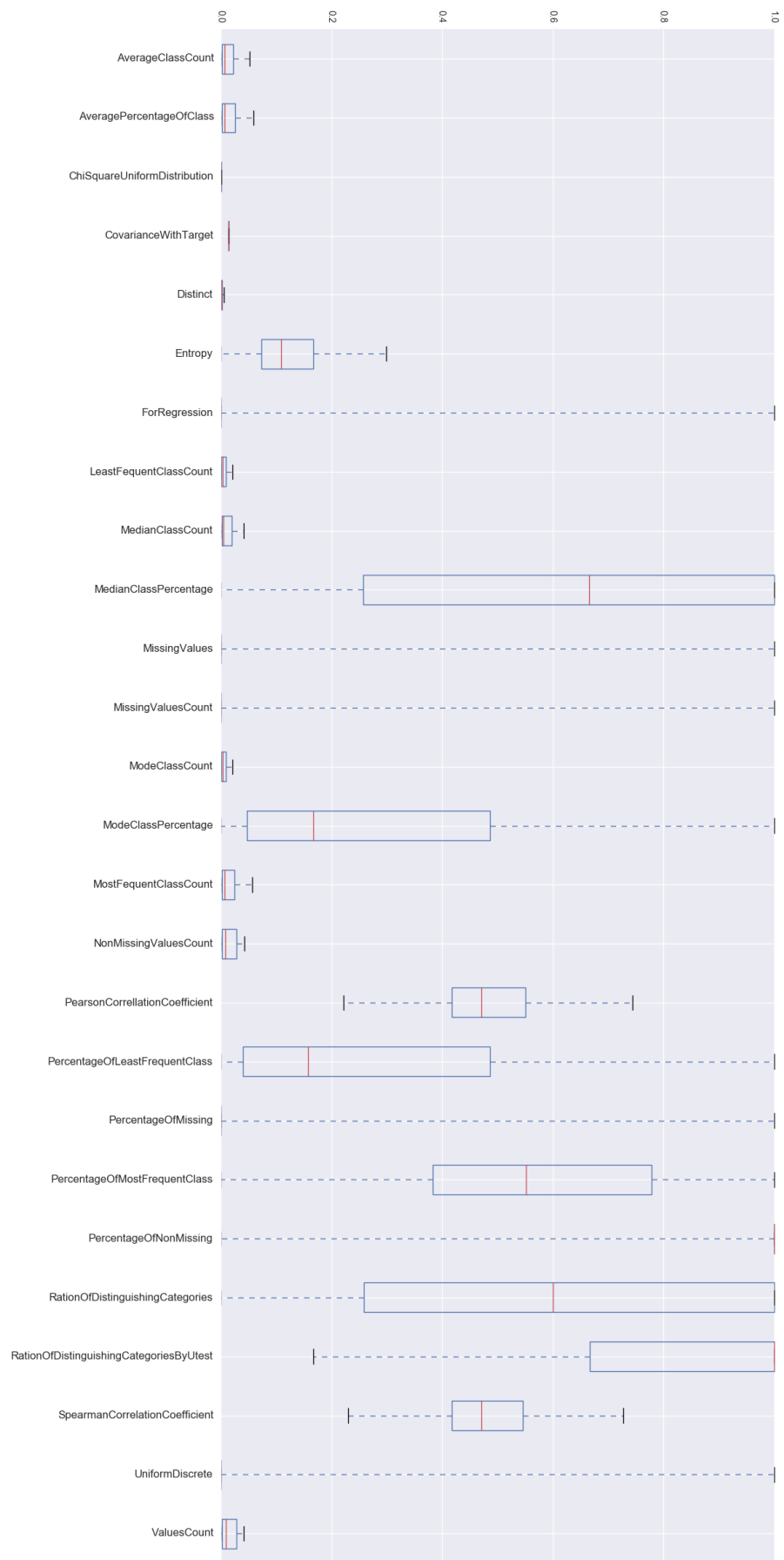


Figure 5.3: Distribution of values of categorical metafeatures after normalization.

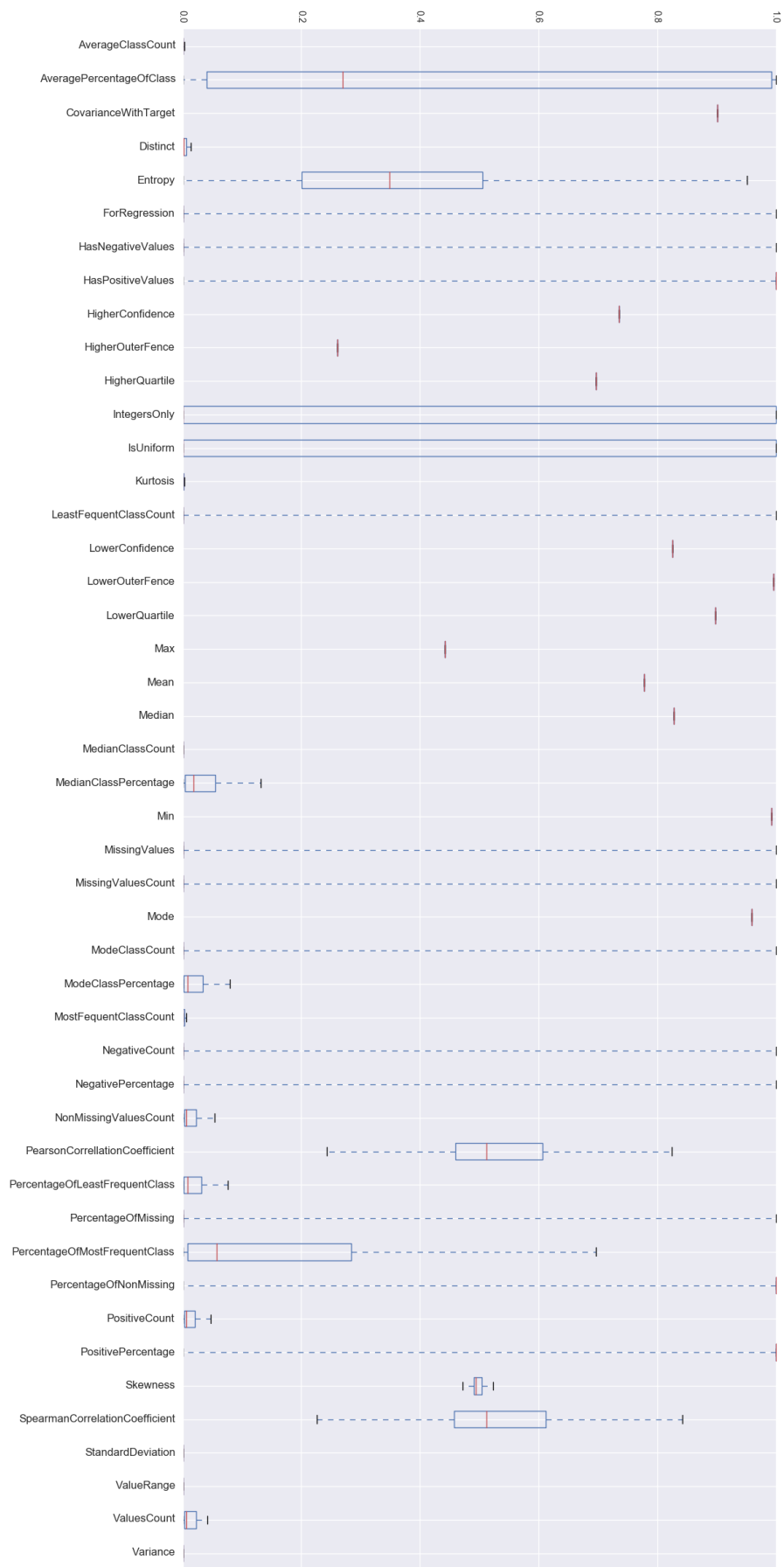


Figure 5.4: Distribution of values of numerical metafeatures after normalization.

# Chapter 6

## Experiment Proposal

In the previous chapters, we have discussed algorithms related to the algorithm ranking problem. We are able to rank algorithms based on distance and evaluate the quality of such ranking. As was discussed in Sections 2.5 and 2.6, we can easily plug distance based ranking into the quality evaluator. The remaining argument missing is to provide *IDatasetDistance* interface. We have discussed several algorithms being able to conform to this interface. Some of these algorithms need extra information as an input – weights. This enables us to optimize the weights in order to maximize the quality of ranking. We begin by defining the *optimization problem* in which some space of parameters is searched in order to get the optimal solution given some evaluation criteria. We then introduce the class of optimization algorithms called *Genetic Algorithms* (GAs) that are inspired by the evolution of species occurring in the nature. The general form of the algorithms is outlined and different parts are discussed – *selection*, *crossover* and *mutation* in more details.

We can use the GAs to optimize the weights of the algorithms for measuring the dataset distance. Metadata available was discussed in the previous chapter. Several experiments are proposed on the data available. Baseline, attribute alignment, and global attribute distance measure based on weighted  $p$ -norm with different values of  $p$  explored and weights optimized by the evolution. Further, attribute assignment with the attribute distance measure based on weighted  $p$ -norm with different values of  $p$  tried and weights optimized by the evolution. Final experiment proposed in this section is the aggregation of the global and attribute distance measures with the importance of both counterparts also optimized.

A big picture of all algorithms plugged together is provided. This will be possible because we have been careful in proposing algorithms in such a way that algorithms are easily pluggable to others, as they conform to the same interfaces. This overview is useful, as the number of algorithms plugged in together is large, and the whole workflow is complicated. We also discuss the choice of all other settings of all algorithms (extending attribute space by *dummy* attributes, parameters for the GA, etc.) The parameters will be chosen according to the results of Theorems 14, 15, and 16, so the resulting distance between datasets is a metric. We will review the results of our experiments and discuss them. As we will see, new proposed algorithms can outperform the baseline and even improve the global attribute distance with statistical significance.

## 6.1 Optimization

We have provided a means of measuring the quality of distance based ranking. Some of the distances are based on weighted  $p$ -norms. We could try to find out such weights that result in the best ranking according to the evaluator. This is captured by the following definition:

**Definition 41.** *In the optimization problem we have an objective function  $f$  over a domain (or search space)  $A$ :  $f : A \rightarrow \mathbb{R}$ . In the case of a minimization problem, the solution is  $x \in A$  such that  $\forall y \in A : f(x) \leq f(y)$ . The maximization solution is  $x \in A$  such that  $\forall y \in A : f(x) \geq f(y)$ .*

In some cases, it may be hard to find the solution of the optimization problem. Some algorithms do not guarantee the optimal solution and returns its approximation instead.

## 6.2 Genetic Algorithms

*Genetic algorithm* is a meta search heuristic based on Charles Darwin's evolution theory [28] and the laws of inheritance inferred from Gregor Mendel's inheritance theory [82]. According to the evolution theory, the following facts hold:

- Every species is fertile enough that if all offspring survived to reproduce the population would grow.
- Despite periodic fluctuations, populations remain roughly the same size.
- Resources such as food are limited and are relatively stable over time.
- Individuals in a population vary significantly from one another, and much of this variation is inheritable.

From these assumptions, the theory infers the following:

- A struggle for survival ensues.
- Individuals less suited to the environment are less likely to survive and less likely to reproduce; individuals more suited to the environment are more likely to survive and more likely to reproduce, and leave their inheritable traits to future generations, which produces the process of natural selection.
- This slowly effected process results in populations changing to adapt to their environments, and ultimately, these variations accumulate over time to form new species.

According to the inheritance theory, inherent properties of each organism are encoded in a structure called genotype. Genotype consists of genes. Each gene corresponds to some trait in organism (e.g. colour of eyes). Genotype is inferred from parents' genotype by crossing over their genetic material. Genotype may be also altered during organism lifetime by mutation. A phenotype is the composite of an organism's observable characteristics or traits, such as its morphology, development, biochemical or physiological properties, phenology, behaviour, and

products of behaviour. Relationship between genotype and phenotype is often conceptualized as follows:

$$genotype + environment \rightarrow phenotype.$$

Genetic algorithms (GA) were described by Holland [50], who utilized principles of the evolution and inheritance theory. Given an optimization problem, GA views the solution to the problem as an individual. In the original John Holland's work, the individual was binary encoded, but other encodings are suitable as well. The algorithm can also treat genotype equal as phenotype and the individual directly maps to the solution. In other situation, phenotype can be derived from the individual either deterministically or stochastically. Example of the former would be treating negative values of the individual as positive or encoding of the neural network, the example of the latter would be creating an individual according to the grammar described in the genotype with different rules applicable at the same time.

A number of individuals form a population. At first, a population of individuals is created (either randomly or by using some known sub-optimal solutions). Individuals are then evaluated based on their ability to solve the problem by a function known as fitness. Individuals proceed to next generation with probability proportional to their fitness (this step is known as *selection*). In each generation, new individuals are created from random parents in the current population (this step is known as *crossover*) and some individuals in the current population are altered (this step is known as *mutation*). New generations continue to be created until a termination criterion is satisfied (usually conditions on fitness of some individual, average fitness in population, number of generations or time elapsed since the start of the algorithm). A pseudocode of simple genetic algorithm is shown in Algorithm 19.

---

**Algorithm 19:** Genetic Algorithm

---

// Pseudocode of the main loop of the genetic algorithm.

```

1  $P_0 \leftarrow \text{initialize-population}();$ 
2  $t \leftarrow 1;$ 
3 while termination-criterion not satisfied do
4    $P_{t+1} \leftarrow \text{selection}(P_t);$ 
5    $\text{crossover}(P_{t+1});$ 
6    $\text{mutation}(P_{t+1});$ 
7    $t \leftarrow t + 1;$ 
8 end
9 return bestIndividual( $P_t$ );
```

---

Since genetic algorithms are stochastic and do not guarantee finding an optimal solution, it could be sometimes beneficial to repeat the whole process and take the best individual from all runs. Mutation, crossover and other steps altering the population are often generalized as genetic operators. Genetic operators will be described when applied to one or several individuals; expansion of the operators on the whole population is typically done by applying the operator on each individual or on a sample of individuals from the population.

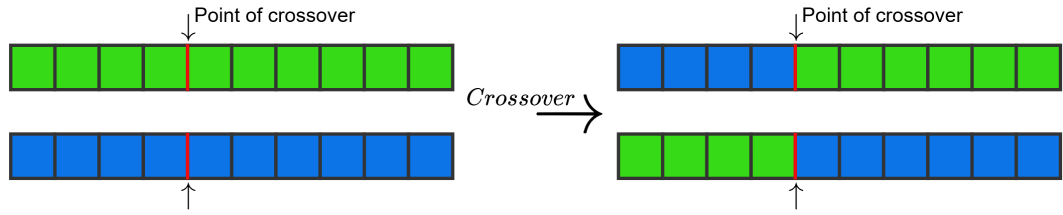


Figure 6.1: Example of the crossover genetic operator.

### 6.2.1 Selection

Selection determines how many offspring the individuals will have in the next generation. This should be based on fitness - in general, fitter individuals should have more offspring than those less fit. Common approaches to the selection are:

- *Roulette selection* – let  $f_k$  be a fitness of an individual  $k$ . Let  $F_s$  be the sum of the fitness of all individuals ( $F_s = \sum_{i=1}^n f_i$ , where  $n$  is the population size). For each position in the next generation, the roulette is spun. In each spin an individual is selected with probability  $\frac{f_k}{F_s}$ .
- *Scaling* – same as the roulette selection, except that fitness is scaled at the beginning. The most common scaling function is linear function. This can solve some problems in case all individuals have similar fitness (more like random walk) or when there are very large fitness gaps between individuals (high pressure on selecting best individuals).
- *Rank based* – individuals are sorted by fitness in ascending order. Probability of selection is higher with higher index in the sorted set of individuals.
- *Tournament selection* – for each position in next generation, a tournament of  $n$  individuals is held. The best individual is selected by the tournament with some fixed probability  $p$ . If the best individual is not selected, the second best individual is selected with probability  $p$  and so on. If all previous individuals are not selected, select the worst individual from the tournament. This rescales the population and thus the evolution pressure remains constant (extraordinary good individuals does not flood entire population and even the minor differences between individuals are recognized).

Some alternations of GAs perform the selection not only among the new offspring, but also among parents. Furthermore, the best solution found so far may be lost during the process. To counter this, the best individuals are sometimes guaranteed to be inserted into the next generation. This is referred to as *elitism*.

In binary coding, crossover is typically implemented as a one point crossover – two parents are selected, then one point in both parents is chosen randomly, and the parts induced by the point chosen are swapped. This creates two offspring individuals. A more general alternative is the  $n$ -point crossover, where more points are chosen when creating the offspring. One point crossover is illustrated in Figure 6.1.

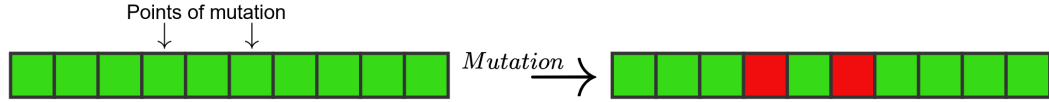


Figure 6.2: Example of the mutation genetic operator.

### 6.2.2 Mutation

Mutation operator alters a part of an individual, thus introducing new features into population. Mutation helps explore those parts of the search space that would be otherwise hard to reach with selection and crossover only. In particular, mutation helps the genetic algorithm get out of the local optima. In binary coding, mutation is typically implemented in a way that each bit has some small probability  $p_{mutation}$  of being flipped. This is illustrated in Figure 6.2.

### 6.2.3 Modifications

One of the advantages of genetic algorithms is their versatility. It is easy to change different parts of the algorithm to accustom for the needs. For example if we have some knowledge about the problem we aim to solve, we may introduce operators tailored to the problem. Such operators often achieve better results than classic non-specialized operators. Alternatively, we can create additional evolutionary pressure if the evaluation of the fitness is costly (either time or money) so the algorithm converges quickly. Other approaches include controlling the parameters, so the algorithm does not get stuck in local optima and the diversity of individuals in the population is empowered.

## 6.3 Experiments

We have all the pieces we need to propose different experiments settings. We have different ranking measures, different distances measures that can be optimized, optimization algorithm, data, metadata, and we are also able to evaluate the quality of different ranking algorithms. In this chapter, we finally connect these components together. Since we have designed the system interfaces in a versatile way, we will be able to plug in different components into algorithms, thus changing the behaviour in a certain way. This will enable us to use the same framework for evaluation although the settings used may be totally different.

When proposing the full framework for experiments, we will use the top-down approach. This is because the algorithm on the top will be either the same or will not change so often and will be providing the unified experiment framework. The experiment part will be mainly carried in the bottom of the framework where we will be trying different settings and combination. The rest of this section is dedicated to the description of our experiments.

The goal of the experiments is to find a good settings for solving the ranking problem. We will use the OpenML dump discussed in Chapter 5 as the data source. The quality of settings will be verified by the Ranking Quality Assessment algorithm (Algorithm 6). The Ranking Quality Assessment needs an implementation of *IRanking* interface as an input. According to the discussion

in Section 2.7, the baseline algorithm is useful in order to verify that the extra complexity present in different models has indeed improved the ranking quality. Therefore, as the first setting to try, we will add the baseline algorithm built by Algorithm 7. This algorithm does not need any additional information other than data, thus we are finished with this experiment branch. Different implementation of *IRanking* interface discussed in the thesis were distance based rankings. One suitable algorithm was  $k$ -NN algorithm (Algorithm 5).

Other distance based algorithms were also discussed but we were inclined to use the simpler algorithm, as the whole framework is quite large and we did not want to add an extra layer of logic or complexity without a specific reason. The transformation of *IDistanceRanking* to *IRanking* is done by the *partial* application as described in Algorithm 4. The  $k$ -NN algorithm needs, besides data, a parameter  $k$  and a notion of distance between datasets, as captured by the *IDatasetDistance* interface (Algorithm 1). As discussed in Section 5.3, the size of the neighbourhood should be set high enough so it is probable that the neighbourhood will contain all algorithms. We have set the  $k$  to 17 (10 percent of the training datasets).

Throughout the thesis, we have discussed many options how to implement the *IDatasetDistance*. In this chapter, our aim will be to set up all the algorithms conforming to the *IDatasetDistance* interface so the resulting distance is a metric. We will explore the non-metric settings in the chapters to follow. In Section 3.2 we reviewed distance based on global attributes. As we have shown, any distance defined on global metafeatures based on the weighted  $p$ -norm is a metric. The corresponding *IDatasetDistance* implementation was outlined in the Global Metadata Distance algorithm (Algorithm 8). We will try the most common types of  $p$ -norms – that is 1-norm, 2-norm and  $\infty$ -norm. We let the Genetic Algorithm optimize the weights of the  $p$ -norms. The settings and the genetics operators used for optimizing the weights will be discussed later.

Other implementations of the *IDatasetDistance* interface were proposed in Chapter 4. These algorithms were able to deal with the unstructured dataset space by attribute assignment. We will discuss them starting with the simpler ones. Algorithm 9 needs just the mapping of attribute into a single number –  $\sigma$ . We have decided to use selectors for this approach. Concrete mapping for attributes were number of categories in the case of categorical attributes and difference between maximum and minimum in the case of the numerical attributes. We could also try different attribute evaluation functions  $\sigma$  but regarding the time complexity to conduct the experiments, we decided to focus computational power on more expressive languages. Furthermore, Algorithm 9 is a special case of Algorithm 16, and it is sufficient to try the more generic version to assess the potential of attribute assignment techniques. Other attribute assignment algorithms were Attribute Assignment algorithm (Algorithm 12) and the more generalised version the Combined Attribute Assignment algorithm (Algorithm 16) that split the assignments into multiple assignments given by the selectors.

As discussed in Section 4.3, we found it more sensible to calculate assignments of numerical and categorical attributes separately and not to mix numerical and categorical attributes together. The latter would limit the number of metafeatures that could be used. Furthermore, we argued that the distance between a categorical and a numerical attribute should be naturally high – possibly reached

by the penalization for addition of the *dummy* attributes to each selector. We thus decided not to use the Attribute Assignment algorithm but rather to use its more generic version – Algorithm 16. We used two selectors – one selecting categorical attributes and the other selecting the numerical attributes (Algorithms 15 and 14). According to Corollary 4, the Combined Attribute Assignment algorithm produces a metric if the attribute distance measures of corresponding selectors are metric on the attribute space. According to this, we have to define categorical and numerical attribute distance as a metric. Again, we will use the most common weighted  $p$ -norms – 1-norm, 2-norm and  $\infty$ -norm. We shall set the  $p$  of the norm in the same manner and we will not try one value of  $p$  for the categorical distance and a different value for the numerical distance. However, we let weights to be set independently (also, the number of metafeatures is different). We will also optimize the weight of each assignment result. Again, we will use Genetic Algorithm for the optimization with the settings discussed later. Finally, we will carry out experiments with the aggregation of global and attribute metafeatures (Algorithm 18 – Dataset Distance Aggregation). We will use Algorithms 8 and 16 as sub-distances. The selectors used will be the same as in the rest of the experiments. Again, we will use the same value of  $p$  for each algorithm and selector. We let GA optimize all weights occurring in the algorithm including weight of each sub-distance.

We have prepared a cluster of computers to conduct the experiments with various hardware and operating system. The system was composed of MySQL database for storing the results, local SQLite database with metadata and experiment results that were distributed together with the application. As this database was large and was needed to be retrieved before each experiment, we decided to add to the application to reduce network load with multiple runs and to allow for faster queries to the database. The experiments run on the Microsoft .Net platform using C# programming language. This is a platform that is currently supported by Windows, although there is .Net runtime called Mono for Unix-like system that has limited capability compared to the whole framework. However, we made sure to use the part of .Net that can be run by Mono, therefore we could use all the major OS platform - Windows, Unix and Mac OS. To allow for really easy redistribution, the Docker Image [83] was created. Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. This means that every docker image should be deployable to every computer that has docker installed. The image also has all the requirements installed like installed shared libraries and packages. This allows for smooth running of the experiments just by running the image. The image is automatically downloaded from the docker repository after entering the run command.

Before we discuss the concrete settings for the experiments, we would like to recapitulate the complexities of the whole workflow and underlying algorithms.

If the ranking is precomputed, the computation of ranking quality has a complexity of  $\mathcal{O}(n_d n_a)$  where  $n_d$  is the number of datasets and  $n_a$  is the number of algorithms. The complexity of ranking is constant for the baseline (whereas building up the baseline function takes  $\mathcal{O}(n_d n_a + n_a \log(n_a))$  time). For the distance based ranking, the  $k$ -NN algorithm takes  $\mathcal{O}(n_d \log(n_d) + n_a \log(n_a))$  time

per dataset, provided we have a distance matrix precomputed. This results in  $\mathcal{O}(n_d(n_d \log(n_d) + n_a \log(n_a)))$  steps for all datasets. We will conduct the distance computation outside to save time as discussed in Section 2.5. To compute the distance between every pair of dataset we need  $\mathcal{O}(n_d^2 c(\Delta))$ , where  $c(\Delta)$  is the cost of computing the distance  $\Delta$  between some pair of datasets. The complexity of computing the distance  $\Delta$  using the global metadata for some parameter  $p$  is  $\mathcal{O}(n_m)$ , where  $n_m$  is the size of the vector of global metadata. The missing complexity is computing  $\Delta$  using the assignment techniques. The selectors we use merely check whether attribute is numerical or categorical, therefore the complexity of each selector is  $\mathcal{O}(n_{att})$ .

The Attribute Alignment (Algorithm 9) in our case has the complexity of  $\mathcal{O}(n_{att} \log(n_{att}))$ , where  $n_{att}$  is the bound of number of attributes in datasets (given by the selector). This is because computing the evaluation function  $\sigma$  is easy if the  $\sigma$  represents number of categories or difference between maximum and minimum. The complexity of Attribute Assignment with selectors (Algorithm 16) is  $\mathcal{O}(c(\delta)n_{att}^2 + n_{att}^3)$ , where  $c(\delta)$  is the cost of attribute distance function. In our case the  $c(\delta)$  is  $\mathcal{O}(n_{att\_met})$  – the size of the vector of attribute metadata. The  $c(\delta)n_{att}^2$  part is for computing the distance matrix between two sets of attributes and the rest is for the Hungarian method. Finally, the complexity of Dataset Distance Aggregation (Algorithm 18) is the sum of complexities of individual sub-distances.

The exact values of different variables influencing the complexities are shown in Table 6.1.

Table 6.1: Values of variables influencing the complexity given by the training data.

Variable	Description	Value
$n_d$	Number of attributes	170
$n_a$	Number of algorithms	115
$n_m$	Number of global metafeatures	31
$n_{att}$	Bound of number of attributes	50
$n_{att\_met}$	Number of attribute metafeatures	47

The total complexity for the whole workflow for different setups of the ranking algorithms are in Table 6.2.

When optimizing the weights of either attribute distance or dataset distance, the complexity provided is the complexity to evaluate one individual. It is interesting that given one weight, we have to evaluate the weighted  $p$ -norm  $\mathcal{O}(n_d^2(n_{att}^2))$  times to build up a distance matrix. As the weights are all that compose the individual in this case, we can look at our task as a reinforcement learning task, as we get a feedback from environment after  $\mathcal{O}(n_d^2(n_{att\_met}n_{att}^2))$  steps.

Apparently, workflows including attribute assignments are the most costly ones. Just expression  $n_d^2(n_{att\_met}n_{att}^2 + n_{att}^3)$  in our case is equal to  $170^2(47 + 50^2 + 50^3) \approx 2 * 170^2 50^3 = 7,225,000,000$ . Note that this is not the exact amount of steps taken as the expression is in  $\mathcal{O}$ , it merely gives the idea about the complexity of the workflow for our data. It takes up to three days to compute the ranking quality of the Attribute Assignment algorithm optimized by evolution with 100 individuals and 100 generations on the Intel I7 computer with sufficient amount

Table 6.2: Total complexity of the whole workflow for ranking quality evaluation for different ranking algorithms.

Algorithm	Total Complexity (in $\mathcal{O}$ )	Optimizing
Baseline	$n_d n_a + n_d n_a + n_a \log(n_a)$	No
Attribute Alignment	$n_d n_a + n_d(n_d \log(n_d) + n_a \log(n_a))$ $+ n_d^2 n_{att} \log(n_{att})$	No
Global Distance	$n_d n_a + n_d(n_d \log(n_d) + n_a \log(n_a))$ $+ n_d^2 n_m$	Yes
Attribute Assignment	$n_d n_a + n_d(n_d \log(n_d) + n_a \log(n_a))$ $+ n_d^2(n_{att\_met} n_{att}^2 + n_{att}^3)$	Yes
Aggregation	$n_d n_a + n_d(n_d \log(n_d) + n_a \log(n_a))$ $+ n_d^2(n_{att\_met} n_{att}^2 + n_{att}^3 + n_m)$	Yes

of memory. This was a value that had to be taken into account when proposing the settings for assignment algorithms.

In algorithms with attribute assignments, the decision about how to extend attribute space with *dummy* has to be made. We did not want to penalize too heavily for having different number of attributes. As we want to have a metric and such small penalization is not possible for artificial *dummy<sub>n</sub>* attributes by Theorem 15, we decided to use *dummy<sub>a</sub>* attribute from the attribute space. Based on the same argument with small penalizations, we created *dummy<sub>a</sub>* attribute that has the value of every metafeature equal to the median value of all values of that metafeature in both the training and testing set.

The settings for the evolution was set according to our previous experiments and few short preliminary experiments we performed to estimate good values. We were not able to tune the parameters because of the computation times mentioned above. We decided to use tournament selection with elitism to preserve the best values. The tournament probability of better individual winning was set to 60 percent and the tournament size was set to three. This was to discourage earlier convergence and to boost the generalization ability. We have used crossover with the probability of 75 percent and the mutation with the probability of 10 percent. Population size was set to 100 in order to have reasonably big population and still be able to finish the computation in reasonable time. The number of generation was set as a termination criterion. The exact value was set to 70. This was again chosen based on the time of computation, while allowing for some sufficient amount of evolution cycles to evolve interesting properties. Furthermore, we did not want to have too many generations so the algorithm does not have a big opportunity to overfit the found solutions. The weights were randomly initialized out of  $\langle 0, 1 \rangle$  uniform distribution. However some operators could push the weight values outside of this interval – even to zero or negative values. According to Theorem 8, the weights must be strictly positive to have a metric. Therefore we use genotype to phenotype mapping by using the absolute value of the weights. We still allow weights to be zero. This would break the coincidence metric axiom but we rather interpret it as the algorithm decided that

the corresponding metafeature is a noise, and therefore the attribute space should not contain this metafeature.

### 6.3.1 Results

The results of experiments proposed in this chapter are described in the following paragraphs. Our framework uses Algorithm 6 to measure average Spearman’s rank correlation coefficient (Equation 2.7) on the training and testing data. The higher value the better, whereas value 1 is a theoretical maximum, value -1 is a theoretical minimum, and value 0 is as good as a random guessing.

The baseline scored 0.551659 on the training set and 0.540807 on the testing set. Attribute Alignment algorithm had both testing and training score below 0.5. We believe this was because we did not optimize this algorithm. We will not discuss this algorithm further, as it is a special version of the Attribute Assignment algorithm.

The raw results of all metric-producing experiments on the testing set can be seen in Table 6.3. As the runs did not have any order by definition, we sorted the values so that the first run of every algorithm corresponds to the best result of that algorithm and so on.

From the raw results it can be seen that some algorithms are better than others. From the point of view of the median of the results, the order of algorithms is as follows (from best to worst): (Aggregation with  $p = 1$ ), (Aggregation with  $p = 2$ ), (Global with  $p=\infty$ ), (Global with  $p = 1$ ), (Global with  $p = 2$ ), (Aggregation with  $p = \infty$ ), (Assignment with  $p = 2$ ), (Assignment with  $p = \infty$ ), (Assignment with  $p = 1$ ), baseline.

To have a proper comparison, results of each algorithm on the testing set were compared based on the result of two tailed *Mann-Whitney U Test* [25]. This test is a non-parametric test with the null hypothesis that both samples come from the same population. The  $\alpha$  statistics used was 0.05. The results of the tests are in Table 6.4. All algorithms except the Assignment algorithm with the  $p = 1$  were significantly better than the baseline. The best results had the aggregation of assignment and global metadata. There was no algorithm that would be significantly better than any of the Aggregated algorithms. On the contrary, Aggregated algorithms with  $p = 1$  and  $p = 2$  were significantly better than every other algorithm. There was no clear winner between those two. The results proof our hypothesis that algorithms based on attribute assignment produce useful results for ranking prediction. Also, our theory that the best results are probably obtainable by the aggregation of assignment and global distance is also supported by the results. Interesting observation is that the assignment with  $p = 1$  produces the worst results but the aggregation with the same  $p$  provides the best results. Perhaps alignment with  $p = 1$  provide best additional information to support decision by the global metadata.

The best results on the testing set were produced by the aggregation of global and attribute metafeatures where the  $p$  was set to 1. The evolution progress is shown together with the baseline in Figure 6.4. We have measured evaluation score of the currently best individual only for statistics purpose and such measurement did not influence the run of the algorithm. It is however useful to see whether some overfitting occurred. As can be seen, the baseline was surpassed



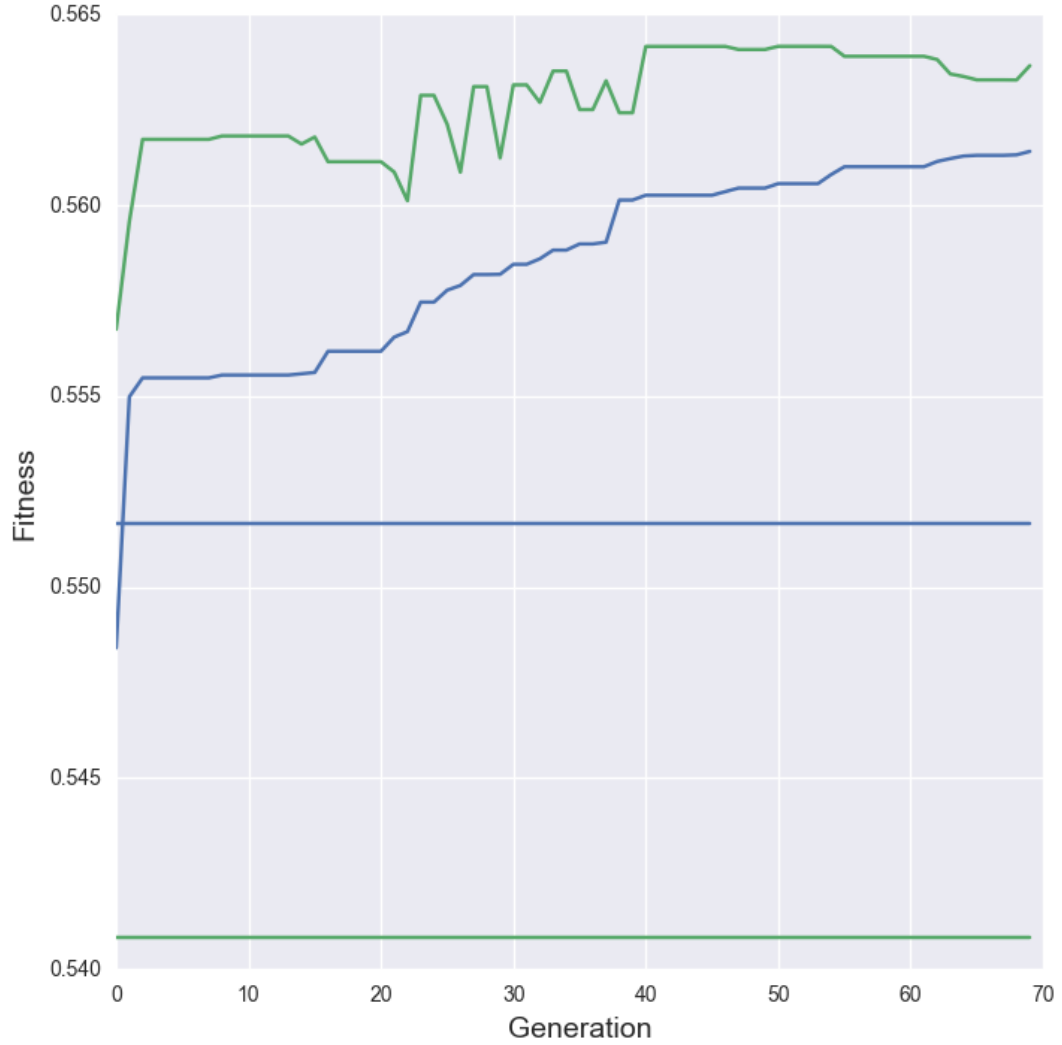


Figure 6.4: Evolution progress of the individual with the best result on the testing set together with the baseline. Results on the training set are blue, on the testing set are green. On the  $x$ -scale is the evolution progress, on the  $y$ -scale is the quality of ranking measured by the Spearman’s correlation coefficient.

after the first generation on the training set and was above the testing baseline at all times. We can also see very high correlation between the improvement on the training set with the improvement on the testing set. At some points, some decrease of performance on the testing set can be seen but is compensated for in the further generations.

### 6.3.2 Visualisation of the Distance

The visualization can be important for humans to interpret the results. This can be possible using kernel methods described in Section 4.1.2. Using kernelized PCA, we can project the space into two dimensions, which can be easily visualized. This requires inner product kernel. However, algorithms we use are producing metrics instead of kernels. There is a connection between these two concepts as metric represents dissimilarities and kernels similarities. We can transform distance to similarity by subtracting the distance from some large enough constant.

This may not be an inner product kernel, however lots of kernel methods perform well enough with similarities close to inner products [125]. Other approach is to repair the resulting similarity by one of the techniques proposed in [37]. We have used kernelized PCA by taking the distance evolved during the run of aggregation of global and attribute approach with the best result on the testing set. The distance to similarity was transformed as follows:

$$\text{similarity}(x, y) = 100 - \Delta(x, y).$$

The visualization of the training set is depicted in Figure 6.5. We can see few clusters, but the space is mostly well covered. To check the plausibility of the visualization, we investigated the cluster of datasets in the top right region defined by  $x$ -component  $> 1.3$  and  $y$ -component  $> 0.2$ . The datasets are listed in Table 6.5 together with several selected metafeatures. Every dataset in the cluster is a binary classification tasks with very similar Default Accuracy. Therefore, the visualization seems to be plausible. Being able to visualize the data is important, as it allows for a much easier interpretation.

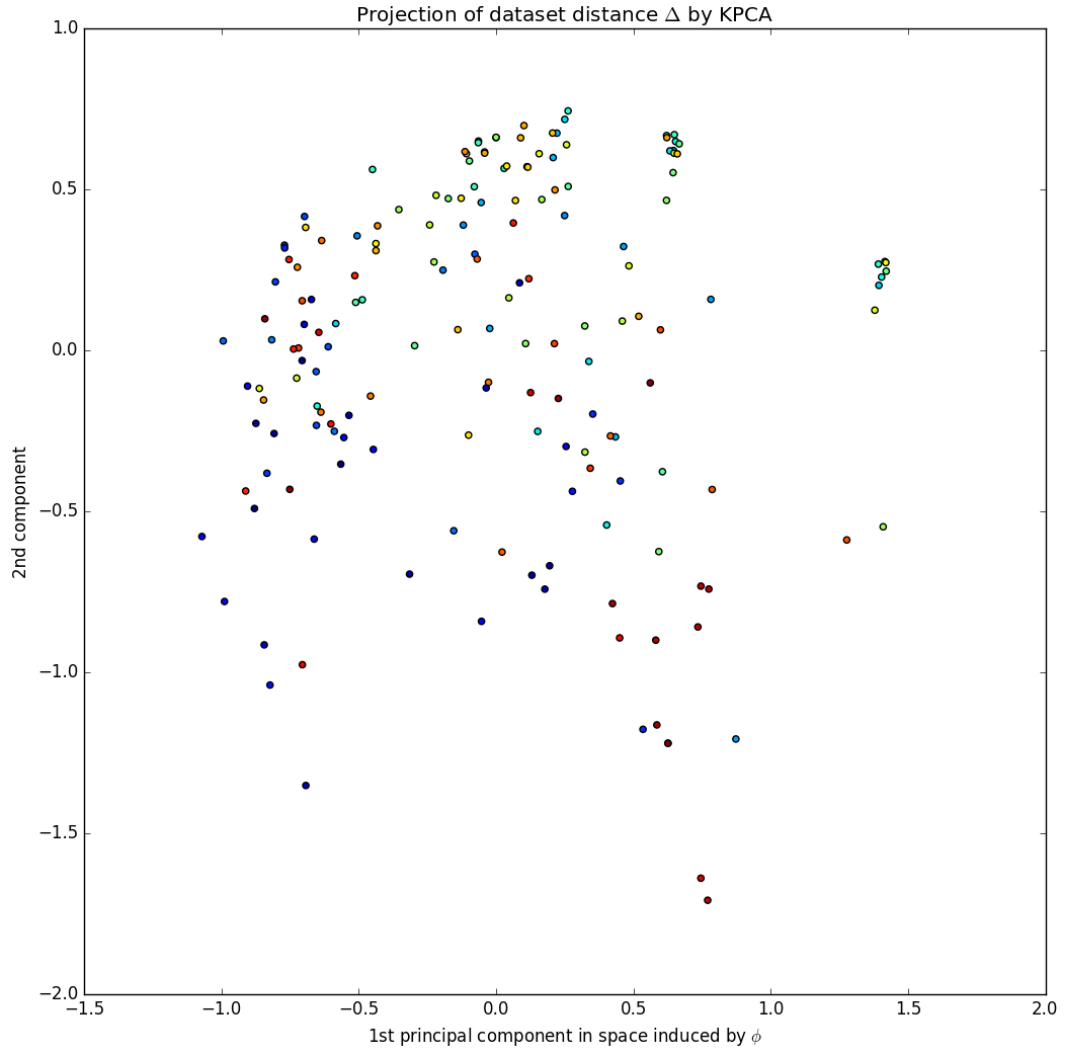


Figure 6.5: Projection of the training datasets using kernelized PCA and similarity based on the aggregation of global and attribute distance with the best result on the testing set.

Table 6.3: Evaluation of ranking quality of metric algorithms on the testing dataset.

Run	$p = 1$		
	Global	Assignment	Aggregation
1	0.556977	0.5495	0.563649
2	0.556756	0.54561	0.563345
3	0.556341	0.545248	0.563046
4	0.556282	0.543699	0.56146
5	0.555896	0.542449	0.560571
6	0.55554	0.540697	0.559257
7	0.555343	0.540422	0.558868
8	0.555299	0.540328	0.558767
9	0.555105	0.54024	0.556617
10	0.554622	0.53886	0.556128
Median	0.555718	0.541573	0.559914

Run	$p = 2$		
	Global	Assignment	Aggregation
1	0.554876	0.553777	0.563032
2	0.554851	0.547558	0.560352
3	0.554324	0.547426	0.560273
4	0.554175	0.547255	0.558913
5	0.554089	0.54587	0.55852
6	0.553832	0.544282	0.557389
7	0.553628	0.543237	0.556919
8	0.553458	0.543211	0.556625
9	0.553423	0.542909	0.556509
10	0.553093	0.542443	0.554418
Median	0.553961	0.545076	0.557955

Run	$p = \infty$		
	Global	Assignment	Aggregation
1	0.557636	0.549426	0.560815
2	0.55736	0.546354	0.560662
3	0.557084	0.544777	0.556537
4	0.557026	0.544307	0.552372
5	0.556069	0.543048	0.551713
6	0.55585	0.542675	0.551215
7	0.555551	0.541701	0.55055
8	0.555429	0.540912	0.550265
9	0.555058	0.540544	0.545764
10	0.554629	0.537932	0.545444
Median	0.555959	0.542861	0.551464

Table 6.4: Statistical comparison of different algorithms and their ranking quality results on the testing set. Row  $i$  defines what algorithms had significantly worse results than algorithm  $i$ . N stands for no and Y for yes.

	Aggregation, $p = 1$	Aggregation, $p = 2$	Global, $p = \infty$	Global, $p = 1$	Global, $p = 2$	Aggregation, $p = \infty$	Assignment, $p = 2$	Assignment, $p = \infty$	Baseline
Aggregation, $p = 1$	N	Y	Y	Y	Y	Y	Y	Y	Y
Aggregation, $p = 2$		Y	Y	Y	Y	Y	Y	Y	Y
Global, $p = \infty$			N	Y	N	Y	Y	Y	Y
Global, $p = 1$				Y	N	Y	Y	Y	Y
Global, $p = 2$					N	Y	Y	Y	Y
Aggregation, $p = \infty$						Y	Y	Y	Y
Assignment, $p = 2$							N	N	Y
Assignment, $p = \infty$								N	Y
Assignment, $p = 1$									N

Table 6.5: Datasets in one of the clusters in the visualization of the distance (Figure 6.5).

Name	OpenMLId	Attributes	Classes	DefaultAccuracy	Rows
fri_c0_250_5	776	6	2	0.5	250
fri_c3_250_5	744	6	2	0.564	250
no2	886	8	2	0.502	500
wind	847	15	2	0.532552	6574
pbc	810	19	2	0.550239	418
analcatadata_apnea3	764	4	2	0.535715	450

# Chapter 7

## Metric Relaxation

In the previous chapters, the means of algorithm ranking have been discussed. Lots of effort has been dedicated to design algorithms that produce metrics on the dataset space induced by the attribute assignment. These algorithms were derived from the weighted  $p$ -norms on the attribute space. This enabled us to optimize the weights using evolutionary algorithms. We have also proven the effectivity of such approach. On the other hand, we could argue that more complex distance measure can be build on the attribute space. For instance, it seems like a good idea to enable the model to ask questions whether the attribute contain only integers or also continuous values. But instead of weighting difference of zeroes and ones, it would be perhaps a better idea to split the distance computation based on these value. For such elaborate decision, a more expressive language will be required than just a fixed-size vector of real numbers. With such added complexity we can however lose the guarantee of the attribute metric. We begin this chapter of defining relaxed versions of metrics. *Semimetric* and *quasimetric* are derived from metric by omitting one of the axioms – triangle inequality and symmetry axiom respectively. We discuss which metric axioms are more important when defining a distance measure. We present a simple way of modifying arbitrary distance measure to be at least a semimetric. Then we revisit the theorems about preservations of metric from the space of attribute to the space of datasets when using attribute alignment, and we will show that the same facts hold for the preservation of semimetric. We then define a class of algorithms derived from Genetic Algorithms (see Section 6.2) called *Genetic Programming* (GP). Genetic Programming, compared to the Genetic Algorithms, tries to evolve a function (or program) solving the problem instead of finding a solution to an instance of problem. Genetic Programming algorithms can be configured to produce the functions of arbitrary expression power, which will be suitable for finding more delicate attribute distance measures. We also show examples of typical GP operators. We review one of the phenomenon sometimes observable in the GP experiments – the *bloat problem*. Bloat is the tendency of some GP functions to grow in complexity, which is often accompanied by the decrease in generalization abilities of the evolved programs. In the rest of the section, we propose new batch of experiments. We discuss their expressive abilities and set their parameters in alignment with our theoretical results. We review the results of the new algorithms compared to the previous ones.

## 7.1 Metric Spaces Revisited

In the definition of the metric (Definition 13), we have opted to use a redundant definition as the axiom 1 can be derived from the remaining axioms (Theorem 1). This enables us however to relax the definition of the metric to introduce the terms semimetric and quasimetric:

**Definition 42.** A semimetric on a set  $X$  is a function satisfying the first three axioms but not necessarily the fourth (triangle inequality).

$$d : X \times X \rightarrow [0, \infty), \quad (7.1)$$

and for all  $x, y, z$  in  $X$ , the following conditions are satisfied:

1.  $d(x, y) \geq 0$  (non-negativity).
2.  $d(x, y) = 0 \Leftrightarrow x = y$  (coincidence axiom).
3.  $d(x, y) = d(y, x)$  (symmetry).

**Definition 43.** A quasimetric on a set  $X$  is a function satisfying all axioms except symmetry.

$$d : X \times X \rightarrow [0, \infty), \quad (7.2)$$

and for all  $x, y, z$  in  $X$ , the following conditions are satisfied:

1.  $d(x, y) \geq 0$  (non-negativity).
2.  $d(x, y) = 0 \Leftrightarrow x = y$  (coincidence axiom).
3.  $d(x, z) \leq d(x, y) + d(y, z)$  (triangle inequality).

In both quasimetrics and semimetrics, it is no longer possible to derive non-negativity axiom using the rest of the axioms. This is the reason why we explicitly included the first axiom into the metric definition.

There has been a long debate in the community whether all these properties are equally important for the similarity function in metalearning. For example, there are arguments against the triangle inequality [10], a popular example is as follows: “a man is similar to a centaur, the centaur is similar to a horse, but the man is completely dissimilar to the horse”. This example clearly violates the triangle inequality and speaks against using it. Moreover, it seems the coincidence axiom is also not very important. We can for example imagine, that the algorithm decides, that some of the metadata are not important for the similarity, thus returning zero even in cases where these values are different. We have addressed this by treating the attributes with the weight of zero as a noise. Regarding the symmetry axioms, there are some examples from the real world that are against it. For example, if we would go from place A to place B, the distance could be different from going back as we could go uphill, some paths could be blocked in one direction, etc. However, in the case of datasets, this example is not very useful because distance in the sense of dataset rather means dissimilarity than actual distance. When comparing two objects, we do not mind whether the comparison is made in a different order. Therefore, we did not consider quasimetrics further in the thesis.

Given an arbitrary distance measure  $d$  on some space, we can repair the measure in such a way that it is a semimetric. For example function  $d'$  defined as follows:

$$f(x, y) = \frac{|f'(x, y)| + |f'(y, x)|}{2} \quad (7.3)$$

is always non-negative and symmetric. To enforce coincidence axiom, we can easily detect equal arguments on the input and return 0 otherwise. Similarly, if the  $d'$  would return 0 for a non-matching input we could return some small non-negative number  $\phi$  instead. Or we could return  $\epsilon + d(x, y)$ , for small  $\epsilon > 0$ . Both approaches results in semimetric, however there is a distinction. The former approach could break triangle inequality in the case  $d$  was a metric. If we found three objects  $k, l, z$  such that  $d(k, z) < \phi/2$ ,  $d(z, l) < \phi/2$ , we could break the triangle inequality. That would happen if the  $\phi$  was returned for the tuple  $k, l$  instead of 0. For the triangle inequality to hold, it must be the case that the distance  $\phi = d(k, l) \leq d(k, z) + d(z, l)$ . But at the same time, we have  $d(k, z) + d(z, l) < \frac{\phi}{2} + \frac{\phi}{2} = \phi$  from the assumptions, which is a contradiction. We cannot do the same argument for the latter approach, as  $\epsilon$  was amended to all distances of non equal objects.

### 7.1.1 Attribute Assignment with Relaxed Attribute Measure

In Section 4.5, we have discussed how Algorithm 12 preserves the metric properties. It turns out that Theorems 14, 17 and Observation 4 are valid also for a semimetric:

**Corollary 6.** *Let  $a, b$  are lists of attributes,  $\delta$  is a semimetric on the space of attributes  $\mathbb{A}$ . Then Algorithm 12 preserves all semimetric axioms and the resulting distance  $\Delta$  on the dataset space  $\mathbb{D}$  is a semimetric.*

*Proof.* By following the proof of Theorem 14, as the proof does not use derivation of the first metric axioms from the others (Theorem 1), and proof of each axiom uses only the corresponding axiom on the attribute distance to prove the preservation without the use of the rest of the axioms.  $\square$

**Corollary 7.** *Let  $\mathbb{A}$  space of attributes and  $A$  its subset,  $\delta$  is a semimetric on  $A$ ,  $\mathbb{D}$  space of datasets. Then  $\forall D \subseteq \mathbb{D}$ ,  $E$  supported by  $A$ , Algorithm 12 preserves all semimetric axioms and resulting distance  $\Delta$  on  $\mathbb{E}$  is a semimetric.*

*Proof.* By following the proof of Corollary 6 as the proof does not require elements outside of  $D$  and  $A$  and we can replace the whole  $\mathbb{D}$  by  $D$  and the whole  $\mathbb{A}$  by  $A$  respectively.  $\square$

**Observation 7.** *Let  $\mathbb{A}$  be space of attributes and  $\mathbb{D}$  space of datasets. Let  $D$  be a subset of  $\mathbb{D}$  and  $\Delta$  a semimetric on  $D$ . Let  $A$  be the source of  $D$  and  $\delta$  be a distance measure on  $A$ , such that Algorithm 12 induces  $\Delta$  using  $\delta$ . Then  $\delta$  is not necessarily a semimetric on  $A$ .*

*Proof.* By following the proof of Observation 4, where we have shown that even though dataset distance  $\Delta$  is a metric (thus consequently semimetric), we can define attribute distance  $\delta$  to violate non-negativity.  $\square$

We can draw the similar conclusions as with metric spaces. If we have a choice of optimizing between semimetric on attribute level and semimetric on dataset level, choosing the former implies optimizing the latter. This is not valid in the opposite direction.

## 7.2 Genetic Programming

To create more elaborate distance measures, more expressive language than the one used in the previous sections is required. We will also need a tool that can search in the language space and can find expressions that give a good similarity for the ranking prediction. *Genetic programming* (GP) can provide the needed functionality.

Genetic programming is based on the same idea as genetic algorithms, where the encoding is often linear and of a fixed-length, but the search space is different. Genetic algorithms search the space of possible solutions to the problem, while genetic programming algorithms search the space of functions or programs (hopefully able to solve the problem) instead. There are many ways of representing a program. Three common ways of representation used in genetic programming are [95]:

- *Tree representation:* tree representation is traditional in genetic programming, and we will also use this representation in this thesis. Inner nodes of the tree represent operators (number of successors of the node equals arity of the operator) and leaves represent operands. Tree structures can be easily evaluated and are easy to interpret. Genetic operators are also easy to implement, as we will see later in this chapter. Usual assumption of tree representation is the property called *closure* [95]. It says that every inner node should handle arbitrary input from other operators and operands. The closure is usually obtained by using auto-conversion and/or defining operands of the same type and furthermore, every operator takes input of that type and return the output of that type. This is called type consistency. Finally, the faulty values for some operator can be handled by returning a default value for faulty input, extending the domain of that operator or decrease the fitness if the exception is thrown.
- *Linear representation:* programs are represented as a sequence of some programming language. As we are interested in more in the actual functions taking two attribute metafeatures and not in elaborate functions taking lists or other more complex structures, Linear representation will not be considered further in this thesis.
- *Strongly typed Genetic Programming:* proposed in [88]. It removes the closure requirements by defining types. This is done by introducing some restrictions into the process. We gain more freedom in defining functions. Furthermore, it can reduce the search space, as the algorithm can avoid generating some invalid input.

The search space is determined by a language  $L$  consisting of two sets – the *function set* (having arity greater than zero) and the *terminal set* (having zero arity). Terminals are either constants or input variables, and they occur only in the leaves of the tree representing the program. Functions are aggregating other functions and terminals, and they occur only in the inner nodes of the tree. The choice of  $L$  is very important. Program solving the problem have to be encodable in this language - on the other hand, a too complex language will increase the search space exponentially, thus making finding a sufficiently good program nearly impossible. Genetic programming was used successfully in many domains.

### 7.2.1 Initialization

At the beginning of the GP run, each individual in the initial population has to be randomly initialized. This can be achieved using following methods:

- *Full* method: This method receives an integer specifying the depth of a new individual as an input. This method creates layers sequentially. If the depth of the layer is lower than the target depth, new nodes are created by using functions, otherwise only terminals are used. This method creates a full tree of the target depth.
- *Grow* method: This method takes an integer specifying maximum depth as an input. New layers are created by using both functions and terminals at random. If maximum depth were to be violated, only terminals are used. By allowing terminals in the inner nodes, the distance between the root node and lists may be less than the specified maximum depth.
- *Ramped half-and-half*: This method creates half of the new individuals by using the full method and the other half by using the grow method.

When generating subtrees, we have to worry only about arity and generate required number of arguments. Because of the closure property we can use arbitrary inputs for every operator.

### 7.2.2 Crossover

The principle of the crossover operator is the same as in the original genetic algorithms. Given two parents, one node from each parent is randomly selected. Subtrees corresponding to these nodes are swapped afterwards. This is a valid operation because of the closure property. The whole process is illustrated in Figure 7.1. Crossover points does not have to be selected with uniform probability. Typical GP primitive sets lead to trees with an average branching factor (the number of children of each node) of at least two, therefore the majority of the nodes will be leaves. Consequently, the uniform selection of crossover points leads to crossover operations frequently exchanging only very small amounts of genetic material (i.e., small subtrees); many crossovers may in fact reduce to simply swapping two leaves. To counter this, authors in [69] suggested the widely used approach of choosing functions 90% of the time and leaves nodes 10% of the time.

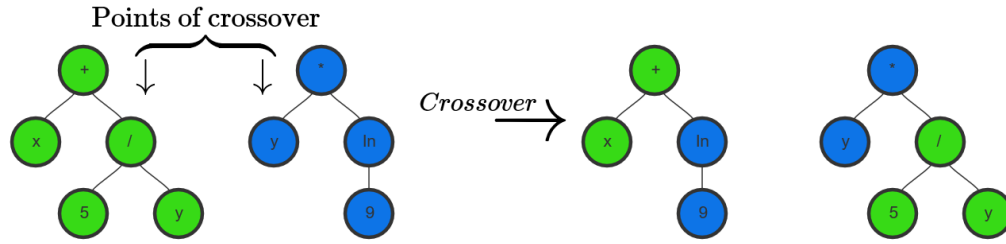


Figure 7.1: Example of the crossover in the Genetic Programming with the tree representation.

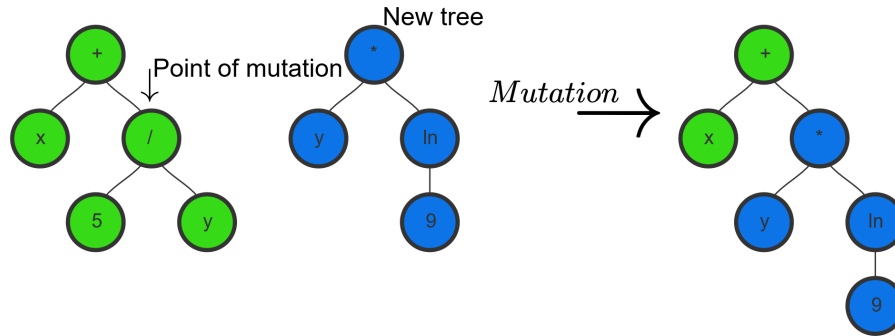


Figure 7.2: Example of the mutation in the Genetic Programming with the tree representation.

### 7.2.3 Mutation

Mutation alters part of the tree. A node in the parent is randomly selected. A random tree is initialized by one of the initialization method and the selected node is replaced by the new tree. Again, the closure property guarantees this to be a valid operation. The whole process is illustrated in Figure 7.2.

### 7.2.4 Bloat Problem

*Bloat* refers to a rapid growth of individual sizes without corresponding significant increase of fitness in later generations. In general, software bloat means that a computer program contains features that are never or rarely used. Growth alone could be beneficial, after all we are often searching complex program spaces, but without the fitness improvement it is nearly always bad. Larger individuals take more time to evaluate, take more space to store, are harder to interpret, and their ability to generalize is greatly reduced. There are three main theories explaining bloat [95]:

1. *Replication accuracy theory* states that the success of a GP individual depends on its ability to have offspring that are functionally similar to the parent. As a consequence, GP evolves towards (bloated) representations that increase replication accuracy.
2. *Removal bias theory* divides nodes in a GP tree into two categories – active code and inactive code. Inactive code is either not executed, or it is executed and its output is then discarded (for example, an inactive code would be a

subtree consisting of  $+(0 + 0 + 0 + 0)$ ). All remaining code is considered active. The theory observes that inactive code in a GP tree tends to be low in the tree, residing, therefore, in smaller-than-average-size subtrees. Crossover events excising inactive subtrees produce offspring with the same fitness as their parents. On average, the inserted subtree is bigger than the excised one, thus such offspring are bigger than average while retaining the fitness of their parent leading ultimately to growth in the average program size.

3. The *nature of the program search spaces theory* predicts that above a certain size, the distribution of fitness does not vary with size. Since there are more long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Over a time GP samples longer and longer programs simply because there are more of them.

Techniques were designed to prevent or decrease bloat. More comprehensive survey is in [79], we will list some of the approaches:

1. Size and depth limits. This approach checks after applying genetic operator whether the offspring is beyond the size or depth limit. If it is not, the offspring enters the population. If, instead, the offspring exceeds the limit, one of the parents is returned. Obviously, this implementation does not allow programs to grow too large. However, there is a serious problem with this way of applying size limits, or more generally, constraints to programs: parent programs that are more likely to violate a constraint will tend to be copied (unaltered) more often than programs that do not. That is, the population will tend to be filled up with programs that nearly infringe the constraint, which is typically not what is desired. The problem can be fixed by not returning parents if the offspring violates a constraint. This can be realized using two different strategies. Firstly, we can just return the oversized offspring, and assign it a fitness of 0, so that the selection will get rid of it in the next generation. Secondly, we can simply declare the genetic operation failed, and try again. This can be done in two alternative ways: a) the same parent or parents are used again, but new mutation or crossover points are randomly chosen (which can be done up to a certain number of times before giving up on those parents), or b) new parents are selected and the genetic operation is attempted again.
2. Anti-Bloat genetic operators. This approach modifies genetic operators to reduce the bloat. Among the bloat-control methods are size fair crossover and size fair mutation [72]. These work by constraining the choices made during the execution of a genetic operation so as to actively prevent growth. In size-fair crossover, for example, the crossover point in the first parent is selected randomly, as in standard crossover. Then the size of the subtree to be excised is calculated. This is used to constrain the choice of the second crossover point so as to guarantee that the subtree chosen from the second parent will not be “unfairly” big.
3. Anti-Bloat selection modifies the selection so that bloated individuals have lower probability to be selected into next generation. Tarpeian method

[94] controls bloat by acting directly on the selection probabilities in the following equation:

$$E[\mu(t+1) - \mu(t)] = \sum_t l(p(l,t) - \phi(l,t)), \quad (7.4)$$

where  $E$  is the expectation operator,  $\mu(t+1)$  is the mean size of the programs in the population at generation  $t+1$ ,  $l$  is the program size,  $p(l,t)$  is the probability of selecting programs of size  $l$  from the population in generation  $t$  and  $\phi(l,t)$  is the proportion of programs of size  $l$  in generation  $t$ . This is done by setting the fitness of randomly chosen longer-than-average programs to 0. This prevents them from being parents. By changing how frequently this is done, the anti-bloat intensity of Tarpeian control can be modulated. An advantage of the method is that the programs whose fitness is zeroed are never executed, thereby speeding up runs. Parsimony pressure method [69] changes the selection probabilities by subtracting a value based on the size of each program from its fitness. Clearly, bigger programs have lower fitness and potentially less offspring under this approach. That is, the new fitness function is:  $f_{new}(x) = f(x) - cl(x)$ , where  $l(x)$  is the size of program  $x$ ,  $f(x)$  is its original fitness and  $c$  is a constant known as the parsimony coefficient.

### 7.3 Experiment Proposal

We want to extend our whole framework with more expressive attribute distance. Again, we would like to get attribute distance in the form that can fit into our workflow. In the metric experiments we decided to use attribute alignment with selectors, as we wanted to utilize even metadata specific numerical and categorical attributes. We will do the same decision again. That gives us more specific idea for what we are looking for – we need two functions, first computing distance between two numerical attributes and the second computing distance between two categorical attributes. In the previous section, we argued that we will use tree representation for evolved functions. To get there, we need to specify functions, terminals, make sure that we maintain the closure property, decide on the initialization and genetic operators. The main thing we need to keep in mind when proposing the whole design of the GP algorithm is that we want to generate more expressive functions. However, we will make one exception. The algorithm from Section 6.3 also evolves the weights of each selector. To evolve them using GP we would have to combine GP with GA as the vector of weights is not a tree. We decided to use the same weights for the numerical and categorical selector instead of evolving them.

If not said otherwise, the proposed functions and terminals can be used regardless of the type of program evolved. All functions and terminals will be type consistent, which means that all functions will have all arguments and output of the same type. This type will be a real number in our case. If some  $n$ -ary function is not defined on the whole  $\mathbb{R}^n$ , we will propose its extended definition on the whole  $\mathbb{R}^n$ .

### 7.3.1 Functions

When discussing which function to use, we have argued that if we want to create trees with more expressive power than the previous experiments with the metric based on  $p$ -norms. To do that we should start by enabling the same functionality – to add function for addition, division, square root (as we were using only  $p \in \{1, 2, \infty\}$ ), maximum of two values (required by the infinity norm),  $p$ -th power and abs. We should initialize the function set of the GP accordingly.

- Basic mathematical functions: add, subtract, multiply and divide will be used. Only division needs to be generalized. The following generalization was chosen:

$$\text{Divide}(x, y) = \begin{cases} 0; & \text{if } y = 0, \\ \frac{x}{y}; & \text{otherwise.} \end{cases}$$

Based on the discussion above, we also included maximum. We decided not to include power of  $p$  explicitly, as for  $p = 1$  or  $p = 2$  it can be easily evolved by the times function.

- Boolean functions: normally, boolean function returns true and false, which is usually used for branching further in the program. This would however break the closure. For the sake of type consistency, we used a trick to design Boolean functions and we proposed boolean functions (with some arity  $i$ ) according to the following pattern:

$$\text{Boolean}(a_1, \dots, a_n, x, y) = \begin{cases} x; & \text{if } b(a_1, \dots, a_n), \\ y; & \text{otherwise.} \end{cases}$$

Namely:

$$\text{LessThan}(a_1, a_2, x, y) = \begin{cases} x; & \text{if } a_1 < a_2, \\ y; & \text{otherwise.} \end{cases}$$

$$\text{LessThanOrEqual}(a_1, a_2, x, y) = \begin{cases} x; & \text{if } a_1 \leq a_2, \\ y; & \text{otherwise.} \end{cases}$$

In theory, maximum function can be obtained by these boolean functions. However, it is quite complicated to evolve as  $\max(x, y)$  corresponds to  $\text{LessThan}(x, y, y, x)$  and all four inputs must match. As the maximum function is quite important, we decided to add the maximum as an extra function nevertheless.

- Other functions: we have introduced square root and base 2 logarithm. These functions were generalized by following:

$$\text{SquareRoot}(x) = \begin{cases} \sqrt{x}; & \text{if } x \geq 0, \\ \sqrt{|x|}; & \text{otherwise.} \end{cases}$$

$$\text{Log}_2(x) = \begin{cases} \log_2(x); & \text{if } x \geq 0, \\ 0; & \text{if } x = 0, \\ \log_2(|x|); & \text{otherwise.} \end{cases}$$

Some other functions were also discussed:

- Polynomial functions: We did not want to expand the domain too much, and this type of functions can be expressed by combining basic functions, so we did not introduce polynomials into population.
- Periodic functions: like sin, cos. We have argued that an evolving program will not benefit from periodicity, thus we did not introduce such functions into GP domain.
- Boolean functions greater than, greater than or equal. These were not introduced into the domain because they can be expressed by the means of Boolean functions already in the domain:

$$\text{GreaterThan}(a_1, a_2, x, y) = \text{LessThanOrEqual}(a_2, a_1, x, y),$$

$$\text{GreaterThanOrEqual}(a_1, a_2, x, y) = \text{LessThan}(a_2, a_1, x, y).$$

### 7.3.2 Terminals

1. Constant terminals: we have proposed terminals that represent real and integer numbers. When creating such a terminal, a random number is generated and set as a value of the new terminal.
2. Metadata Terminals: again, we wanted to make the GP at least expressively strong enough to be able to evolve the same functions created in the previous section. In order to do this, we should cover all the attribute metadata used in the metric experiments. By the nature of metadata, some of them will be available only for the categorical trees and some for the numerical trees being evolved. Every metadata will be initialized with either one or zero. As the distance tree computes the distance function of two datasets  $a$  and  $b$ , the zero or one will define whether the value of the terminal variable should be taken from the metadata of dataset  $a$  or  $b$ .

Random number terminals were also discussed. We have argued that an evolved program would not benefit from stochasticity, therefore we did not introduce such terminals into domain. Note that this is different from generating constants, because random number terminals generate a new number each time they are evaluated.

The example with the individual generated for the numerical distance is in Figure 7.3.

### 7.3.3 Algorithm Specification

We need to evolve two trees. We are going to evolve these two trees as one individual. The mutation and crossover will be first applied on the categorical trees and then also on the numerical trees. As the initialization methods for the evolution, we will use the ramped half-and-half to initialize every tree. Ramped half-and-half was chosen because according to some authors [95], it creates more diversity. The initial maximal depth was set to 6. The mutation and crossover



probabilities were set regardless of whether they are used for the categorical or numerical part of an individual. The mutation chance was set according to our previous experiments to 0.2. The probability of crossover happening was set to 0.7. The termination criterion was set to the generation count. We did not want to encourage bloating and over-fitting, so the generation target was set to 80. We believe this was the reasonable amount of generation to evolve reasonably good distance measures. Compared to the previous experiments with genetic algorithm, we also increased the population size to 120 individuals. GP is more dependent on bigger populations, as a lot of distance measures generated really bad input compared to the genetic algorithms where everything was a  $p$ -norm, and even the worst weights produced somewhat reasonable output.

For the rest of the workflow settings, we will not make any changes. We set up number of neighbours for the  $k$ -NN to 17 and use the dummy attribute as the one already in the attribute space given by the median of every metafeature.

Given two datasets  $a$  and  $b$ , respectively their numerical and categorical metafeatures, the evolved tree can now compute the distance using Algorithm 16. This does not guarantee any of the metric properties. For instance, algorithm can produce zero easily by instantiating the minus node with two children – each of them initiated by a constant of the same value. It would be very hard to constrain the GP algorithm to evolve only metrics or semimetrics. It would require either a limited set of functions the algorithm can use, or complicated operators, which would ensure these properties. Instead, based on the discussion in Section 7.1, we decided to amend the values produced by the trees according to Equation 7.3, return 0 if  $x = y$  and add a small  $\epsilon > 0$  if  $\delta(x, y) = 0$  and  $x \neq y$ . This will guarantee that we will have a semimetric on the attribute space. According to Corollary 6, the resulting distance  $\Delta$  on the dataset space is a semimetric. Thus, we sacrificed triangle inequality to gain more expressive language to describe attribute distance measures. Other option would be to repair the resulting distance between datasets. We have decided to repair the attribute distance, as aligned to the results of Corollary 7 and Observation 7.

The increase in the population size slightly increased the amount of time to conduct the experiments. Also, if the GP tree was deep enough, its evaluation usually took slightly more time compared to  $p$ -norms. This was a major factor as the function was evaluated many times. Despite this fact, we decided not to lower the number of runs so the results are easier to compare. Therefore, the number of runs was again set to 10.

The whole workflow with the GP and the semimetric repairment is shown in Figure 7.4.

### 7.3.4 Results

The results are shown in Table 7.1. The GP managed to get above the baseline on the validation set in three out of 10 cases. It is not surprising that the statistical comparison of the GP and other algorithms shown in Table 7.2 resulted in all algorithms except the baseline outperforming the GP.

These poor results could be explained by overfitting, as the training results were decent enough. For example, bloating occurred – in the first generation the average number of nodes was around one hundred after the initialization. We



Figure 7.4: UML diagram of the workflow for the GP experiments. This time the focus will be on the GP tree, which is going to be evolved using the genetic programming by the fitness from the Ranking Quality Evaluator.

Table 7.1: Evaluation of the ranking quality of the trees produced by the GP algorithm on the testing dataset.

Run	GP Result
1	0.546221
2	0.542101
3	0.541881
4	0.540339
5	0.538439
6	0.536444
7	0.536215
8	0.532274
9	0.530289
10	0.512501
median	0.537441

could observe values above one thousand in the generation 70. This is not a rare behaviour of algorithms with high expression capabilities, as they can very easily learn some noise present in the training data. In the next chapter, we will try to improve generalization abilities of the trees being evolved.

Table 7.2: Statistical comparison of GP and previous algorithms and their ranking quality results on the validation set. Y stands for GP significantly worse than algorithm in the corresponding column, N stands for Not able to reject the null hypothesis (algorithms equally performing).

	<div> Aggregation, <math>p = 1</math>  Aggregation, <math>p = 2</math>  Global, <math>p = \text{inf}</math>  Global, <math>p = 1</math>  Global, <math>p = 2</math>  Aggregation, <math>p = \infty</math>  Assignment, <math>p = 2</math>  Assignment, <math>p = \infty</math>  Assignment, <math>p = 1</math>  Baseline </div>									
GP	Y	Y	Y	Y	Y	Y	Y	Y	Y	N

# Chapter 8

## Regularization

In the previous chapter, we relaxed the metric assumptions a little bit, and we proposed a genetic programming algorithm to evolve trees measuring attribute distance that fit into our workflow. In the experiments, we observed very poor results of the algorithm ranking model produced by the GP caused by overfitting. In this chapter, we will focus on improving the generalization abilities of the GP algorithm. There are many ways how to do that. We will review some of them. The so called *bootstrapping* modifies the initialization phase. Some individuals with already interesting fitness are inserted into populations and their useful blocks may be distributed over the population. *Regularization* [4] is a set of techniques aiming for boosting the generalization abilities of machine learning model. This is done by penalizing complex hypothesis or by encouraging the properties that we think help in generalization. In this chapter, we will introduce two regularization techniques that we believe could help in improving our models – both of them have been already used in our experiments in [114, 111]. The first approach uses technique called *coevolution* during the evolution of the trees. The second approach – called *multi-objectivization* – splits the single objective into multiple objectives in which we can measure some other interesting properties. To do that, we will need a *multi-objective optimization* algorithm. We will present *NSGA-II* algorithm as it is one of the best for two-objective optimization. We will propose new batch of experiments combining genetic programming algorithm from the previous chapter with some of the techniques discussed in this chapter. We will again compare their results with the rest of the algorithms. We also review some of our previous experiments using multi-objectivization for the algorithm ranking problem.

### 8.1 GP Modifications

In this section, we will review two techniques of modifying the GP algorithm – bootstrapping and coevolution. Both can be used to improve the generalization abilities of the GP algorithms.

#### 8.1.1 Bootstrapping

The *bootstrap* problem may occur in complex domains. When the population is initialized, all individuals often have very low fitness. This happens especially

when the ratio of good to bad solutions is very small. It is then hard for the genetic programming algorithm to estimate good places to evaluate and the run of the algorithm is similar to the random walk algorithm.

One approach to deal with the bootstrap problem is proposed in [47]. The problem and/or domain is simplified, so there is a better chance that some good individuals are generated during initialization. After sufficient solutions are found for the simplified problem/domain, we increase the difficulty of the problem but let the population as it is. We expect that the solutions for simplified problems will not have very low fitness for the more difficult problem (as would probably happen with random initialization). The whole process is repeated until the more difficult problem is equal to our original problem. This approach is called the *incremental evolution*.

The second approach arises from the research about initializing the population [33]. If the initial population to the GA is good, then the algorithm has a better possibility of finding a good solution [21], [128] to seed the GP with that information [22], i.e., the initial population is seeded with some of those possible solutions or partial solutions of the problem. It can be easily combined with some other algorithms. Let other algorithms find some possible or partial solution and pass these solution for the GP initialization.

The bootstrapping can be also used to boost the generalization abilities of the GP. We can insert individuals with good generalization abilities with the expectation that useful blocks of information will be distributed over the population, changed in then novel ways while still maintaining the generalization abilities of the original blocks.

### 8.1.2 Coevolution

In some cases, it may be beneficial to evolve different part of an individual separately. For example, the GP algorithm presented in the previous chapter simultaneously evolves two tree - one for measuring the distance between numerical attributes, second for evolving distance between categorical attributes. Instead of thinking about this as individuals consisting of pair of trees, we could think of this as two species - categorical and numerical one. The fitness of an individual would be based on a cooperation of this individual with one or more individuals of the other species. In this case, for one tree for measuring distance between numerical attributes, in each generation, we would choose one or more categorical trees and evaluate ranking quality of this tuple. This has the negative effect that the fitness has to be reset after each generation as the fitness of an individual can change, as the fitness is dependent on the population of other species, which increase the computation time as we cannot pass the fitness of unchanged individuals to the next generation. This does not concern us too much, as this increase is not in the order of magnitudes. The major benefit is that the individual does not have time to overfit, as the stable part is needed for overfitting. Since each generation connects different representations of each species together, only those properties that are generally useful are usually kept in the individuals. More information about coevolution can be found in [123, 57, 98, 97].

## 8.2 Multi-objectivization

Even if some problem at hand is in fact single-objective (we try to minimize the error rate of the algorithm), it can be sometimes also expressed as a problem with more objectives. Such an approach is called *multi-objectivization* and it has been shown that it can improve the performance of single-objective optimization algorithms, especially in cases where the optimized function contains plateaus [20]. Pilát and Neruda [93] used multi-objectivization for the hyper-parameter tuning of classifiers. They added two objectives to guide the search – the root mean squared error and the kappa statistic – while they tried to optimize the hyper-parameters for the best accuracy of the model. In the field of machine learning, multi-objective optimization can also be used for regularization [55]. In such case, the regularizing term is added as another objective rather than summing it with the optimized criterion.

Throughout this thesis, we discussed many attribute distance measures. We began with attribute distance measures that were metric, in Chapter 7 we relaxed this a little bit and discussed semimetrics. We can use the multi-objectivization to add an extra objective to the original one – resulting ranking quality. The second criterion will be the similarity of a distance measure to a metric. We have two choices for which measure we would like to use. We have an attribute and dataset distance measures. As our training set covers only a limited number of datasets and attributes out of dataset and attribute space, Theorem 17 and Observation 4 will be useful. According to those, the optimization towards a metric on the training datasets does not optimize metric on the attributes in the training dataset, but the opposite is true. In that sense, the optimization towards metric on the attribute space is somewhat stronger.

As the multi-objective optimization is more complex than single-objective optimization, we will devote some space to a brief introduction. We will formally define a multi-objective optimization problem, discuss Pareto dominance and the first Pareto front. We also introduce NSGA-II algorithm and discuss why it is suitable for our needs.

### 8.2.1 Multi-objective Optimization

**Definition 44.** A multi-objective optimization problem is defined as a tuple  $\langle D, O, F, C \rangle$ , where  $D$  is the design (decision) space,  $O \subseteq R^n$  is the objective space,  $F = \langle f_1, \dots, f_n \rangle$  with  $f_i : D \rightarrow \mathbb{R}$  is the set of  $n$  objective functions, and  $C = \{c_1, \dots, c_l\}$  is the set of  $l$  constraints.

There are some challenges to overcome compared to single-objective optimization. With the single optimization, the solutions are linearly order according to  $f$ . This may not apply to multi-objective problems, as it may be the case that  $f_1(x)$  is better than  $f_1(y)$  but at the same time  $f_2(y)$  has better objective value than  $f_2(x)$ . This is formalized by the definition of Pareto dominance.

**Definition 45.** Individual  $x$  Pareto dominates individual  $y$  ( $x \prec y$ ) (equivalently, individual  $y$  is Pareto dominated by the individual  $x$ ), if for each objective  $f_i : f_i(x) \leq f_i(y)$ , and there is at least one objective  $f_i$  for which  $f_i(x) < f_i(y)$ .

If neither  $(x \prec y)$  nor  $(x \succ y)$ , we say that  $x$  and  $y$  are (mutually) non-dominated.

Pareto dominance is not a total order on  $D$ , if there is a pair that is mutually non-dominated. This gives a notion to a goal of the multi-objective optimization, as we will be looking for the set that is not dominated by other elements in  $D$ .

**Definition 46.** *The solution of multi-objective optimization problem  $\langle D, O, F, C \rangle$  is a Pareto set  $P \subset D$ , such as for each  $x \in D$  and  $y \in P$ , the individual  $y$  is not dominated by the individual  $x$ . The image of the Pareto set  $P$  under the objectives  $F$  is a subset of  $O$  called the Pareto front.*

In practise, finding the enumeration of the solution of the multi-objective optimization problem is often not possible because the solution may be uncountable because it can be uncountable subset of  $\mathbb{R}$ . If the goal is to enumerate the solution and not to provide function of all elements in the Pareto set, no algorithm can provide a complete solution. This gives a notion of approximation of the solution:

**Definition 47.** *A Pareto set approximation  $A \in D$  is a finite set of points in the decision space such that for each two points  $x, y \in A$ ,  $x$  and  $y$  are mutually non-dominated.*

### 8.2.2 Multi-objective Evolutionary Algorithms

There has been a large number of multi-objective evolutionary algorithms proposed in the past. Examples are the Non-dominated Sorting Genetic Algorithm (NSGA [117]), NSGA-II [32] and Multi-objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES) [53]. For more complete survey of multi-objective evolutionary algorithms please refer to [76] and [127].

In this thesis, we will use NSGA-II. Although this algorithm is rather old, it is still among the best optimizers for two-objective problems [54]. Newer algorithms usually outperforms NSGA-II when the number of objective functions is high. In this case, almost all solutions in the population become non-dominated and the convergence property of the algorithm becomes severely deteriorated. In this thesis, we will have at most two objectives, therefore the NSGA-II algorithm is a suitable choice. The main idea of the algorithm is its environmental selection. The evolution prefers individuals who dominate more and who bring more diversity to the population. NSGA-II first sorts individuals to numbered sets called fronts. Individuals from some front dominate all individuals from the fronts with the higher number. Each individual is assigned the number of its front called rank. Compared to its predecessor NSGA where assigning the individuals according to their front took  $\mathcal{O}(MN^3)$  time, where  $N$  is the population size and  $M$  number of objectives, the fast Non-dominated sort outlined in Algorithm 20 reduced the time complexity to  $\mathcal{O}(MN^2)$ . The diversity in each front is empowered by so called distance. Individuals in each front with bigger differences in its objectives are preferred. The boundary individuals (with at least one objective being the highest or the lowest in its front) are labelled as most distant. The assignment of the distance to each individuals is outlined in Algorithm 21. Given the notion of rank and distance, we can define the partial order  $\prec_n$  that the algorithm uses to guide the evolution:

$$\begin{aligned} x \prec_y & \text{ if } x.rank < y.rank, \\ x \prec_y & \text{ if } x.rank = y.rank \text{ and } x.distance > y.distance. \end{aligned}$$

That is we prefer solutions with better rank. In the case of a tie, we prefer individuals with better distance. To ensure elitism (i.e. the fact that the best found solutions are not lost during the selection), NSGA-II first merges the parent and children population and the ranks are assigned based on the merged population. Another important feature of NSGA-II are the operators which are used. The usual crossover operator is the so called *simulated binary crossover* (SBX) [30]. This operator performs arithmetic crossover (i.e. it makes a weighted average of two parents), but the weights are selected in such a way that the change in the values of the variables is similar to the change of variables when one-point crossover on binary encoded strings is used. Basically, it means that the variables of the offspring have higher probability to be closer to one of the parents than if the weights are selected uniformly. The mutation operator [31] – called Polynomial Mutation uses a similar idea. The relative changes in the values of the variables should be similar to those of a bit-flip mutation on binary strings. The generation increment of the NSGA-II is described in Algorithm 22. The complexity of each increment is as follows:

1. Nondominated sorting is  $\mathcal{O}(M(2N)^2)$ .
2. Crowding-Distance assignment is  $\mathcal{O}(M2N \log(2N))$ .
3. Sorting on  $\prec_n$  is  $\mathcal{O}(2N \log(2N))$ .

The  $N$  stands for the population size and  $M$  is the number of objectives. Total complexity of the generation increment is  $\mathcal{O}(MN^2)$ .

## 8.3 Experiments

In the new experiments, we will be amending the experiments used in Chapter 7. We will propose experiments using bootstrapping, coevolution and antibloat operators and compare their results with the rest of the algorithms used in this thesis. We will also present the results of our multi-objectivization experiments for the algorithm ranking problem. These were performed over the similar, though not the same, dataset.

### 8.3.1 Coevolution

To implement coevolution, we will amend the GP algorithm presented in Chapter 7. The algorithm evolved both trees as one individual. We will split the population into two, one will correspond to the numerical and second to the categorical population. The functions and terminals for each population will be also split accordingly. In every generation we will generate random bijection between categorical and numerical trees. We will calculate the fitness as if these two trees would be a single individual, and we will still use the fitness from the original GP algorithm.

We would require about 150 nodes to represent the weighted attribute metric from Chapter 6. It is difficult for the GP to find similar distance measures as the search space is very big. Therefore, we decided to try bootstrapping so the GP can use the useful components of the metric evolved in the previous

---

**Algorithm 20:** Fast-non-dominated Sort

---

// Pseudocode for computing the rank of individuals.

**input** :  $I \leftarrow$  List of individuals

```
1 foreach  $p$  in  $I$  do
2    $S_p \leftarrow \emptyset$ ;
3    $n_p \leftarrow 0$ ;
4   foreach  $q$  in  $I$  do
5     if  $p \prec q$  then
6        $S_p \leftarrow S_p \cup \{q\}$ ;
7     end
8     else if  $q \prec p$  then
9        $n_p \leftarrow n_p + 1$ ;
10    end
11  end
12  if  $n_p = 0$  then
13     $p_{\text{rank}} \leftarrow 0$ ;
14     $F_1 \leftarrow F_1 \cup \{p\}$ ;
15  end
16 end
17  $i = 1$ ;
18 while  $F_i \neq \emptyset$  do
19    $Q \leftarrow \emptyset$ ;
20   foreach  $p$  in  $F_i$  do
21     foreach  $q$  in  $S_p$  do
22        $n_q \leftarrow n_q - 1$ ;
23       if  $n_q = 0$  then
24          $q_{\text{rank}} \leftarrow i + 1$ ;
25          $Q \leftarrow Q \cup \{q\}$ ;
26       end
27     end
28   end
29    $i \leftarrow i + 1$ ;
30    $F_i \leftarrow Q$ ;
31 end
```

---

---

**Algorithm 21:** Crowding Distance Assignment

---

```
// Pseudocode for computing distance between individuals in
// the Pareto front. The distance is used by the NSGA-II to
// maintain diversity in the Pareto front.
input :  $F \leftarrow$  List of objectives
input :  $I \leftarrow$  List of individuals

1 size = len( $I$ );
2 foreach  $i$  in  $I$  do
3   |  $i$ .distance  $\leftarrow 0$ ;
4 end
5 foreach  $f \in F$  do
6   |  $I \leftarrow \text{sort}(I, f)$ ;
7   |  $I[0]$ .distance  $\leftarrow I[\text{size} - 1]$ .distance  $\leftarrow \infty$ ;
8   |  $f_{\min} \leftarrow f(I[0])$ ;
9   |  $f_{\max} \leftarrow f(I[\text{size} - 1])$ ;
10  | for  $k$  in  $\{1, \dots, \text{size} - 2\}$  do
11  |   |  $I[k]$ .distance  $\leftarrow I[k]$ .distance +  $\frac{f(I[k - 1]) + f(I[k + 1])}{f_{\max} - f_{\min}}$ ;
12  | end
13 end
```

---

---

**Algorithm 22:** NSGA-II

---

```
// Pseudocode for population increment of the NSGA-II
// algorithm.
input:  $P_t \leftarrow$  Parent population in the time  $t$ 
input:  $Q_t \leftarrow$  Offspring population in the time  $t$ 

1  $R_t \leftarrow Q_t \cup P_t$ ;
2  $F \leftarrow \text{fast-non-dominated-sort}(R_t)$ ;
3  $P_{t+1} \leftarrow \emptyset$ ;
4  $i \leftarrow 1$ ;
5 while  $|P_{t+1}| + |F_i| \leq N$  do
6   | crowding-distance-assignment( $F_i$ );
7   |  $P_{t+1} \leftarrow P_{t+1} \cup F_i$ ;
8   |  $i \leftarrow i + 1$ ;
9 end
10  $\text{sort}(F_i, \prec_n)$ ;
11  $P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$ ;
12  $Q_{t+1} \leftarrow \text{apply-operators}(P_{t+1})$ 
```

---

experiments. As in the GP experiments, we do not evolve weights. Therefore, we decided to use individuals who did not discriminate between categorical and numerical attributes and whose selector weights were around the same value. We also decided not to use the best individual as we wanted to allow GP algorithm to fine tune the distance itself. It could be hard otherwise to beat the given distance thus creating the same problem again that we are addressing with the bootstrapping.

We have chosen the individual with the numerical weight equal to 2.508826 and categorical to 2.250011. With the weights included, the training fitness was equal to 0.554701 on the training set and to 0.54561 on the testing set respectively. As the GP did not incorporate the weights, the fitness slightly changed to 0.553206 on the training set and to 0.544519 on the testing set respectively.

We have not changed any other parameters.

### 8.3.2 Antibloat

In the antibloat experiments we have reused the framework from Chapter 7. We have only amended the tournament selection. If the size of the individual exceeds the limit, we decrease the fitness of the individual. This decrease is only for the selection purposes, we have not amended this for the sake of elitism and ranking quality reporting. Let us suppose that we have some individual with fitness  $f \in \langle 0, 1 \rangle$ . We rescaled the fitness by every of the following, once per each penalty:

1. One percent down per each 10 nodes above 200 in the categorical tree.
2. One percent down per each 10 nodes above 200 in the numerical tree.
3. One percent down per each 5 points above 20 measured in the maximum width of the levels in the categorical tree.
4. One percent down per each 5 points above 20 measured in the maximum width of the levels in the numerical tree.
5. One percent down per each 5 points above 20 measured in the maximum depth of the levels in the categorical tree.
6. One percent down per each 5 points above 20 measured in the maximum depth of the numerical tree.

We set a cap to every such rescaling to 0.9, as we did not want a single penalty to completely negate the fitness of particular individual. Furthermore, we did not allow the fitness to become less than zero.

### 8.3.3 Results

The results of individual runs of the GP algorithm with antibloat operator and coevolution with bootstrapping can be found in Table 8.1. Statistical comparison with the results of the previous algorithms can be found in Table 8.2. The algorithm with bootstrapping and coevolution managed to beat the baseline and all the pure assignment based algorithms. It also managed to match the level of

Table 8.1: Evaluation of the ranking quality of the trees produced by the GP algorithm using antibloat operator and bootstrapping with coevolution on the testing dataset.

Run	Coevolution+Bootstrap	Antibloat
1	0.559953	0.553059
2	0.553473	0.54562
3	0.552748	0.543022
4	0.551044	0.541717
5	0.550729	0.5394
6	0.548995	0.539193
7	0.548331	0.533419
8	0.548253	0.532851
9	0.544634	0.53082
10	0.544221	0.528787
median	0.549862	0.539297

the first combination of the global metafeatures and assignments. In some runs the overfitting was still present – although the algorithm managed to improve the fitness on the training set, in some cases it did not improve the results on the validation set. That significantly decreased its score in the overall results. However, one run managed to outperform every other run using the global attributes only and produced one of the best results using solely the assignments. This suggest that attribute assignment has a very good potential to be improved with further empowering the generalization abilities of GP algorithm. The runs with the antibloat operator reduced bloating, however we did not observe a big difference in the resulting ranking quality. This suggests that the bloating is not the only factor reducing the generalization abilities.

Table 8.2: Statistical comparison of GP using antibloat operator and coevolution with bootstrapping and previous algorithms and their ranking quality results on the validation set. W stands for GP statistically worse than the algorithm in the corresponding column, I stands for inconclusive and B stands for statistically better.

	Coevolution+Bootstrap Aggregation, $p = 1$ Aggregation, $p = 2$ Global, $p = \text{inf}$ Global, $p = 1$ Global, $p = 2$ Aggregation, $p = \infty$ Assignment, $p = 2$ Assignment, $p = \infty$ GP Baseline										
Coev+Boot	W	W	W	W	W	I	B	B	B	B	B
Antibloat	W	W	W	W	W	W	W	I	I	I	I

### 8.3.4 Multi-objectivization

We have experimented with the multi-objectivization for the algorithm ranking in [111]. Herein, we have used similar OpenML dump. The difference was in fewer filters applied, as we did not compare with propositional approaches and therefore we did not include the requirements that all datasets have all global metadata available.

The whole workflow was derived from the one used in Chapter 7, although we did not explicitly repair the distance function to a semimetric.

As discussed in Section 8.2, it is better to include metric similarity of the attribute distance instead of dataset distance. For that reason we have used resemblance of attribute distance measure  $\delta$  to a metric as a second criterion. To be precise, for each selector we measured

$$E_{selector} = \frac{e_1 + e_2 + e_3 + e_4}{4},$$

where  $e_i$  is the ratio of instances (tuples of triples) where metric axiom  $i$  did not hold. We then based the second criterion on the aggregation of  $E$  over all selectors:

$$f_2 = 1 - \frac{\sum_{s \in Selectors} E_s}{|Selectors|}.$$

If we repaired the distance function to a semimetric, as we have already mentioned, it would be enough to measure just the amount of cases where triangle inequality held.

As a multi-objective algorithm the NSGA-II was chosen. The size of the population was set to 200. The tournament approach was selected as the selection mechanism. A better individual was chosen according to rank and crowding distance. The probability of better individual winning the tournament was set to 0.7. Also, the NSGA-II uses elitism, so the best individual were guaranteed to be copied to the next generation. Tree mutation and crossover were used as genetic operators. The probabilities of mutation and crossover were set to 0.2 and 0.7 respectively. The termination criterion was set to 80 generations. This was mainly we noticed that the bloating usually appeared around this generation.

We have performed seven runs and we obtained significantly better results compared to the baseline algorithm. Furthermore, we have observed the high correlation of the metric similarity and prediction accuracy. This supports the hypothesis that the metric properties are important for the generalization abilities of the induced dataset similarity measure. This was the most obvious during the second run, whose results are depicted in Figures 8.1 (training) and 8.2 (testing). The Spearman's rank correlation coefficient between the first and second criterion of the second run on the testing set was 0.734. This means that in the testing set the individuals more similar to a metric had better results for the prediction of the algorithm ranking.

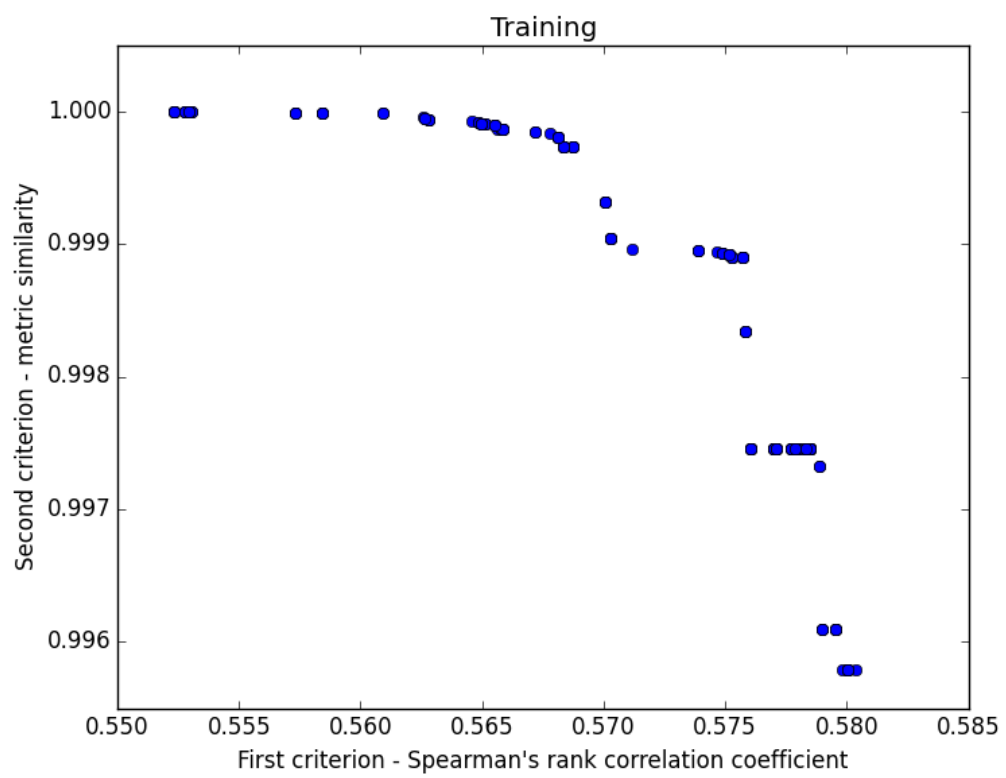


Figure 8.1: Results of the first Pareto front of the second run on the training set.

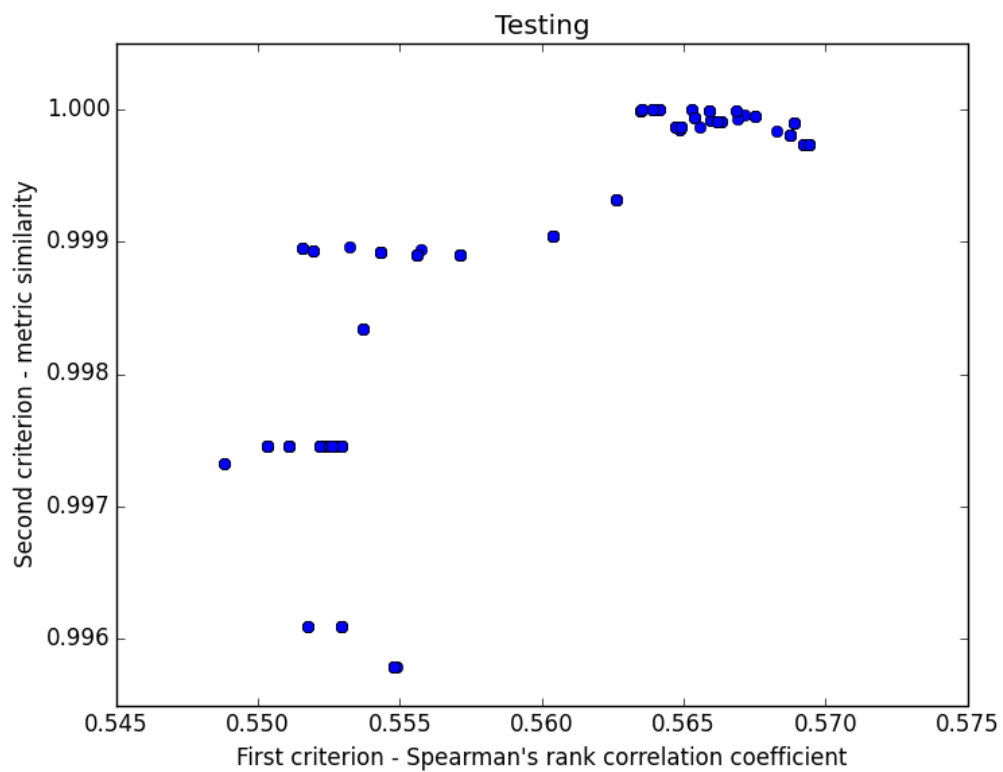


Figure 8.2: Results of the first Pareto front of the second run on the testing set. Note the high correlation between both criteria - the higher values of the accuracy criterion are associated with higher values of the metric similarity criterion.

# Chapter 9

## Conclusion

In this thesis, we have studied the non-propositional approach for comparing two datasets. We have focused on algorithm ranking problem in metalearning. However, our approach is not limited solely to this area and can be easily reused even in other fields. Specifically, the distance measures defined on some set  $\mathbb{X}$  can be transformed by attribute assignment algorithms to a distance measure on the power set  $2^{\mathbb{X}}$  of  $\mathbb{X}$ .

The main contribution lies in the design of multiple algorithms for measuring the distance between two datasets that can handle non-propositional metadata and their unique theoretical properties. The difference between them is in the computational complexity, expression power and guarantees on the resulting dataset distance measure.

When proposing our algorithms, we built our work on the related approaches. The literature suggested several methods to handle the non-propositionality, either directly in the metalearning fields or in some other areas of computational intelligence. However, we have identified several areas that could improve the reviewed literature. For instance, the current approaches are either losing important information, lack metric properties or assume that the order is important (although we can reshuffle attributes in datasets without changing the information). This drove our motivation to propose new methods in the first place. Furthermore, authors of [59] recognized the problem of building the distance over non-propositional datasets as non-trivial.

The main idea behind our algorithms rests upon the idea of attribute assignment. To measure distance between two datasets, we first propose an attribute distance measure. We first amend the datasets so their cardinalities match, by inserting artificial attributes into the dataset with less attributes. Then we find bijection of attributes from the first dataset to another so the sum of the distances defined by the bijection and the attribute is minimized. This sum of distances is also the desired distance between datasets.

We have proven that under certain conditions, the resulting distance between datasets is a metric. The main condition for this is that the underlying attribute distance measure itself is a metric. Other conditions include several restrictions on extending the attribute space by a *dummy* attribute. We have also proven that the above is valid also in the other directions. However, the first direction is somewhat stronger if we have only training data and is optimizing towards a metric on either dataset or attribute subspace defined by training data. Some

other ideas were considered, for instance the normalization of the sum of attribute distances by the number of attributes. However, we have proven that this can break the triangle inequality axiom.

We have designed our algorithm to be extensible. It is possible to use the metadata specific for some attribute types, for instance for categorical and numerical metadata. This is achieved by splitting the distance into two, one for each metadata type. We have verified that this does not affect the metric properties of the algorithms. It is also possible to combine multiple metric distance into one. This allows our assignment approach to be combined with propositional approaches. As both approaches return useful information, we argued that by combining them, even more accurate sense of distances between datasets should be obtained.

We have designed generic workflow for measuring quality of ranking. As we made sure that all our algorithms conform to specified interfaces, it is possible to replace different parts of the workflow. This allowed us to compare different algorithms and their combinations proposed in the thesis in a unified way. We have also proven that it is possible to optimize distance based measures without changing their properties. This enabled us to employ optimization methods to boost the performance of our algorithms. We have employed genetic algorithms and genetic programming for this purpose.

Results of genetic algorithm experiments suggest that attribute alignment algorithms can be successfully used for algorithm ranking. Every parameter settings we optimized produced statistically better results than the baseline algorithm with a single exception. The aggregation of global and attribute approach produced the best results. With genetic programming, we traded triangle inequality for higher expression power. It was so powerful that it could overfit the training data very easily. To counter this, we have employed several approaches to improve generalization abilities. With this we managed to further improve the results of the assignment algorithms on our data. Especially coevolution combined with bootstrapping of the population managed to obtain promising results. We also reviewed one of our previous work where we used multi-objectivization to boost the generalization abilities of models being evolved. We introduced metric resemblance as a second criterion. Results suggested that there is a high correlation between the second criterion and generalization abilities of the models.

We have also demonstrated visualization of non-propositional dataset representation using the kernelized PCA. We investigated the visualization in more depth, and the results seems to be plausible as the visualisation rendered similar datasets in the same cluster.

## 9.1 Future Work

The work presented in this thesis can be extended in several directions. We wanted to focus on the attribute based distance. As the whole workflow of all the pieces plugged together became quite complicated, we did not want to add extra parts that would distract from the main idea by increasing the complexity of the workflow. It would be possible however to use ensemble based learning on top of our models to get the combined accuracy of our models.

As we are using distance based algorithms, it could be beneficial to try better

methods, such as weighted  $k$ -NN, that would make use of the distances of the nearest neighbours. It would also help to have more data. As we are dealing with high dimensional dataset space, hundreds of datasets is still very small amount to reasonably cover the space.

Our algorithms need lots of parameters. As our resources are limited and the training of models took significant amount of time, we could not dedicate much space to tuning of the parameters, and parameters we used were based on the previous experiments or short preliminary experiments. We would like to tune the parameter of  $k$  of the  $k$ -NN, different parameters and genetic operators of genetic algorithms used, with different strategies for adding the dummy attribute (either constant or different attribute when selected from attribute space). Even in this thesis, we have observed that parameters can significantly improve the overall results of experiments.

We would also like to enhance Genetic Programming algorithm with types. Typed genetic programming [88, 71] can help reduce the space that is searched and allows for more elaborate constructs and operators.

A room for improvement can be also seen in the metafeatures we are extracting from the attributes. Currently, only simple, statistical and theoretical metafeatures are extracted. It could be interesting whether we can extract some sort of landmarks on the attribute level. For example, we can try to predict target values using only a single attribute and use the results as a new metafeatures. It could also help to find metafeatures that are really useful for the generalization. It may well be the case that some metafeatures are just used to overfit the data and instead of contributing to the generalization abilities of the models, they are just downgrading the validation results.

We also see an opportunity in expanding the theoretical work. After the alignment, we use the sum of individual attribute distances given by the assignments to get the metric on datasets provided that certain conditions hold. It could be interesting to see whether some other aggregations can be used without sacrificing the nice property of metric preservations. It could be also beneficial to define sort of normalization of the resulting dataset distance that does not violate metric axioms.

# Bibliography

- [1] *Attribute-Relation File Format (ARFF)*, January 2001.
- [2] Salisu Abdulrahman and Pavel Brazdil. Measures for combining accuracy and time for meta-learning. In *Proceedings of the International Workshop on Meta-learning and Algorithm Selection co-located with 21st European Conference on Artificial Intelligence, MetaSel@ECAI 2014, Prague, Czech Republic, August 19, 2014.*, pages 49–50, 2014.
- [3] Salisu Abdulrahman, Pavel Brazdil, Jan N. van Rijn, and Joaquin Vanschoren. Algorithm selection via meta-learning and sample-based active testing. In *MetaSel@PKDD/ECML*, volume 1455 of *CEUR Workshop Proceedings*, pages 55–66. CEUR-WS.org, 2015.
- [4] Yaser S. Abu-Mostafa, Malik Magdon-Ismael, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.
- [5] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 555–564. ACM, 2014.
- [6] A. Agresti. *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, 2nd edition, 2002.
- [7] Fabio Aioli. Transfer learning by kernel meta-learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham W. Taylor, and Daniel L. Silver, editors, *ICML Unsupervised and Transfer Learning*, volume 27 of *JMLR Proceedings*, pages 81–95. JMLR.org, 2012.
- [8] Shawkat Ali and Kate A. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1–3):173 – 186, 2006. Neural NetworksSelected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04)7th Brazilian Symposium on Neural Networks.
- [9] Ehsaneddin Asgari and Mohammad R. K. Mofrad. Protvec: A continuous distributed representation of biological sequences. *CoRR*, abs/1503.05140, 2015.
- [10] E Gregory Ashby and Nancy A. Perrin. Toward a unified theory of similarity and recognition. *Psychological Review*, 95:124–150, 1988.

- [11] George Athanasopoulos and Rob J. Hyndman. The value of feedback in forecasting competitions. *International Journal of Forecasting*, 27(3):845 – 849, 2011. Special Section 1: Forecasting with Artificial Neural Networks and Computational Intelligence Special Section 2: Tourism Forecasting.
- [12] K. Bache and M. Lichman. UCI machine learning repository, 2013.
- [13] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. Jade - a java agent development framework. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer, 2005.
- [14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.
- [15] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117 – 127, 1992.
- [16] Pavel Brazdil, Christophe G. Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning — Applications to Data Mining*. Cognitive Technologies. Springer, 2009.
- [17] Pavel B. Brazdil and Carlos Soares. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In *Proceedings of Principles of Data Mining and Knowledge Discovery, 4th European Conference (PKDD 2000)*, pages 126–135. Springer, 2000.
- [18] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277.
- [19] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.
- [20] Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Christian Klein, Frank Neumann, and Eckart Zitzler. On the effects of adding objectives to plateau functions. *Trans. Evol. Comp*, 13(3):591–603, June 2009.
- [21] Edmund K. Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.
- [22] Darren A. Casella and Walter D. Potter. New lower bounds for the snake-in-the-box problem: Using evolutionary techniques to hunt for snakes. In Ingrid Russell and Zdravko Markov, editors, *FLAIRS Conference*, pages 264–269. AAAI Press, 2005.
- [23] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.
- [24] The UniProt Consortium. UniProt: a hub for protein information. *Nucleic Acids Research*, 43(D1):D204–D212, January 2015.

- [25] Gregory W. Corder and Dale I. Foreman. *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2014.
- [26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [27] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 1967.
- [28] Charles Darwin. *On the origin of species by means of natural selection; or, The preservation of favoured races in the struggle for life / by Charles Darwin*. New York :D. Appleton and Co.,. <http://www.biodiversitylibrary.org/bibliography/39967> — Includes index. — [2] p. of publisher’s advertisements at end.
- [29] Peter Dawyndt, Hans De Meyer, and Bernard De Baets. {UPGMA} clustering revisited: A weight-driven approach to transitive approximation. *International Journal of Approximate Reasoning*, 42(3):174 – 191, 2006.
- [30] Kalyanmoy Deb, Ram Bhusan Agrawal, and Ram Bhushan Agrawal. Simulated Binary Crossover for Continuous Search Space. 1995.
- [31] Kalyanmoy Deb and Mayank Goyal. A Combined Genetic Adaptive Search (GeneAS) for Engineering Design. 1996.
- [32] Kalyanmoy Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.
- [33] Pedro A. Diaz-Gomez and Dean F. Hougen. Initial Population for Genetic Algorithms: A Metric Approach. 2007.
- [34] Alexei Drummond and Allen G. Rodrigo. Reconstructing genealogies of serial samples under the assumption of a molecular clock using serial-sample upgma. *Molecular Biology and Evolution*, 17(12):1807–1815, 2000.
- [35] S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327, April 1976.
- [36] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, April 1972.
- [37] Gregory E Fasshauer. Positive definite kernels: past, present and future. *Dolomite Research Notes on Approximation*, 4:21–63, 2011.
- [38] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. *Agent-Oriented Software Engineering IV: 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003. Revised Papers*, chapter From Agents to Organizations: An Organizational View of Multi-agent Systems, pages 214–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
- [39] R. A. Fisher. *Annals of Eugenics*.

- [40] Neal Ford. *Functional thinking*. O'Reilly Media, Sebastopol, CA, 2014.
- [41] Johannes Fürnkranz and Johann Petrak. An evaluation of landmarking variants. In *Proceedings of the ECML/PKDD Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning (IDDM-2001)*, pages 57–68, 2001.
- [42] Mario Graff and Riccardo Poli. Practical performance models of algorithms in evolutionary program induction and other domains. *Artif. Intell.*, 174(15):1254–1276, October 2010.
- [43] Mario Graff and Riccardo Poli. Practical performance models of algorithms in evolutionary program induction and other domains. *Artificial Intelligence*, 174(15):1254–1276, October 2010.
- [44] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
- [45] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *In Neural Information Processing Systems*, page 2003. MIT Press, 2003.
- [46] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.
- [47] Inman Harvey, Phil Husbands, and Dave Cliff. *Seeing the Light: Artificial Evolution, Real Vision*. 1994.
- [48] R. J. Henery. Methods for comparison. In Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors, *Machine learning, neural and statistical classification*, pages 107–124. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
- [49] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. *Kernel methods in machine learning*, 2008.
- [50] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.
- [51] Holger Hoos, Marius Thomas Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *CoRR*, abs/1405.1520, 2014.
- [52] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, chapter Sequential Model-Based Optimization for General Algorithm Configuration, pages 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

- [53] Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evol. Comput.*, 15(1):1–28, March 2007.
- [54] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. Evolutionary many-objective optimization: A short review. In *IEEE Congress on Evolutionary Computation*, pages 2419–2426. IEEE, 2008.
- [55] Y. Jin. *Multi-objective machine learning*, volume 16. Springer, 2006.
- [56] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.
- [57] Kenneth A. De Jong and Mitchell A. Potter. Evolving complex structures via cooperative coevolution. In *Evolutionary Programming*, pages 307–317, 1995.
- [58] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, CP’11*, pages 454–469, Berlin, Heidelberg, 2011. Springer-Verlag.
- [59] Alexandros Kalousis and Melanie Hilario. Representational issues in meta-learning. In *ICML*, pages 313–320. AAAI Press, 2003.
- [60] Alexandros Kalousis and Theoharis Theoharis. Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337, 1999.
- [61] O. Kazík, J. Šmíd, and R. Neruda. Evolutionary optimization of meta data metric for method recommendation. In *Cybernetics and Intelligent Systems (CIS), IEEE Conference on*, pages 123–127, Nov 2013.
- [62] Ondřej Kazík, Klára Pešková, Martin Pilát, and Roman Neruda. Meta learning in multi-agent systems for data mining. In *International Conference on Intelligent Agent Technology (IAT 2011)*, pages 433–434. IEEE Computer Society, 2011.
- [63] Ondřej Kazík and Roman Neruda. *Data Mining Process Optimization in Computational Multi-agent Systems*, pages 93–103. Springer International Publishing, Cham, 2015.
- [64] Ondřej Kazík and Roman Neruda. Ontological modeling of meta learning multi-agent systems in OWL-DL. *IAENG International Journal of Computer Science*, 39(4):357–362, Dec 2012.
- [65] Ondřej Kazík, Klára Pešková, Martin Pilát, and Roman Neruda. Implementation of parameter space search for meta learning in a data-mining multi-agent system. In *Proceedings of the 2011 Tenth International Conference on Machine Learning and Applications, ICMLA ’11*, pages 366–369. IEEE Computer Society, 2011.

- [66] Ondřej Kazík, Klára Pešková, Martin Pilát, and Roman Neruda. Meta learning in multi-agent systems for data mining. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 2:433–434, 2011.
- [67] Ondřej Kazík, Klára Pešková, Jakub Šmíd, and Roman Neruda. Clustering based classification in data mining method recommendation. In *International Conference on Machine Learning and Applications (ICMLA 2013)*, pages 356–361. IEEE Computer Society, 2013.
- [68] Pavel Kordík and Jan Černý. On performance of meta-learning templates on different datasets. In *IJCNN*, 2012.
- [69] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [70] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- [71] T. Křen and R. Neruda. Generating lambda term individuals in typed genetic programming using forgetful A\*. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1847–1854, July 2014.
- [72] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1092–1097, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
- [73] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. *CoRR*, abs/1205.3193, 2012.
- [74] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. *Machine Learning and Data Mining in Pattern Recognition: 8th International Conference, MLDM 2012, Berlin, Germany, July 13-20, 2012. Proceedings*, chapter Selecting Classification Algorithms with Active Testing, pages 117–131. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [75] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. *Proc. Pacific Symposium on Biocomputing*, 7:566–575, 2002.
- [76] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-objective evolutionary algorithms: A survey. *ACM Comput. Surv.*, 48(1):13:1–13:35, September 2015.
- [77] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence*, 53:745–778, August 2015. To appear.

- [78] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002.
- [79] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evol. Comput.*, 14(3):309–344, September 2006.
- [80] M. Maratea, L. Pulina, and F. Ricca. The Multi-engine ASP Solver ME-ASP: Progress Report. *ArXiv e-prints*, May 2014.
- [81] Marco Maratea, Luca Pulina, and Francesco Ricca. *AI\*IA 2013: Advances in Artificial Intelligence: XIIIth International Conference of the Italian Association for Artificial Intelligence, Turin, Italy, December 4-6, 2013. Proceedings*, chapter Automated Selection of Grounding Algorithm in Answer Set Programming, pages 73–84. Springer International Publishing, Cham, 2013.
- [82] Gregor Mendel. Versuche über Pflanzen-Hybriden. *Verhandlungen des naturforschenden Vereines in Brünn*, 42:3–47, 1866.
- [83] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.
- [84] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [85] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.
- [86] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.
- [87] Mustafa Misir and Michèle Sebag. Algorithm Selection as a Collaborative Filtering Problem. Research report, December 2013.
- [88] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3:199–230, 1994.
- [89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [90] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*,

pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

- [91] Klára Pešková, Jakub Šmíd, Martin Pilát, Ondřej Kazík, and Roman Neruda. Hybrid multi-agent system for metalearning in data mining. In *Proceedings of the International Workshop on Meta-learning and Algorithm Selection co-located with 21st European Conference on Artificial Intelligence, MetaSel@ECAI 2014, Prague, Czech Republic, August 19, 2014.*, pages 53–54, 2014.
- [92] Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 743–750. Morgan Kaufmann, 2000.
- [93] Martin Pilát and Roman Neruda. Multi-objectivization and surrogate modelling for neural network hyper-parameters tuning. In *Emerging Intelligent Computing Technology and Applications*, pages 61–66. Springer Berlin Heidelberg, 2013.
- [94] Riccardo Poli. *Genetic Programming: 6th European Conference, EuroGP 2003 Essex, UK, April 14–16, 2003 Proceedings*, chapter A Simple but Theoretically-Motivated Method to Control Bloat in Genetic Programming, pages 204–217. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [95] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.
- [96] Stefan Poslad. Specifying protocols for multi-agent systems interaction. *ACM Trans. Auton. Adapt. Syst.*, 2(4), November 2007.
- [97] Mitchell A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, Fairfax, VA, USA, 1997. UMI Order No. GAX97-28573.
- [98] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.*, 8(1):1–29, March 2000.
- [99] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
- [100] Jan Ramon and Maurice Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.
- [101] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
- [102] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

- [103] J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, Oct 2006.
- [104] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 3 edition, December 2009.
- [105] H. Saigo, J-P. Vert, and T. Akutsu. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC Bioinformatics*, 7(246):1–12, May 2006.
- [106] Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.
- [107] Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Advances in kernel methods. chapter Kernel Principal Component Analysis, pages 327–352. MIT Press, Cambridge, MA, USA, 1999.
- [108] Tianze Shi and Zhiyuan Liu. Linking glove with word2vec. *CoRR*, abs/1411.5595, 2014.
- [109] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, June 2007.
- [110] J. Šmíd and R. Neruda. Comparing datasets by attribute alignment. In *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, pages 56–62, Dec 2014.
- [111] J. Šmíd, M. Pilát, K. Pešková, and R. Neruda. Multi-objective genetic programming for dataset similarity induction. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 1576–1582, Dec 2015.
- [112] Jakub Šmíd. Agent optimization by means of genetic programming. Master’s thesis, Charles University in Prague, Prague, Czech Republic, 2012.
- [113] Jakub Šmíd and Roman Neruda. Using genetic programming to estimate performance of computational intelligence models. In Marco Tomassini, Alberto Antonioni, Fabio Daolio, and Pierre Buesser, editors, *Adaptive and Natural Computing Algorithms (Proceedings of ICANNGA 2013)*, volume 7824 of *Lecture Notes in Computer Science*, pages 169–178. Springer Berlin Heidelberg, 2013.
- [114] Jakub Šmíd, Martin Pilát, Klára Pešková, and Roman Neruda. Co-evolutionary genetic programming for dataset similarity induction. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1160–1166. IEEE, 2015.
- [115] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

- [116] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Comput. Oper. Res.*, 45:12–24, May 2014.
- [117] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, September 1994.
- [118] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.
- [119] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260.
- [120] Fevrier Valdez, Patricia Melin, and Oscar Castillo. A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation. *Expert Systems with Applications*, 41(14):6459 – 6466, 2014.
- [121] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.
- [122] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, August 2010.
- [123] Rudolf Paul Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, Fairfax, VA, USA, 2004. AAI3108645.
- [124] David H. Wolpert. The supervised learning no-free-lunch theorems. In *In Proc. 6th Online World Conference on Soft Computing in Industrial Applications*, pages 25–42, 2001.
- [125] Adam Woznica, Alexandros Kalousis, and Melanie Hilario. Distances and (indefinite) kernels for sets of objects. In *ICDM*, pages 1151–1156. IEEE Computer Society, 2006.
- [126] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.
- [127] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagarathan Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.
- [128] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195, June 2000.

# List of Figures

2.1	Space of optimisation problems was projected to two dimensions by the PCA algorithm and coloured according to the best algorithm in that area. Grey colour is for the instances that had multiple best algorithms (according to some small margin) [116]. . . . .	12
2.2	Types of ranking. The left one is linear complete, middle one is weak and complete and the ranking on the right is linear and incomplete. Adapted from [16]. . . . .	14
2.3	Example of the tree for estimating the algorithm duration on a new dataset. Terminals are blue, functions are green. The node labelled $D:i$ represents distance from the $i$ -th nearest dataset. Similarly, the node labelled $E:i$ denotes error of the algorithm on the $i$ -th nearest dataset. $Com$ is the complexity terminal. . . . .	16
2.4	The dendrogram as a result of the agglomerative clustering [67]. Datasets are assigned to clusters according to their similarity. . .	22
2.5	Overview of claspfolio2 framework [51]. . . . .	28
2.6	Overview of Agent-Group-Role model of our recommendation system Pikater [91]. . . . .	29
4.1	Linguistic regularities preserved by the word2vec algorithm [84, 85].	44
4.2	Example of using the kernelized PCA in the scikit-learn library [89] to make the data linearly separable. . . . .	48
4.3	Example of the identity alignment. If the attributes are sorted by $\sigma$ , each attribute is aligned to the attribute with the same order. .	53
4.4	Example of transformation. Find attributes that falsify identity alignment and switch them. At least one less pair of attributes now falsifies the identity alignment. . . . .	54
4.5	Method one - attributes of datasets $a$ and $b$ were aligned according to their $\sigma$ . . . . .	60
4.6	Extending optimal assignment by row of dummy attributes without changing the cost. . . . .	62
4.7	Proof of Theorem 14 – metric axiom 4 is preserved. Alignment $f$ is extended and then reshuffled so the triangle inequality can be applied. Yellow nodes are original attributes of dataset $x$ , green nodes are attributes of dataset $y$ and blue of dataset $z$ . Red nodes are <i>dummy</i> attributes used for extending the datasets so the corresponding assignments are of the same cardinality. . . . .	65

4.8	Part of proof of Theorem 15. Constant $k$ representing a distance between a regular and an artificial <i>dummy<sub>n</sub></i> attribute needs to be big enough in order not to create the shortcut between the most distant points. . . . .	66
5.1	Distribution of number of attributes among datasets in the OpenML dump. . . . .	78
5.2	Distribution of global metafeatures computed by OpenML. . . . .	79
5.3	Distribution of values of categorical metafeatures after normalization. . . . .	88
5.4	Distribution of values of numerical metafeatures after normalization. . . . .	89
6.1	Example of the crossover genetic operator. . . . .	93
6.2	Example of the mutation genetic operator. . . . .	94
6.3	UML diagram of the whole workflow. Violet rectangles represent different ranking algorithms that will be tried in the experiments. Global Dataset Distance, Combined Attribute Assignment and Dataset Distance Aggregation will have their weights optimized by the genetic algorithm. The fitness will be provided by the Ranking Quality Evaluation. . . . .	100
6.4	Evolution progress of the individual with the best result on the testing set together with the baseline. Results on the training set are blue, on the testing set are green. On the $x$ -scale is the evolution progress, on the $y$ -scale is the quality of ranking measured by the Spearman's correlation coefficient. . . . .	101
6.5	Projection of the training datasets using kernelized PCA and similarity based on the aggregation of global and attribute distance with the best result on the testing set. . . . .	103
7.1	Example of the crossover in the Genetic Programming with the tree representation. . . . .	111
7.2	Example of the mutation in the Genetic Programming with the tree representation. . . . .	111
7.3	Example of the tree evolved by the GP for the numerical distance between two attributes. The terminals are blue compared to inner-nodes which are true. The label in the node describes the type. For instance, label $l$ represents LessThan function, similarly $UD:0$ is a variable gaining a value depending whether an attribute corresponding to the left argument (left is determined by the number 0) correspond to an uniform distribution. . . . .	116
7.4	UML diagram of the workflow for the GP experiments. This time the focus will be on the GP tree, which is going to be evolved using the genetic programming by the fitness from the Ranking Quality Evaluator. . . . .	118
8.1	Results of the first Pareto front of the second run on the training set. . . . .	130

8.2	Results of the first Pareto front of the second run on the testing set. Note the high correlation between both criteria - the higher values of the accuracy criterion are associated with higher values of the metric similarity criterion. . . . .	131
-----	---	-----

# List of Algorithms

1	<i>IDatasetDistance</i> : dataset distance interface . . . . .	18
2	<i>IRanking</i> : Interface for ranking calculation . . . . .	18
3	<i>IDistanceRanking</i> : Interface for distance based ranking calculation	19
4	Distance Ranking Transformation: transforming a <i>IDistanceRanking</i> interface to generic <i>IRanking</i> interface . . . . .	19
5	<i>K</i> -NN Ranking: <i>k</i> -NN based implementation of <i>IRanking</i> . . . . .	20
6	Ranking Quality Assessment . . . . .	23
7	Ranking Baseline . . . . .	25
8	Distance $\Delta$ using global metadata: <i>IDatasetDistance</i> . . . . .	41
9	Attribute alignment . . . . .	55
10	Hungarian algorithm . . . . .	56
11	<i>IAttributeDistance</i> : Dataset distance interface . . . . .	57
12	Attribute assignment . . . . .	57
13	<i>ISelectorInterface</i> : Interface for selecting subset of attributes. . .	58
14	<i>NumericalAttributesSelector</i> : <i>ISelectorInterface</i> for selecting nu- merical attributes . . . . .	58
15	<i>CategoricalAttributesSelector</i> : <i>ISelectorInterface</i> for selecting numerical attributes . . . . .	58
16	Combined Attribute Assignment . . . . .	59
17	Vectorized Attribute Distance: <i>IAttributeDistance</i> . . . . .	71
18	Dataset Distance Aggregation: <i>IDatasetDistance</i> . . . . .	72
19	Genetic Algorithm . . . . .	92
20	Fast-non-dominated Sort . . . . .	125
21	Crowding Distance Assignment . . . . .	126
22	NSGA-II . . . . .	126

# List of Tables

4.1	Equation 4.22 holds for every possible case. . . . .	54
4.2	Possible values of dataset $a$ . . . . .	59
4.3	Possible values of dataset $b$ . . . . .	59
4.4	Distance matrix of the attributes of datasets $a$ and $b$ . The matrix will serve as an input of the Hungarian algorithm. . . . .	60
4.5	Results of application of the Hungarian algorithm. The optimal alignment is coloured. . . . .	60
4.6	Possible values of dataset $c$ . . . . .	61
4.7	Counterexample that metric on some subspace of datasets does not imply metric on its source attribute subspace . . . . .	70
5.1	Excluded global metafeatures. . . . .	83
6.1	Values of variables influencing the complexity given by the training data. . . . .	97
6.2	Total complexity of the whole workflow for ranking quality evaluation for different ranking algorithms. . . . .	98
6.3	Evaluation of ranking quality of metric algorithms on the testing dataset. . . . .	104
6.4	Statistical comparison of different algorithms and their ranking quality results on the testing set. Row $i$ defines what algorithms had significantly worse results than algorithm $i$ . N stands for no and Y for yes. . . . .	105
6.5	Datasets in one of the clusters in the visualization of the distance (Figure 6.5). . . . .	105
7.1	Evaluation of the ranking quality of the trees produced by the GP algorithm on the testing dataset. . . . .	119
7.2	Statistical comparison of GP and previous algorithms and their ranking quality results on the validation set. Y stands for GP significantly worse than algorithm in the corresponding column, N stands for Not able to reject the null hypothesis (algorithms equally performing). . . . .	119
8.1	Evaluation of the ranking quality of the trees produced by the GP algorithm using antibloat operator and bootstrapping with coevolution on the testing dataset. . . . .	128

8.2	Statistical comparison of GP using antibloat operator and coevolution with bootstrapping and previous algorithms and their ranking quality results on the validation set. W stands for GP statistically worse than the algorithm in the corresponding column, I stands for inconclusive and B stands for statistically better. . . . .	128
-----	--	-----

# Acronyms

<b>AGR</b>	Agent-Group-Role.
<b>ARFF</b>	Attribute-Relation File Format.
<b>ASP</b>	Answer Set Programming.
<b>CBOW</b>	Continuous Bag-of-Word Model.
<b>CSP</b>	Constraining Satisfaction Programming.
<b>EPA</b>	Evolutionary Program-induction Algorithms.
<b>FIPA</b>	Foundation for Intelligent Physical Agents.
<b>GA</b>	Genetic Algorithm.
<b>GloVe</b>	Global Vectors for Word Representation.
<b>GP</b>	Genetic Programming.
<b>JADE</b>	Java Agent DEvelopment Framework.
<b>k-NN</b>	k-Nearest Neighbours algorithm.
<b>MAS</b>	Multi-agent system.
<b>ML</b>	Machine Learning.
<b>NFL</b>	No Free Lunch theorem.
<b>NLP</b>	Natural language processing.
<b>NSGA-II</b>	Non-dominated Sorting Genetic Algorithm II.
<b>OpenML</b>	Open Machine Learning repository.
<b>OWL-DL</b>	One of Web Ontology Language.
<b>PCA</b>	Principal Component Analysis.
<b>QBF</b>	Quantified Boolean Formula.
<b>RBF</b>	Radial Basis Function.

<b>RMSE</b>	Root Mean Squared Error.
<b>SAT</b>	Satisfiability Problem.
<b>SMAC</b>	Sequential Model-Based Algorithm Configuration.
<b>SVM</b>	Support Vector Machine algorithm.
<b>SW</b>	Smith-Waterman score.
<b>UCI</b>	Dataset Repository of University of California, Irvine.
<b>UPGMA</b>	Unweighted Pair Group Method with Arithmetic Mean.
<b>WEKA</b>	Waikato Environment for Knowledge Analysis.

# Attachments

All auxiliary materials including OpenML data dump and the text of the thesis are available at <https://github.com/jaksmid/dissertation>.