Ranking of Classification Algorithms in AutoML

Helena Graf

March 14, 2018
Version: Final

Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

Intelligent Systems Group (ISG)

Bachelor's Thesis

Ranking of Classification Algorithms in AutoML

Helena Graf

Matriculation Number: 7011643

1. Reviewer **Prof. Dr. Eyke Hüllermeier**
Department of Computer Science
Paderborn University

2. Reviewer **Prof. Dr. Axel-Cyrille Ngonga-Ngomo**
Department of Computer Science
Paderborn University

Supervisors Prof. Dr. Eyke Hüllermeier and Prof. Dr. Axel-Cyrille Ngonga-Ngomo

March 14, 2018

Helena Graf
Ranking of Classification Algorithms in AutoML
Bachelor’s Thesis, March 14, 2018
Reviewers: Prof. Dr. Eyke Hüllermeier and Prof. Dr. Axel-Cyrille Ngonga-Ngomo
Supervisors: Prof. Dr. Eyke Hüllermeier and Prof. Dr. Axel-Cyrille Ngonga-Ngomo

Paderborn University
Intelligent Systems Group (ISG)
Department of Computer Science
Pohlweg 51
33098 and Paderborn

Abstract

As the demand for machine learning functionality increases dramatically, automated machine learning (AutoML) aims to (partially) automate the creation of machine learning applications. Current AutoML approaches, however, often search for a single fitting learning algorithm for a new data set without explicitly considering how properties of the data set influence algorithm performances. Furthermore, in comparison to recommending a ranking of learning algorithms, this has the disadvantage that no alternatives are presented. Due to a certain inaccuracy of the prediction, these may have an equal or even better performance than the first suggestion regarding the examined data set.

In this thesis, the problem of predicting a ranking of classification algorithms based on properties of a data set is considered. To this end, two approaches are presented and compared. The first uses regression models to predict the accuracy of each classifier on a new data set, and then constructs a ranking from these predictions. Alternatively, a preference-based model directly learns and returns rankings. In an experimental study, it is shown that the regression-based approach performs superiorly to the preference-based alternative. However, the latter still outperforms a static ranking based on how often a classifier has performed best on previously considered data sets.

Abstract (German)

Da die Nachfrage nach Funktionalität im Bereich des maschinellen Lernens dramatisch ansteigt, hat automatisiertes maschinelles Lernen (AutoML) das Ziel, die Erstellung von Anwendungen im Kontext des maschinellen Lernens (teilweise) zu automatisieren. Aktuelle Ansätze dieser Art suchen jedoch oft nach einem Lernalgorithmus für einen neuen Datensatz, ohne explizit zu berücksichtigen, wie Eigenschaften des Datensatzes die Performanz der Algorithmen beeinflussen können. Außerdem hat dies im Vergleich zu dem Vorschlag eines Rankings von Lernalgorithmen den Nachteil, dass keine Alternativen präsentiert werden. Diese können allerdings, aufgrund der Ungenauigkeit der Vorhersagen, gleichwertig oder sogar bessere Genauigkeiten auf diesem Datensatz haben.

In dieser Arbeit wird das Problem betrachtet, ein Ranking von Klassifizierungsalgorithmen basierend auf den Eigenschaften eines Datensatzes zu erstellen. Dafür werden zwei Ansätze vorgestellt und verglichen. Der erste benutzt Regressionsmodelle, um die Genauigkeit jedes Klassifizierers für den neuen Datensatz vorherzusagen, und erzeugt dann basierend auf den Vorhersagen ein Ranking. Alternativ lernt ein Präferenzmodell direkt Rankings und gibt sie zurück. In einer experimentellen Studie wird gezeigt, dass der regressionsbasierte Ansatz eine bessere Performanz aufweist als die präferenzbasierte Alternative. Allerdings ist letztere noch einem statischen Ranking überlegen, welches darauf basiert, wie oft ein Klassifizierer die beste Performanz auf zuvor betrachteten Datensätzen hatte.

Acknowledgement

I would like to thank Prof. Dr. Hüllermeier for trusting me with this topic, and for his useful feedback on the matter. A very special gratitude goes out to Marcel Wever and Felix Mohr, without whose helpful insights and encouragement this thesis could not have been completed. Thank you for your explanations when I was struggling, and for pointing me in the right direction when I ran the risk of losing focus. My sincere thanks also goes to the proofreaders, who meticulously corrected my spelling and pointed out weak explanations. Last but not least, I am grateful for my family and friends for always supporting me and believing in me, and reminding me that I, too, need some sleep once in a while.

Contents

1 Introduction

The potential of big data is evident, and an increasing amount of information is collected and available for analysis - but this potential is not utilized. In a white paper, the International Data Corporation claims that in 2012, out of the 2.8 zettabytes (ZB) of available data only 3% were tagged as to enable further processing, and only 0.5% were analyzed [GR12]. A follow-up paper in 2017 projects that in 2025, 15% of the estimated 163 ZB of global data will be tagged, and approximately 3% analyzed [GR17]. While this is more optimistic, it still shows that there is a huge gap between the amount of data that could potentially be used and the amount of data actually available. This indicates that the demand of data to be analyzed cannot be covered by data scientists alone, and the process is not accessible enough to non-experts. It thus calls for automation of the process in a way that not much expertise in the field of machine learning is needed to gain insights about the collected data.

One of the most prominent machine learning tasks is classification: A class is assigned to an instance, for example clients of a bank may be either deemed creditworthy or not, based on factors like other existing credits or the job of the client. But selecting a fitting classifier for a new data set is difficult, since algorithm performances can vary substantially among data sets, and it is not feasible to simply apply a large number of them to empirically find a good match. For example, on a data set about the electricity prices in the Australian state New South Wales [Har99], the predictive accuracy for the Multilayer Perceptron[1] is 0.7887 [Cac17c]. The predictive accuracy of the Random Forest[2] algorithm on the same data set is 0.9236 [Cac17b], a much higher value. On a different data set, with the topic of vehicle silhouettes [Sie87], we get a predictive accuracy of 0.7979 for the Multilayer Perceptron [Cac17d], and 0.7518 for Random Forests [Cac17a], showing an advantage of the former on this data set[3]. So in each case, one would have picked a different algorithm in order to achieve the best results, and this example just illustrates the choice between two algorithms - in reality, the number of available algorithms is much larger. In general, this means that for a different data set, a different algorithm might yield the best performance.

[1] With standard hyperparameters (L:0.3,M:0.2,N:500,V:0,S:0,E:20,H:a).
[2] With standard hyperparameters (P:100,I:100,num-slots:1,K:0,M:1.0,V:0.001,S:1).
[3] Hyperparameters as above.

1.1 Problem Statement

Since there is no one best classifier for all data sets, it can be concluded that how well a classifier performs on a given data set is dependent on certain properties of the data set. Combined with the need for automated machine learning and the importance but difficulty of selecting a fitting classification algorithm, this calls for an approach that exploits properties of data sets to automatically suggest well-performing classifiers for a new problem.

Thus, the aim of this thesis is the implementation and evaluation of two different approaches to ranking classification algorithms based on past-performances of the algorithms and according to properties of the data set. The two approaches include one regression-based approach that breaks down the problem of predicting a ranking into predicting a single performance value for each algorithm and then ranking them accordingly, and a preference based approach that is concerned with learning the rankings directly. Especially in the case of regression-based ranking there is reason to believe that such a prediction is possible, as regression models have been used successfully to predict the performance of an algorithm dependent on the hyperparameter configuration [Egg+15].

1.2 Objectives

The thesis has two main goals. The first is to investigate the question whether it is possible to learn rankings of classification algorithms based on certain properties of data sets. This question is explored to the extent of empirically determining whether one of the proposed approaches has an advantage over a static baseline that does not use these properties for the prediction of a ranking. Furthermore, it is assessed how well both approaches perform in comparison to the correct ranking of classifiers for a data set. The second main goal is to compare the two different approaches among each other.

In order to conduct this evaluation, an implementation of the two approaches is needed. A third goal is thus the development of an AutoML-tool that, after a pre-processing phase, returns a ranking of classifiers for a given data set, using either regression or preference models for the prediction. The pre-processing phase constitutes of recording classifier performances on a number of data sets together with properties of the data sets to serve as training data, and training the regression models or preference model respectively.

1.3 Thesis Structure

The following paragraphs give an outline of the thesis structure by providing a brief summary of each chapter.

Chapter 2 - Fundamentals

First, some preliminaries are discussed. A brief overview of tasks from the field of machine learning which are relevant to this thesis is given, and methods used for evaluation are explained.

Chapter 3 - Approach

The next chapter follows up by describing the approach of this thesis for ranking classification algorithms. The two different proposed methods are contrasted. Following the details of the approach, the implementation thereof is presented.

Chapter 4 - Evaluation

The evaluation of the different ranking implementations is described by first clarifying the experimental setup used for the evaluation, followed by a discussion of the results.

Chapter 5 - Related Work

After the approach of this thesis has been laid out in detail, the scope is extended to related work in the area of AutoML in general and ranking of learning algorithms more specifically. Differences and similarities in the approaches are discussed briefly.

Chapter 6 - Conclusion

In the last chapter, the results which have been achieved are revisited with the goals in mind. Lastly, future work is outlined.

Fundamentals 2

The core topic of this thesis falls into the category of machine learning. Hence, first the concept of machine learning and the tasks related to the approach of this thesis are explained. Second, means used for evaluation of the approach are discussed.

2.1 Machine Learning

The core of machine learning is learning algorithms, which are used to induce a general model from a set of data samples. The concept of machine learning is explained here with the help of an example.

Suppose an aspiring gardener wants to learn how to distinguish between different species of iris plants, a genus of ornamental plants with colorful flowers. More specifically, the focus lies on the three species iris versicolor, virginica and setosa. In order to do so, the gardener has observed four different *features*, namely the length and width of their petals and sepals, for a number of different individuals of which they know the species, which is illustrated in Table 2.1. The goal of the gardener is then to learn how to distinguish the species based on these features, that is to derive a *model* from the data that will predict which species (out of the considered three) an unknown iris plant is. The gardener decides to build a decision tree from the data with forks on the basis of feature values, so that they can determine the species of the plant without much calculation. They observe that all iris setosa plants from the sample have a petal width of ≤ 0.6 cm, and that of the plants with a petal width of > 0.6 cm, most plants with a petal width ≤ 1.7 cm are of the iris versicolor species. The remaining plants with a petal width of > 1.7 cm are mostly iris virginica plants. While this model (illustrated in Figure 2.1) will not correctly predict the species of all iris plants, the gardener settles with the approximation they have found.

In formal terms, the gardener is searching for an unknown *target function* $f : X \to Y$ from the input space X to the output space Y [ALM12] that represents an ideal way of identifying iris species. X denotes the possible inputs, in this case all combinations of the four features that have been defined, whereas Y are the outputs, or the target feature of the data set, here the species of iris plant. The examples that the gardener recorded are samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ from f such that $f(x_i) = y_i$.

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
5.0	3.5	1.6	0.6	Iris setosa
5.0	3.4	1.6	0.4	Iris setosa
5.6	3.0	4.5	1.5	Iris versicolor
6.7	3.1	4.4	1.4	Iris versicolor
5.9	3.2	4.8	1.8	Iris versicolor
7.2	3.0	5.8	1.6	Iris virginica
5.9	3.0	5.1	1.8	Iris virginica
6.9	3.1	5.1	2.3	Iris virginica

Tab. 2.1.: Features of different iris plants. This is an excerpt from the well-known iris data set [36].

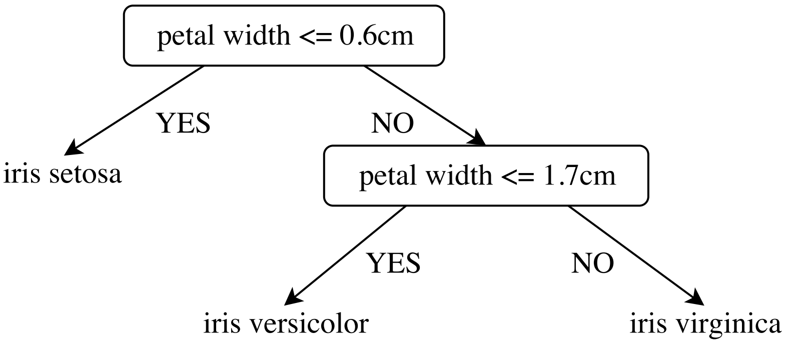

Fig. 2.1.: The decision tree constructed as a simple model to classify iris plants in the gardening example.

These tuples form a data set D which is one of all possible data sets $\mathbb{D}$ for the problem. The gardener then choose a specific approximation $g : X \rightarrow Y, g \approx f$, the decision tree they built, from the hypothesis space H, which in this case consists of possible decision trees, and in general contains candidate formulas considered by the learning algorithm. The learning algorithm itself thus maps a data set to a hypothesis from the hypothesis space and can be defined as $A : \mathbb{D} \rightarrow H$. An overview of this learning problem is provided in Figure 2.2.

2.1.1 Supervised Learning

The problem described above has some additional properties besides being a machine learning problem in general. Firstly, it falls in the category of supervised machine learning. Supervised learning is one of the three main learning paradigms of machine learning, besides reinforcement learning and unsupervised learning [ALM12]. In the context of this thesis, the latter will be neglected as only supervised learning is used. It is characterized by the fact that the learning algorithm is provided with a set of inputs *together* with the outputs, which could be viewed as a kind of teacher,

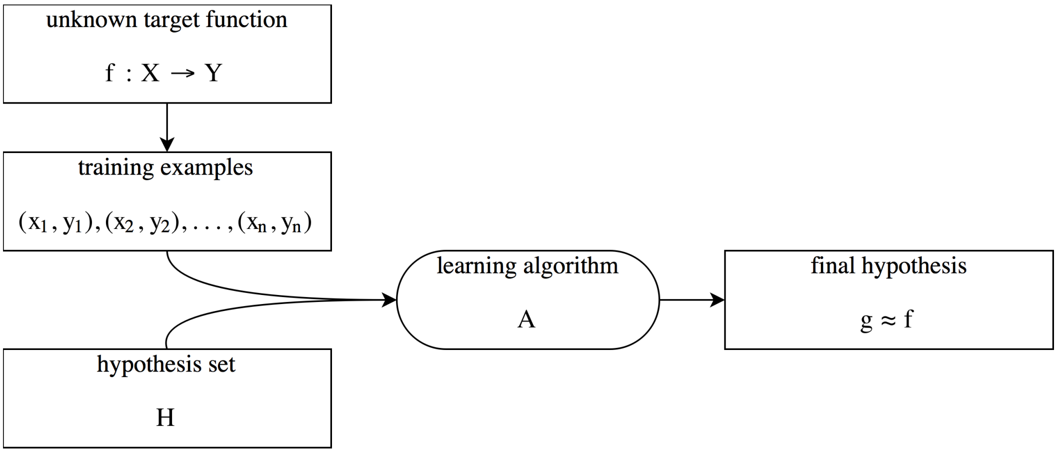

Fig. 2.2.: The learning problem, adapted from [ALM12].

or supervisor, explaining expected results to the learner. In the other cases, no such detailed feedback is provided to the learner.

2.1.2 Classification and Regression

In addition to falling into the category of supervised learning, the problem is a classification problem. The nature of the prediction in the example is to assign one of the three classes, also called labels, 'iris setosa', 'iris virginica' and 'iris versicolor' to a new plant. Formally, this means that the output space Y of the target function consists of a finite set of labels $\{y_1, \dots, y_n\}$, such that each instance of the data set is associated with a label $y_i \in Y$. In this case, the output space is defined as $Y = \{'iris\ setosa', 'iris\ virginica', 'iris\ versicolor'\}$.

Apart from classification, another important category of machine learning problems are regression problems. In the case of regression, the output space of the target function is no longer finite, but instead consist of real numbers. To reconsider the gardening example, the gardener might want to predict the height of a plant on the basis of the same features as used above. Likewise, they would have to observe a number of plants to gather feature values together with the expected target value, which instead of the species would then be the height of the plant in centimeters, that is $Y = \mathbb{R}$.

2.1.3 Preference Learning

Preference Learning is a relatively new subfield of machine learning [FH10]. It is dedicated to the problem of learning how to rank, the precise definition of what this encompasses being defined by the specific task one considers. Out of the three main tasks of preference learning, namely label ranking, instance ranking and object ranking, label ranking is the relevant one in the context of this thesis. But before going into the details of label ranking, it first has to be explained what is meant with a ranking in this context and how it is distinguished from the concept of an ordering, as it is easily confused.

Ranking and Ordering

To clarify the meaning of ranking and ordering, we return to the gardening example. After spending some time studying the different flowers, the gardener has realized they prefer certain iris species to others. They express their preference for a plant in a score, and labels all the plants in no particular order (see Tab. 2.2). An *ordering* then is the result of simply ordering the labels in decreasing order of their respective score. It thus can also be given implicitly by just ordering the original classes, the flower names, as such being a permutation of the labels. A *ranking*, on the other hand, assigns a fixed position to each label; for instance the rank of 'iris virginica' will always be represented by the number in the first position in the ranking, that is the number in that position denotes the *rank* of the label, which is in turn defined as the label's position in the ordering. Therefore, the values in the example in Table 2.2 result in the ranking $[2, 3, 1]$, whereas the ordering is $[3, 1, 2]$. It is notable that these can be the same, and are easily convertible. However, in the remainder of this thesis mostly the term ranking is used to identify both ranking and ordering interchangeably, especially as usually only the term 'ranking' is used.

Species		iris virginica	iris versicolor	iris setosa	
Label	[	1,	2,	3	]
Score	[	0.17,	0.12,	0.89	]
Ranking	[	2,	3,	1	]
Ordering	[	3,	1,	2	]

Tab. 2.2.: Ranking and ordering in direct comparison.

Label Ranking

Similar to classification, in label ranking there is a finite set of labels $Y = \{y_1, \dots, y_n\}$, but it is not the output space. Instead, Fürnkranz and Hüllermeier define the target function for label ranking as $f : X \to S_Y$ where the output space S_Y contains all permutations over the set of labels Y [FH10]. To come back to the topic of rankings and orderings, it becomes clear now that even though it is called label ranking, the labels are actually ordered. An example of label ranking could be, staying in the domain of gardening, to learn the preferences of different gardeners towards iris plants. A label ranking data set used to learn this connection could, for example, map the favorite color and height of a gardener to a ranking of the three iris plants.

2.1.4 Meta Learning

The previous discussion of machine learning problems has conceptually been about the fitting of different machine learning algorithms to problems in the form of data sets. In the context of meta learning regarding machine learning, it is researched whether it is 'possible to learn about the learning process itself, and in particular that a system could learn to profit from previous experience to generate additional knowledge that can simplify the automatic selection of efficient models summarizing the data' [Bra+08]. That is, the original algorithm selection problem [Ric76] is viewed as a learning problem itself that is to be tackled.

The approach of this thesis clearly falls under that category, as it is trying to learn whether a connection exists between properties of a data set, that is its meta features, and the performance values of certain classification algorithms, and how well it can be exploited by regression and preference models respectively, if it exists. The benefit from meta learning in the context of machine learning is that learning how properties of data sets influence the learning process of the classifiers could save much work. Often, a large effort is put into the fitting of an algorithm to a specific data set, but this knowledge is not transferred to other problems that may be of the same nature.

2.2 Evaluation

After having discussed the preliminaries that the theoretical approach of this thesis is built upon, techniques used for evaluation are briefly reviewed. This includes the used methodology of splitting a data set into training and test data to estimate the performance of learning algorithms, and the measures that will be used in the evaluation of the different ranking methods. These include means of comparing the predicted rankings with actual rankings as well as a statistic that reveals whether an advantage regarding one measure can be viewed as significant.

2.2.1 Estimation Procedures

To find out how well a classifier performs, it first needs to be decided which measure to consider. There are a few possibilities for this, however in this thesis the term performance measure refers to the predictive accuracy of a classifier, that is the percentage of classes that a classifier correctly identified in a given data set. But since using the whole data set to train a classifier (build a model) and consequently test it, would skew the analysis - the classifier has already ‘seen’ these examples - a method for dividing data sets into train and test sets is needed.

A popular approach is using a stratified split, for example splitting the data into 70% training and 30% test data, with the important aspect that both subsets have equal (or at least very similar) ratios of classes [Reference Missing]. The learning algorithm is then trained with one part of the data (usually the larger one), and its performance tested on the other. To get more stable results, often the process is repeated: The data is divided in k folds. Then, the learning algorithm is repeatedly trained on k-1 folds, k times, and tested on the left out subset each repetition. Results are averaged. This process is called k-fold (stratified) cross-validation [Reference Missing]. A variant of this is Monte Carlo cross-validation, where after each time the algorithm has been trained and evaluated, a new split is randomly computed [Reference Missing]. This especially causes the fact that the number of folds does not dictate the ratio of the split; one could for example have 5-folds Monte Carlo cross-validation with a 70%/30% train/test split.

An alternative to using a stratified split is using a method called leave-one-out. This method essentially is, for a data set with n instances, n-fold cross-validation (not stratified, naturally). This means that the algorithm is trained on n-1 instances and evaluated on the instance that has been left out. Since this method is executed n times on a data set with n instances, whereas for example 10-fold (stratified) cross-validation is always only executed ten times, it is computationally more expensive. This is especially the case for larger data sets, because generally, the time for training a classifier is increased with the number of training examples. However, as the number of training examples increases, the estimated accuracy of the classifier approaches the accuracy that it would achieve on the entire population (assuming that the samples are independent and not biased), making it desirable to have as many training examples as possible. Therefore, leave-one-out tends to be less biased than the (usually more negatively biased) variant of using k-fold cross-validation with a relatively small k [Koh+95]. On the other hand, this leads to a higher variance of the estimates. So apart from considering the computational expense when choosing k, one has to think about the bias-variance trade-off.

2.2.2 Kendall Rank Correlation Coefficient

To evaluate the quality of the prediction that a ranker produces, a statistic is needed to compare a predicted ranking with the correct ranking. For this, the Kendall

Rank Correlation Coefficient will be used here, as it expresses the similarity of two rankings.

The Kendall Rank Correlation Coefficient, also known as Kendall's tau, is a measure of association between two rankings of the same items [Ken38]. For two independent random variables X and Y and observations of values $(x_1, x_2, ..., x_m)$ for X and $(y_1, y_2, ..., y_n)$ for Y with unique x_i and y_i respectively, the coefficient is based on comparing pairs of observations. If for two pairs (x_i, y_i) and (x_j, y_j) both $x_i < x_j$ and $y_i < y_j$ (or the opposite) holds, they are viewed as concordant. If $x_i < x_j$ but $y_i > y_j$ (or the opposite), they are viewed as discordant. Other cases are not considered. The simplest version of the statistic, called T_A is then defined as

$$T_A = \frac{n_c - n_d}{n_0},$$

with
n_c, the number of concordant pairs,
n_d, the number of discordant pairs, and
$n_0 = n * (n-1)/2$.

To account for ties in the rankings, a second statistic, T_B is defined as

$$T_B = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}},$$

with
n_c, n_d, n_0 as above,
$n_1 = \sum_i t_i(t_i - 1)/2$,
$n_2 = \sum_j u_j(u_j - 1)/2$,
t_i, the number of ties observed for X in the ith position of the ranking
u_j, the number of ties observed for Y in the jth position of the ranking

Thus, the value of the tau (for both versions) lies in the interval $[-1, 1]$, with 1 indicating a perfect agreement of rankings, -1 for a perfect disagreement, and 0 corresponding to no correlation. A high value is therefore desirable in this context, as it indicates that the prediction of the ranker agrees with the actual ranking of algorithms.

For illustration, once again the gardener example is used. Previously, the gardener has expressed their preference for the three different iris species of iris plants with the ranking [2,3,1] (see Tab 2.2). Suppose a different gardener instead has the preferences [3,2,1], and one wants to know to what degree they agree. To compute the Kendall rank correlation coefficient, first, the first position of both rankings would be the compared with the second positions, that is (2,3) with (3,2). This pair is discordant. The next pair, (2,3), (1,1), is concordant, and the last pair, (3,2), (1,1) is concordant as well. As there are no ties, this results in a correlation of $\frac{2-1}{3} = \frac{1}{3}$, indicating a slight agreement.

2.2.3 Loss

While the Kendall rank correlation coefficient is an indication as to what degree a predicted ranking agrees with the correct ranking of classifiers for a certain data set, it does not convey the difference in performance experienced by the user when choosing the first of the recommended algorithms compared to the performance of the actually best algorithm. This difference can be expressed in a measure called loss [LBV12], which is defined as the difference of the performance of the actual best algorithm and the performance of the first recommended algorithm. Therefore, the loss in performance L a user experiences on a data set when following a ranker's recommendation by choosing the first algorithm in the returned ranking can be expressed as

$$L = |p_{best} - p_{first}|$$

with
p_{best}, the performance of the best algorithm on the data set,
p_{first}, the performance of the first recommended algorithm on the data set.

Taking the absolute value of the difference is necessary here, since the 'best' performance value of a classifier does not always need to be the largest value. When choosing the error rate of a classifier on a data set as the performance value, for example, the best possible value is 0, and in general, all classifiers besides the best one will have an equal or higher performance value. When choosing the percentage of correctly classified instances, though, the opposite is true, but the loss always needs to be non-negative. This is ensured by the taking the absolute value of the difference.

However, the ranker recommends a *ranking* instead of a single algorithm in order to allow a user (or automated tool) to test a few of the top-ranked algorithms, which therefore also needs to be taken into account when thinking about loss. This is done here by computing a measure called best three loss (B3L), which denotes the loss resulting in the selection of the best of the first three recommended classifiers in comparison to the actual best-performing classifier in the considered set of classifiers.

Algorithm	A_1	A_2	A_3	A_4
Error Rate	0.1	0.7	0.3	0.4
Rank	2	4	1	3

Tab. 2.3.: An exam.

Similar to the definition above, the loss in performance a user experiences on a data set when following a ranker's recommendation by choosing the best of the first three recommended algorithms can be expressed as

$$B3L = |p_{best} - max(p_{first}, p_{second}, p_{third})|$$

with

p_{best}, p_{first} as above,

p_{second}, the performance of the second recommended algorithm on the data set.

p_{third}, the performance of the third recommended algorithm on the data set.

It is notable that this definition is only valid if the 'best' performance value is the largest possible value, as for example is the case for the percentage of correctly classified instances, as explained above. When choosing a measure like the error rate, where the lowest value is the best value, the maximum in the equation needs to be substituted by a minimum, since the best of the first three recommended algorithms then is the one with the *lowest* performance value.

To demonstrate how the loss measures work, we return to the gardener example from the previous sections. Previously, the gardener has just chosen a tree-based model (see Fig. 2.1) and hoped that this was a good approximation. But they conclude that the error rate of this model is too high, and a better approximation is needed. So instead, they now choose to use an AutoML tool that will return a ranking of classification algorithms for their data set of flowers, a ranker. The ranker has four available algorithms A_1, A_2, A_3 and A_4, and has produced the ranking depicted in Tab. 2.3. When the gardener follows this recommendation, the loss L experienced is $L = |0.1 - 0.3| = 0.2$, and $B3L = |0.1 - min(0.3, 0.1, 0.4)| = 0$. So by trying the best three recommended algorithms, the gardener finds the best of the four available algorithms. But the example also shows, that the loss is always viewed only regarding the algorithms that are contained in the ranking, not all available algorithms - the algorithms A_1 to A_4 are only compared among each other and not, for example, with the tree model constructed earlier.

2.2.4 Mann-Whitney U Test

The measures previously presented can indicate if one of the ranking alternatives is superior to another regarding that measure, but one cannot conclude from those results alone whether this advantage is significant. The question here is whether the null-hypothesis that two distributions belong to the same population can be rejected. This can be determined with the help of the Mann-Whitney U, a non-parametric hypthesis test [MW47]. The Mann-Whitney U is computed by, for two distributions one wants to analyze, counting how many times each value from one distribution is lower than another value from the other distribution. If the U is very high (or low, depending on the hypothesis), the null hypothesis can be rejected. Whether this significance level is reached is expressed by the p-value: for a p-value below 0.05, the null hypothesis can be rejected. Therefore, if two distributions, for example regarding the loss values of two ranking approaches, suggest that one is superior to the other, and the p-value for the Mann-Whitney U is $<= 0.05$, this suggests that the difference is statistically significant.

Approach

3

In this chapter, the two alternatives considered in this thesis to tackle the problem of ranking classification algorithms regarding properties of a data set are explained. First, the theoretical approach is presented. Then, implementation details are discussed.

3.1 Theory

In general, the aim is to predict a ranking of algorithms based on properties, or meta features of a data set. This means, that as training data for a ranking approach (ranker), a number of performance values of classifiers need to be recorded on data sets in a first pre-processing phase, for which also meta features are to be computed. The table of this collected data is illustrated in Table 3.1, for n meta features, k classifiers and m data sets. This approach to ranking is realized here, as pointed out before, in a regression-based and preference-based variant, which are explained in detail and compared briefly in the following sections.

MF 1	MF 2	...	MF n	PV 1	PV 2	...	PV k
v_{11}	v_{12}	...	v_{1n}	p_{11}	p_{12}	...	p_{1k}
v_{21}	v_{22}	...	v_{2n}	p_{21}	p_{22}	...	p_{2k}
...	...	...	...	...	...	...	...
v_{m1}	v_{m2}	...	v_{mn}	p_{m1}	p_{m2}	...	p_{mk}

Tab. 3.1.: Training data for the rankers.

3.1.1 Regression-Based Ranking

One possibility to derive a ranking of classifiers from this information when given a new data set is to use regression models. The idea is to use separate regression models to predict a performance value for each classifier, and to subsequently derive an ordering from these predictions. The training of the ranker is done by splitting the data set compiled beforehand (see Tab. 3.1) into separate data sets that each contain all meta features for all data sets, but only the performance values of one classifier, as illustrated in Table 3.2. The target feature on these individual data sets, the performance value of the classifier, is a numeric value, and thus, a regression model is trained on each of them. All regression models therefore try to learn the target function *meta features→performance value* for their respective classifier. After the regression models have been trained for each classifiers, the training of the ranker is finished. For a query instance, here represented by a new data set, the first step is to compute the meta features. Afterwards, these are fed into each regression model and the predicted performance value for each classifier is saved. In a second step, the classifiers are ordered according to the predictions. This generation of a ranking is presented in Figure 3.1.

Meta Feature 1	...	Meta Feature n	Performance Value 1
v_{11}	...	v_{1n}	p_{11}
v_{21}	...	v_{2n}	p_{21}
...	...	...	...
v_{m1}	...	v_{mn}	p_{m1}

Meta Feature 1	...	Meta Feature n	Performance Value 2
v_{11}	...	v_{1n}	p_{12}
v_{21}	...	v_{2n}	p_{22}
...	...	...	...
v_{m1}	...	v_{mn}	p_{m2}

⋮

Meta Feature 1	...	Meta Feature n	Performance Value k
v_{11}	...	v_{1n}	p_{1k}
v_{21}	...	v_{2n}	p_{2k}
...	...	...	...
v_{m1}	...	v_{mn}	p_{mk}

Tab. 3.2.: How the regression ranker processes the training data. Performance data for each classifier are in a separate table, which are then used to train separate regression models.

3.1.2 Preference-Based Ranking

The second possibility considered in this context is using preference learning to predict a ranking. Each instance of the data set generated beforehand (see Tab. 3.1) contains meta feature information for the considered data set and performance values of all classifiers. The performance values can be converted to preference information by sorting the corresponding classifiers by their performance values for each data set, similar to the prediction step of the regression alternative, which is the first step of building the ranker. This leads to an ordering of classifiers associated with each instance, and implies that the preference learning task at hand is label ranking. In this case, in contrast to the regression-based approach, there exists only one model, a label ranking model. To complete the training of the ranker, this model is built with the training data that has been converted to a label ranking data set in the previous step. To make predictions, the computed meta data for a new data set is fed to this model, which consequently returns an ordering of classifiers, which is

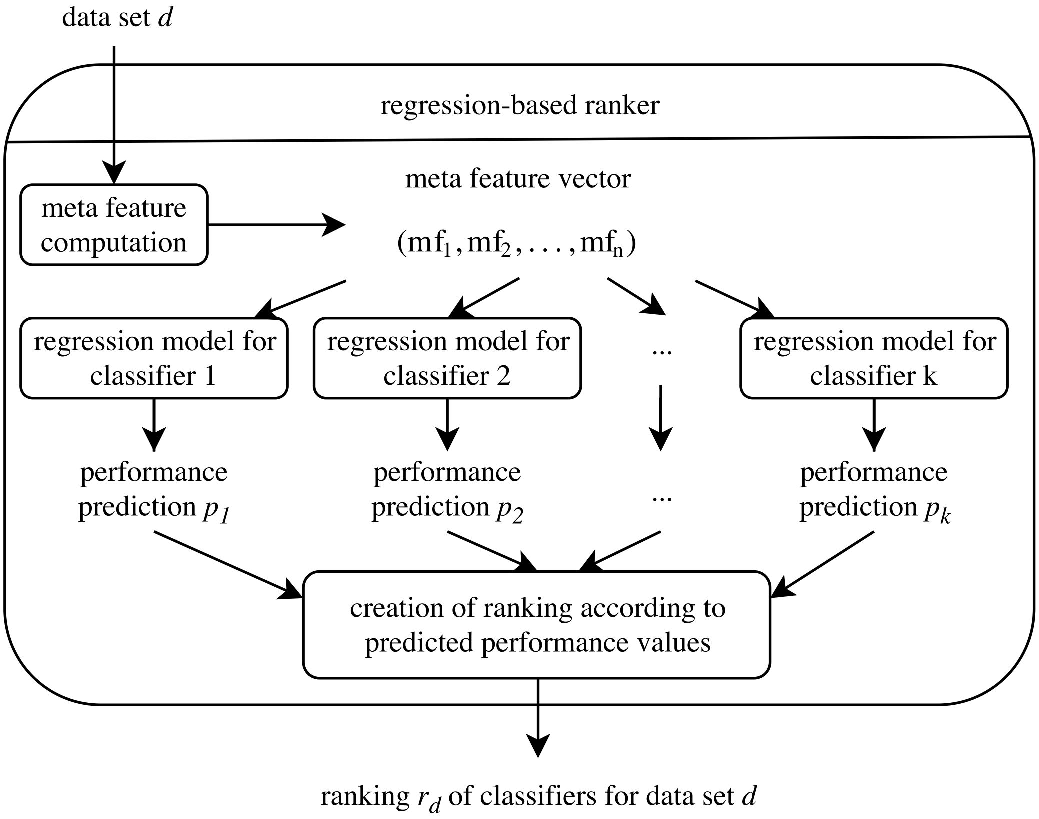
data set d
regression-based ranker
meta feature computation
meta feature vector
$(\mathrm{mf}_1, \mathrm{mf}_2, \ldots, \mathrm{mf}_n)$
regression model for classifier 1
regression model for classifier 2
...
regression model for classifier k
performance prediction p_1
performance prediction p_2
...
performance prediction p_k
creation of ranking according to predicted performance values
ranking r_d of classifiers for data set d

Fig. 3.1.: An overview of how a regression-based ranker constructs a ranking.

depicted in Figure 3.2. Thereby, it attempts to learn the mapping from meta features to an ordering of classifiers directly.

MF 1	MF 2	...	MF n	Ranking
v_{11}	v_{12}	...	v_{1n}	r_1
v_{21}	v_{22}	...	v_{2n}	r_2
...	...	...	...	...
v_{m1}	v_{m2}	...	v_{mn}	p_m

Tab. 3.3.: How the preference ranker processes the training data. The ranker is trained with a table containing meta features and a ranking of classifiers for each data set.

3.1.3 Comparison

While both of the ranking approaches above are used for the same purpose of ranking classification algorithms, they differ in some aspects that are interesting to point out. First, they theoretically do not require the same kind of training data. While the preference ranker only needs rankings, the regression ranker requires the exact performance data of all considered classifiers on the data sets. Although here it does not make a difference, since the performance values are computed anyways as the regression ranker utilizes them, in cases where only rankings of classifiers for a data

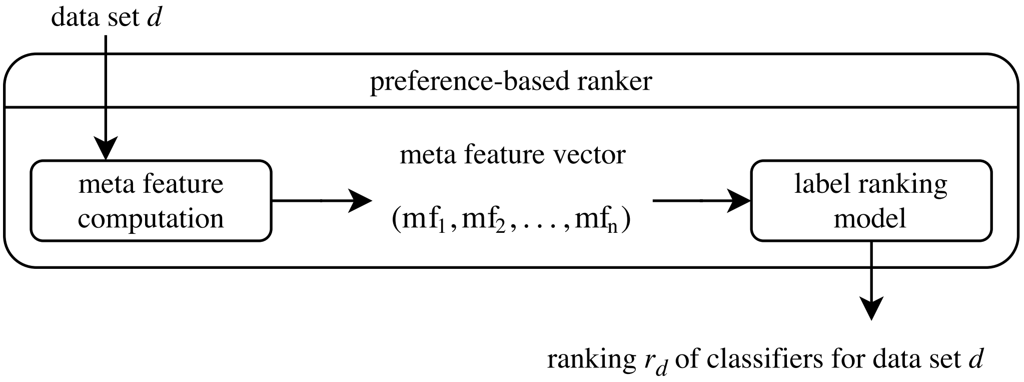
data set d
preference-based ranker
meta feature computation
meta feature vector
$(\mathrm{mf}_1, \mathrm{mf}_2, \ldots, \mathrm{mf}_n)$
label ranking model
ranking r_d of classifiers for data set d

Fig. 3.2.: An overview of how a preference-based ranker constructs a ranking.

set are known or performance values are estimated in a way that is biased, but has the same bias for all classifiers, so that the ranking is not influenced, this may be an advantage. Furthermore this leads to the curios case that training examples for the regression ranker are not directly samples from the target function the ranker learns. The target function for the regression ranker (as for the preference ranker) maps meta features of a data set to a ranking. However, for the regression ranker, the training examples have to include the actual performance values, meaning that they are of the form (*meta features*, *performance values*).

Also, the approaches behave differently regarding the addition (or deletion) of a data set or classifier. If a classifier is added, for the regression-based approach, a regression model needs to be added to the ranker. Therefore, performance values for the classifier for the data set would need to be available or have to be computed anew. For the preference-based approach, however, the whole label ranking model, therefore the whole ranker, would need to be trained again, as for every data set, the ranking of classifiers has changed. Here a regression ranker conceptually has an advantage. The deletion of a classifier is much simpler: for the regression-based ranker, one regression model has to be deleted, which does not affect the other models, and in the case of the preference ranker, in the rankings produced by the label ranking model the unwanted classifier can just be removed from the rankings.

For the addition of a data set, the situation is unfavorable for both rankers. Since a new training example would be added to each regression model, they all need to be trained again on the now extended training data. For the preference ranker, likewise a new training example is added to the training data, and a new model has to be trained. The same is the case for the deletion of a data set.

Regarding training and prediction, conceptually, the preference ranker has the advantage that it both only needs to build one model and to have one model make a prediction for a new data set. On the other hand, the regression ranker needs to build as many models as there are classifiers in consideration, and also have as many models predict a performance value for their respective classifiers, which then also have to be turned into a ranking. In general, this indicates a better scalability of the preference-based approach regarding the number of classifiers (unless they are to be added dynamically) and data sets, although of course this also depends on the specific algorithms used.

3.2 Implementation

The implementation of the rankers themselves is relatively straightforward with the theory of the previous sections in mind. However, since the goals for his thesis includes the development of an AutoML tool that returns a ranking of classification algorithms when given a data sets, the scope of the implementation goes beyond the ranking algorithms, including also pre-processing steps and means of evaluation for the rankers, relevant parts of the implementation are described in the following paragraphs; it is notable that all implementations were carried out in Java, particularly Java 8.

As mentioned before, the implementations of the ranking approaches are relatively close to the theory and will therefore not be described in detail, except for a few notable parts. First, since the tool relies on external libraries for the learning algorithms, these need to be pointed out. For the implementation of the regression models, WEKA was used, a popular libary of machine learning algorithms that includes algorithms for classification, regression, and provides help in the evaluation of these learning algorithms [Hal+09]. Regarding the preference algorithms, the jPL framework was used [Int17]. The jPL framework is a Java framework for the evaluation of preference learning algorithms, it implements several tasks from the context of preference learning, including label ranking, and also provides a representation for data sets that contain label ranking information.

Furthermore, so far, the descriptions of the 'performance measure' used for the classifiers, that is the measure according to which the classifiers are ranked, for example the error rate or predictive accuracy, have been kept non-specific. Likewise, although the measure used in the evaluation in this thesis is predictive accuracy, the implementation has been kept general to allow any measure to be used with the rankers, as long as it is numeric.

Another fact worth mentioning regarding the implementation of the rankers is that both the regression-based approach and the preference-based approach are implemented in a way that abstract superclasses handle the data set conversions and other

```java
@Override
protected void buildRegressionModels
                        (Map  Classifier, Instances> train)
                        throws Exception {
        regressionModels = new HashMap<Classifier,Classifier>();
        for (Classifier classifier : train.keySet()) {
                RandomForest forest = new RandomForest();
                forest.buildClassifier(train.get(classifier));
                regressionModels.put(classifier, forest);
        }
}
```

Fig. 3.3.: A source code excerpt for a ranker based on the random forest algorithm.

necessary tasks, so that it is very easy to add a new ranker implementation. For example, to create a new ranker that uses random forest as a regression model for each classifier, it is only necessary to override a method called `buildRegressionModels` which has the classifiers mapped to the training data necessary for the regression model of each classifier as a parameter and is responsible for the generation of the regression models. How this method is implemented in the case of random forest is illustrated in Figure 3.3. Here we can see that the regression model that has been acquired by training on the provided data is saved together with the classifier for which it predicts performance values. Later, this model is used to predict the performance value of that specific classifier.

Last, while the computation of the meta features is included in the workflow of a ranker in the theoretical explanations of the previous sections, meta features are calculated separately in the implementation since it is a repeating task, to speed up the evaluation of classifiers. The list of meta features used is taken from OpenML [Van+13], a website that not only provides a large number of openly accessible data sets, but also records performance values of learning algorithms on these data sets, and features properties of data sets. Thus the meta features were calculated with the help of an implementation provided by OpenML in its Evaluation Engine [Int18]; the full list of meta features is included in Table A.1 in the Appendix. Like the choice which classifier and performance measure to use, the choice of which meta features to be used in the implementation can be changed flexibly.

In addition to this ranking functionality, methods for the evaluation of both classifiers and rankers are implemented. Although WEKA offers capabilities for the evaluation of classifiers, additional functionality had to be added, as for example the desired estimation procedure of stratified Monte Carlo cross-validation was not supported by WEKA at the time of the implementation. Furthermore, since for the evaluation of the rankers (to build the table depicted in Tab. 3.1), many classifiers had to be evaluated

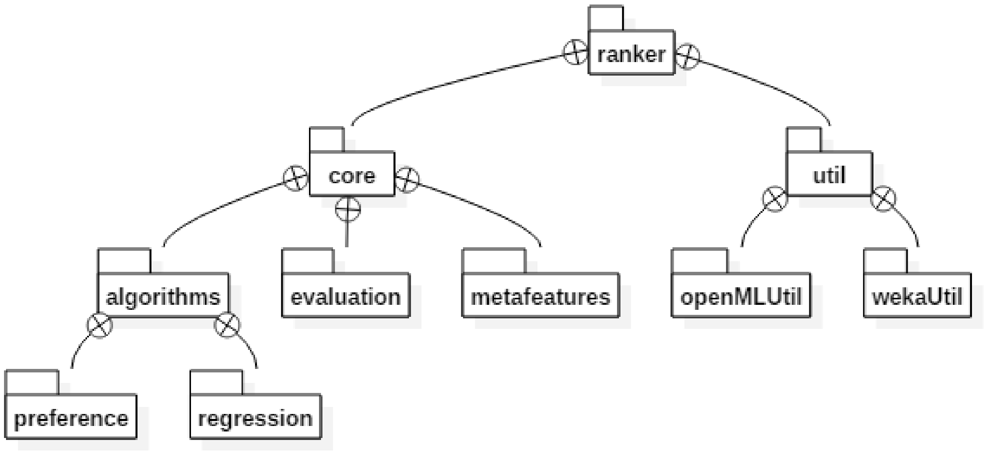

Fig. 3.4.: An overview of the package structure of the tool.

on a large number of data sets, an efficient way to execute this task was needed. This was realized by enabling the tool to take a number of jobs, consisting of classifiers and data sets, as command-line parameters, which enables execution for example on a data mining cluster. For the evaluation of rankers, the functionality needed to be built without the help of existing frameworks, and was realized in a similar way to the implemented evaluation of WEKA classifiers. It was also implemented such that after a ranker has been trained once, it can be evaluated regarding a number of different measures, to speed up the evaluation process.

Apart from the functionality described above, the implementation includes some useful recurring functions from the context of handling data sets and classifiers in the context of WEKA and OpenML. To give a general idea how the tool is structured, an overview of the package structure is given in Fig. **??**

Evaluation 4

This chapter is dedicated to the evaluation of the proposed ranking approaches. First, the oracle and baseline against which the rankers are compared are explained. Second, the experimental setup, that is conditions under which the evaluation was conducted, is set forth. Last, results are presented and discussed.

4.1 Baseline and Oracle

In order to evaluate the implemented rankers, their outputs are compared to a perfect output generated by an oracle. The oracle is implemented as a ranker that, after having been trained on a meta data-performance data set, returns correct rankings when queried for any instance of that data set. The correct ranking is implied by the performance values (highest first, as the measure used here is predictive accuracy), with ties being handled in a sequential manner, i.e. for the same performance values, the algorithm that was encountered first when constructing the ranking will be ranked before all others with the same value.

As rankings returned by this oracle represent the ideal solution, the predicted rankings are compared with them through the measures presented in chapter 2. These include the Kendall rank correlation coefficient, performance loss and B3L regarding classifier performance. The measure chosen for the performance values here is the *predictive accuracy* of a classifier, meaning the number of correctly classified instances in a data set. For all three measures, it is computed whether any difference in performance is significant, by means of the p-value for the Mann-Whitney U. The regression-based rankers pose a special case: internally, they predict a performance value for each classifier, which therefore can be compared to the actual performance values to show how well they are predicted. This is done by computing the root mean square error between the predicted and actual performance values. In summary, it is desirable for the rankers to come as close as possible to the correct ranking, which is reflected by a high rank correlation and a low loss and root mean square error.

Furthermore, we are interested in the question as to what degree knowing about the properties of a data set influences the quality of rankings. Therefore, a ranker that is agnostic to the meta features of a query data set, meaning that it will always return the same ranking for any data set, is used as a baseline. More specifically, a best algorithm strategy that iteratively determines a ranking by counting the number of data sets where an algorithm is the best choice, is implemented. That means the first algorithm in the best algorithm ranking is the classifier that performs best on most data sets, the second is the best on most data sets when only rankings excluding the first algorithm are considered, and so forth. This baseline is evaluated in the same way as the preference rankers; as it does not predict performance values, the root mean square error of predicted performance values cannot be computed. Ideally, rankings returned by the preference and regression based rankers should then be statistically significantly better than the static baseline, as this would indicate a connection of certain meta features and a performance-induced ranking of classifiers for a data set.

4.2 Experimental Setup

In the following paragraphs, it is briefly summarized under which conditions the experiments which led to the results discussed in the next section were observed. This includes which classifiers and data sets were chosen, and the specifications of the machines the evaluation was executed on. It is described how the evaluation was carried out, and which specific regression-based and preference-based implementations were used.

The classifiers considered in the rankings were the learning algorithms implemented in WEKA that are fit for classification, excluding meta and ensemble methods. The full list of the 22 classifiers can be gathered from Table 4.7. They were all used in their default configurations as specified by WEKA.

Data sets for the analysis were gathered from OpenML. From all data sets considered as ‘active’ on OpenML, which yields 2050 data sets that have been approved with no severe issues found for them so far [nd], all data sets that comply with the constraints of the learning problem at hand were selected. Only data sets in the .ARFF format with a defined, nominal target feature which were not specifically designed for streaming analysis[1] were considered, which resulted in a reduced selection of 812 instances. Since this amount of data sets hindered the evaluation considerably, large data sets with more than 1000 instances or 100 features were removed as well, leading to a final selection of 448 data sets. Even though evaluation was only carried out on the shorter list, both the list of 812 and 448 data sets are included in the supplementary material for completeness.

The performance data of the classifiers for the selected data sets was generated on a linux cluster with nodes consisting of two Intel(R) Xeon(R) E5-2670 processors running at 2.6GHz, 64GB RAM. For the computation of a single performance value of a classifier on a data set, two CPU cores and 16GB RAM were used, with a timeout of eight hours. Up to 150 of such processes were executed in parallel. For each of the classifiers, the predictive accuracy (the percentage of correctly classified instances in the data set) was recorded on each data set by means of five times stratified Monte-Carlo cross validation with a 70/30 split in training and test data. The stratified split was generated with the help of JAICore, 'A Java library for infrastructural classes and basic algorithms in the area reasoning, search, and machine learning' [Reference Missing]. When a timeout occurred or the evaluation failed otherwise, the predictive accuracy was set to zero.

All other experiments were carried out on a Windows Machine running Windows 10 Education (version 1709) with a Intel(R) Core(TM) i5-4200U CPU running at 1.60GHz (2.30GHz max) with 12 GB RAM. It is notable that especially the time-related results are to be viewed in regard to this setup. No timeouts were used in this evaluation, and the rankers were evaluated on the training data by means of leave-one-out estimation, that is for the n data sets for which performance values were recorded, they were trained with the meta data - performance examples for n - 1 data sets, and queried for the left-out data set. As n = 448 in this case, the detailed values for each data set are not included in the discussion of the results. They can, however, be found in the supplementary material, offering clues to how each ranker performed in the prediction of a ranking for each of the used data sets, identified by their ID on OpenML. Furthermore, the evaluation was carried out twice for all measures: once with the full meta data used, and one time using only the meta features not generated by landmarkers, that is the ones that require probing to be carried out for a new data set.

For the computation of the Mann-Whitney U and Kendall rank correlation coefficient, Apache Commons implementations were used [REFERENCE MISSING]. Regarding the Mann-Whitney U, the strategy for ties was set to averaging, and the NaNs were set to be removed before the analysis was started.

Furthermore, four alternatives were selected for a regression-based and preference-based alternative each, to get a general idea of how well the respective approach

[1] Data sets that contained the substring 'BNG' in their name for Bayesian Network Generator, which contain data artificially generated by a Bayesian Network [Rij+14] for the sake of data stream analysis were not included in the evaluation.

Ranker	Configuration
LinearRegression	Linear regression with S:0, R:1.0E-8, num-decimal-places:4
M5P	M5 model trees with M:4
RandomForest	Random forest with P:100, I:100, num-slots:1, K:0, M:1.0, V:0.001, S:1
REPTree	REPT tree with M:2, V:0.001, N:3, S:1, L:1, I: 0.0
InstanceBased	Instance-based label ranking with rankaggregation: Placket-Luce (norm tolerance: 1E-9, max iterations: 1000, log likelihood tolerance: 1E-5), baselearner: kNN (k: 10)
InstanceBased	Instance-based label ranking with rankaggregation: Kemeny-Young, baselearner: kNN (k: 10)
InstanceBased	Instance-based label ranking with rankaggregation: Kemeny-Young, baselearner: kNN (k:$\sqrt{\text{number of instances}}$)
PairwiseComparison	Label ranking by pairwise comparison with baselearner: logistic regression (learning rate: 0.001), gradient step: adam (beta1: 0.99, beta2: 0.999, epsilon: 10E-8)

Tab. 4.1.: An overview over the different base algorithms used in the evaluated rankers.

might be suited for the problem. For the regression-based approach, the algorithms chosen were random forest, REPTree, M5P, and linear regression. For all of them, the WEKA implemenation [Hal+09] was used, and the algorithms where used with the standard hyperparameters set by WEKA. As the preference-based ranking falls under the category of label ranking, and in the jPL-framework, which was used for the implementation of the label ranking algorithms and data set representations, only two alternatives for label ranking are implemented [Int17], the other two alternatives were generated by modifying the hyperparameters. The implemented alternatives are label ranking by pairwise comparison and instance-based label ranking. As in early tests it became apparent that the instance-based approach might hold more potential, for label ranking by pairwise comparison, only the default configuration dictated by jPL was used. For instance-based label ranking, three different configurations were evaluated, the default configurations and two custom ones. For the first customization, the rank aggregation algorithm used by the label ranker was set to Kemeny-Young. For the second customization, additionally, the number of neighbors of the base learner of the label ranker, namely k nearest neighbor (kNN), was set to the square root of the number of instances in the training data set. All of the described rankers are included in an overview in Table 4.1. In the following section, these eight variants are compared with the baseline and oracle mentioned in the previous section, and among each others. They are referenced by the names in Table 4.1 to avoid confusion between the ranker and the learning algorithm it is based on.

4.3 Results

In the following subsections, results of the evaluation are discussed in detail. First, the quality of the predictions of the rankers is examined. Then, the times required for building the rankers and for predictions made as well as for the calculation of the meta features is reviewed. At last, additional insights gathered during the evaluation are pointed out.

4.3.1 Accuracy

Table 4.2 shows the results of the evaluation regarding the accuracy of returned rankings, with the best mean values and significant differences emphasized as described in the caption of the table. When looking at these results, it must first be noted that positive findings regarding the Kendall rank correlation can be reported. For the full set of meta data, six out of the eight implemented alternatives outperform the baseline, all of them significantly, as can be learned from Table 4.3, which shows the results of the significance tests according to the p-value for the Mann-Whitney U. Of all rankers, RandomForest has the highest rank correlation with a value of 0.495. In contrast, the two approaches that did not overtake the baseline, the InstanceBased ranker in the default configuration and the PairwiseComparison ranker, show a significant disadvantage compared to the baseline, and thus also compared to the other ranking approaches. Especially the latter approach has a very low rank correlation, that with a mean value of 0.014 shows almost no correlation with the correct ranking.

Regarding performance loss, meaning much worse the performance of the first recommended algorithm is in comparison to the actual best algorithm on the data set, as defined in Chapter 2, only the LinearRegression and RandomForest ranker show a significant advantage, with all label ranking approaches except for the InstanceBased ranker with Kemeny-Young rank aggregation having a significant disadvantage in contrast. Here, an advantage of the regression-based approach in general is indicated, although the best value with a loss of 3.097 (achieved by the RandomForest ranker) is still fairly high, depending on the data set one considers.

Moving on to B3L, the picture largely remains the same, except that surprisingly, the REPTree ranker now also has a significant advantage over the baseline. The lowest value here is achieved by the inearRegression ranker with a mean B3L of 1.267, in comparison to 2.440 for the best algorithm baseline. These values are illustrated in Figure 4.1, with a scatter plot for individual B3L values on the data sets. In this figure, we can observe that, for a large number of data sets, the B3L is actually 0, namely 206 data sets for the BestAlgorithm ranker and 248 for the RandomForest ranker, which corresponds to 46% and 55% of the 448 data sets used in the evaluation. The main portion of the remaining data points is situated between 0.25 and 8 for the RandomForestRanker, and 0.5 and 16 for the BestAlgorithm ranker, with few exceptions.

As for the rank correlation most of the rankers had a significant advantage over the baseline, it is interesting to compare them among each other. From Table 4.2 it is clear that the regression-based rankers are superior to the preference-based rankers, and this advantage is significant, as the weakest of the regression rankers, the REPTree ranker, is still significantly better than the strongest label ranking approach, the InstanceBased ranker with Kemeny-Young rank aggregation, as indicated by a p-value of 4.305E-6. Out of the four regression rankers, three approaches are furthermore significantly better than the REPTree ranker.

Compared to the evaluation on the full set of meta data, the results on the reduced set without probing are less distinctive. For the Kendall Rank correlation, the LinearRegression, M5P and RandomForest rankers are significantly better than the baseline, and for label ranking, both of the InstanceBased approaches with non-default parameters as well, while the other label ranking approaches are significantly worse than the baseline. In total, the RandomForest ranker is still the best alternative with a correlation of 0.439, slightly worse than the correlation achieved on the full meta data. For both loss measures however, while the disadvantages of the label ranking approaches remain, none of the eight alternatives achieves a significantly better loss or B3L than the baseline anymore.

Again, the different approaches can be compared among each other. With reduced meta data, out of the alternatives that pose a significant advantage compared to the baseline, only the RandomForest ranker is significantly better than all other approaches, the others are not significantly different. With loss and B3L, no such statement can be made.

Further interesting points can be observed. First, in Table 4.2 one can see that for the loss and B3L, often, rankers have a similar values. Second, the performance of the preference-based rankers does not differ very much, whether full meta data or reduced meta data was used in the evaluation. While, for example, Kendall's Rank correlation for the LinearRegression ranker is 0.473 for full meta data, it is noticeably lower for reduced meta data with a value of 0.350. On the other hand, the correlation coefficient for the InstanceBased rankers in all configurations does not change at all when removing meta features that involve probing.

Summarizing the observed results, in general, a regression-based approach shows an indication to be stronger than a preference-based approach. Specifically, using random forest as a regression model is the best solution observed in the evaluation, and label ranking by pairwise comparison the weakest, as it falls behind the other ranking approaches and the baseline often. A surprising result is that the label ranking approaches achieve very similar results with or without probing.

Ranker	Kendall’s Rank Correlation				Loss				B3L			
	Min	Max	Mean	±Stdv	Min	Max	Mean	±Stdv	Min	Max	Mean	±Stdv
LinearRegression	-0.255	0.896	• 0.473	±0.221	0	86.667	• 3.469	±7.236	0	31.220	• **1.267**	±3.008
M5P	-0.290	0.870	• 0.470	±0.219	0	82.353	3.780	±6.528	0	82.353	• 1.508	±4.751
RandomForest	-0.281	0.922	• **0.495**	±0.228	0	60.000	• **3.097**	±5.739	0	34.634	• 1.308	±3.146
REPTree	-0.229	0.896	• 0.411	±0.213	0	82.353	4.829	±7.939	0	34.634	1.759	±3.511
InstanceBased	-0.429	0.870	○ 0.222	±0.249	0	82.353	○ 5.401	±9.429	0	82.353	○ 3.615	±8.531
InstanceBased[2]	-0.429	0.887	• 0.340	±0.252	0	98.367	5.437	±10.311	0	82.353	3.294	±8.357
InstanceBased[3]	-0.429	0.870	• 0.335	±0.248	0	82.353	○ 5.382	±9.391	0	82.353	○ 3.511	±8.483
PairwiseComparison	-0.870	0.576	○ 0.014	±0.234	0	97.822	○ 9.762	±13.120	0	82.353	○ 4.008	±8.591
BestAlgorithm	-0.351	0.758	0.296	±0.213	0	82.353	4.480	±9.205	0	82.353	2.440	±7.625
LinearRegression	-0.532	0.835	• 0.350	±0.235	0	82.353	4.921	±9.109	0	82.353	2.211	±6.744
M5P	-0.281	0.887	• 0.344	±0.219	0	87.045	5.367	±9.662	0	82.353	2.060	±5.466
RandomForest	-0.359	0.913	• **0.439**	±0.233	0	82.353	**3.538**	±6.823	0	40.000	**1.587**	±3.748
REPTree	-0.290	0.844	0.301	±0.215	0	91.154	○ 6.772	±10.710	0	82.353	○ 3.013	±7.162
InstanceBased	-0.429	0.870	○ 0.222	±0.249	0	82.353	○ 5.401	±9.429	0	82.353	○ 3.615	±8.531
InstanceBased[2]	-0.429	0.887	• 0.340	±0.252	0	98.367	5.437	±10.311	0	82.353	3.294	±8.357
InstanceBased[3]	-0.429	0.870	• 0.335	±0.248	0	82.353	○ 5.382	±9.391	0	82.353	○ 3.511	±8.483
PairwiseComparison	-0.870	0.524	○ 0.013	±0.232	0	91.667	○ 9.486	±13.158	0	82.353	○ 4.096	±8.669
BestAlgorithm	-0.351	0.758	0.296	±0.213	0	82.353	4.480	±9.205	0	82.353	2.440	±7.625

Tab. 4.2.: Evaluation results with full meta data (top) and no probing (bottom). The **best mean value**, • significant advantages, and ○ significant disadvantages are marked as such.

[2] Rankaggregation: Kemeny-Young

[3] Rankaggregation: Kemeny-Young, base learner KNN with k=$\sqrt{\text{number of instances}}$

Ranker	P-Value		
	Kendall's Rank Correlation	Loss	B3L
LinearRegression	5.798E-30	0.030	0.001
M5P	1.831E-29	0.888	0.012
RandomForest	1.208E-35	0.006	0.001
REPTree	4.471E-14	0.076	0.607
InstanceBased	6.540E-6	0.012	0.002
InstanceBased[4]	0.012	0.089	0.077
InstanceBased[5]	0.014	0.038	0.012
PairwiseComparison	5.457E-59	1.702E-22	1.451E-9
LinearRegression	2.955E-4	0.118	0.568
M5P	0.007	0.070	0.929
RandomForest	1.278E-18	0.118	0.067
REPTree	0.808	1.781E-7	0.004
InstanceBased	6.540E-6	0.012	0.002
InstanceBased[4]	0.012	0.089	0.077
InstanceBased[5]	0.014	0.038	0.012
PairwiseComparison	1.310E-59	2.016E-22	2.587E-10

Tab. 4.3.: Significance of evaluation results (compared to the baseline) with full meta data (top) and no probing (bottom).

4.3.2 Computation Times

Because the ranking itself is implemented as a means of speeding up the process of algorithm selection, the time it takes to rank is not negligible. The full results of how long it takes to build the respective ranking models and how much time they need for predictions can be taken from Table 4.4. Results are rounded to the full millisecond, as due to the times being measured with a stop watch in the code, more accurate measurements are not possible. The times for building the regression rankers are not surprising; the regression models take far more time to be built than most of the label ranking methods, most likely due to the fact the regression based rankers have to train 22 regression models in order to be trained themselves. However, it is notable that while the PairwiseComparison ranker was not among the best solutions concerning the accuracy of the results, it is the ranker with the highest mean build time for full meta data, and second highest for a reduced meta data set, whereas in other cases the high quality of results is correlated with a high time invested in the building of the ranker. The RandomForest ranker, for example, is the ranker with the highest build time (apart from the PairwiseComparison ranker), and also with the rankings of the generally highest quality. But the most important fact to be taken away from Table 4.4 is that the prediction times for all rankers are short

[4] Rankaggregation: Kemeny-Young

[5] Rankaggregation: Kemeny-Young, base learner KNN with k=$\sqrt{\text{number of instances}}$

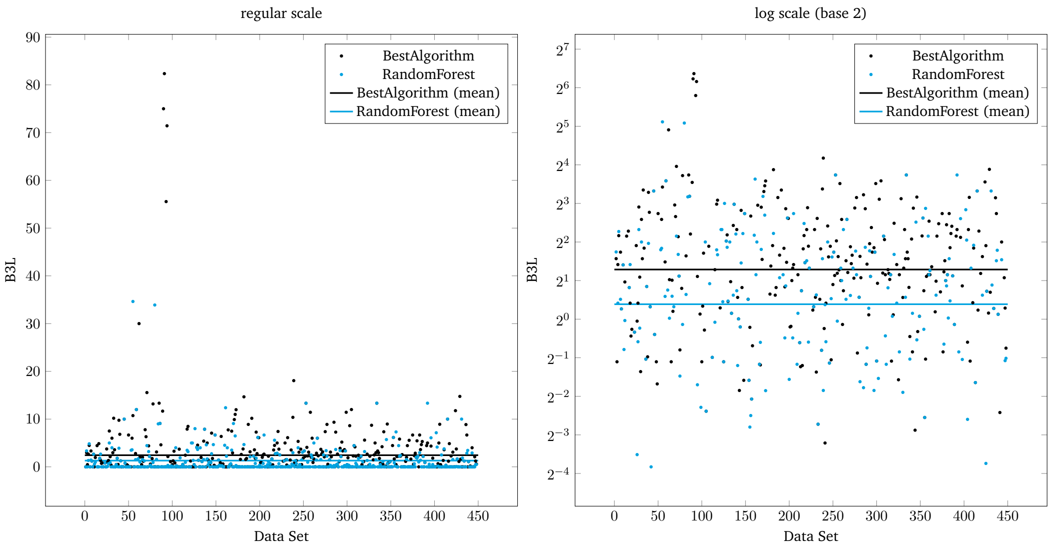

Fig. 4.1.: The B3L for the baseline compared to the random forest ranker (full meta data).

enough to be negligible, with the mean prediction time never being higher than five milliseconds and the maximum never exceeding 50 milliseconds. Furthermore, while the build times are higher, these are not as important due to the fact that each model only needs to be trained once. Although in a real-world scenario, calculation times for meta features also have to be taken into account when ranking, which therefore will be discussed in the following paragraph.

Ranker	Ranker Build Time (ms)				Ranker Prediction Time (ms)			
	Min	Max	Mean	Stdv	Min	Max	Mean	Stdv
LinearRegression	1515	1792	1621	29	0	1	0	0
M5P	3250	5016	3329	94	0	16	0	1
RandomForest	6023	12676	6917	1222	0	16	3	2
REPTree	596	1073	626	25	0	16	0	1
InstanceBased	65	416	82	18	0	47	5	8
InstanceBased[6]	65	112	84	9	0	19	1	3
InstanceBased[7]	65	127	82	8	0	16	1	2
PairwiseComparison	8456	15063	9096	449	0	16	0	2
BestAlgorithm	204	1250	227	52	0	0	0	0
LinearRegression	421	500	446	10	0	0	0	0
M5P	2219	4225	2282	148	0	16	0	1
RandomForest	6175	10535	6419	479	0	16	3	6
REPTree	484	1078	509	32	0	15	0	1
InstanceBased	62	141	77	7	0	16	4	7
InstanceBased[6]	62	125	79	8	0	16	1	3
InstanceBased[7]	62	110	77	7	0	16	1	3
PairwiseComparison	6705	7893	7253	169	0	16	0	2
BestAlgorithm	206	405	232	23	0	0	0	0

Tab. 4.4.: Build and prediction times of the rankers rounded to full milliseconds with full meta features (top) and no probing (bottom).

[6] Rankaggregation: Kemeny-Young

[7] Rankaggregation: Kemeny-Young, base learner KNN with k=$\sqrt{\text{number of instances}}$

Table 4.5 shows the times required for the computation of groups of meta features as discussed in Chapter 3. As can be taken from the table, the computation of the full meta features takes approximately 135 milliseconds on average, while the simple meta features without probing add approximately 3 milliseconds to the prediction times on average, which is already a difference of factor 45, which is quite large especially considering the fact that only light data sets, restricted in the number of features and instances, were used. To emphasize this difference, the meta features were calculated for the popular MNIST data set [LCBnd], which contains pictures of handwritten digits in a 28 by 28 pixel resolution, resulting in 784 features, plus the target feature. The data set contains 70.000 different pictures. For the computation of the full set of meta features, 4 hours and 51 minutes were needed, out of which only 24 seconds were needed for the reduced set of meta features, a difference of factor 735. Regarding these computation times it is also notable that among the probing meta feature groups, some are considerably more expensive than others, so that in the future it might be sensible to consider only light probing with probing meta feature groups that are not as expensive.

Meta Feature Group	Computation Time (ms)			
	Min	Max	Mean	Stdv
NominalAttDistinctValues	0	16	0.127	1.128
SimpleMetaFeatures	0	16	0.221	1.614
Cardinality	0	19	0.654	2.723
Statistical	0	69	1.520	5.443
DecisionStump, 2 folds	0	54	1.536	4.710
RandomTreeDepth1, 2 folds	0	29	1.955	4.785
RandomTreeDepth2, 2 folds	0	18	2.114	4.793
RandomTreeDepth3, 2 folds	0	30	1.636	4.454
REPTreeDepth1, 2 folds	0	56	2.980	6.805
REPTreeDepth2, 2 folds	0	67	2.955	6.999
REPTreeDepth3, 2 folds	0	60	3.259	7.206
J48.001., 2 folds	0	201	5.129	13.935
J48.0001., 2 folds	0	101	4.944	10.657
J48.00001., 2 folds	0	116	4.464	10.744
NaiveBayes, 2 folds	0	200	6.853	17.679
kNN1N, 2 folds	0	1118	22.250	69.212
CfsSubsetEval kNN1N, 2 folds	2	1096	25.806	52.236
CfsSubsetEval NaiveBayes, 2 folds	2	132	23.304	12.918
CfsSubsetEval DecisionStump, 2 folds	2	150	23.087	12.296

Tab. 4.5.: Computation times for groups of meta features.

4.3.3 Further Insights

Further insights to be noted are the accuracies predicted for classifiers by the regression models, and the static ranking constructed by the best algorithm baseline. The accuracy of the predictions achieved by the regression-based rankers are depicted in Table 4.6, for full and reduced meta data respectively. The first thing that attracts attention in this table is the absurdly high values for the LinearRegression and M5P rankers, however, these are all caused by the very high maximum value on one data set with id 685 on OpenML. If these extreme maxima remain out of consideration for the LinearRegression and M5P ranker (the other algorithms do not have this problem) more reasonable values are obtained. For the M5P ranker, on the full meta data the new maximum is 311.526, mean 8.127 and standard deviation 18.407; on the reduced meta data it is a maximum of 474.692, mean 15.444 and standard deviation 27.433. For the LinearRegression ranker, on the full meta data the new maximum is 142.996, mean 9.281 and standard deviation 8.438; on the reduced meta data it is a maximum of 131.456, mean 14.351 and standard deviation 10.299. However, these values are still worse than the ones obtained for the RandomForest and REPTree variants. It comes as little surprise that the RandomForest ranker also delivers the most accurate results for performance estimates, but as these best values are more than 6% off on average for the full meta data and even more than 10% off on average for the reduced meta data, it is questionable whether these prediction can be useful. This is especially the case, since in addition to a relatively high mean, the standard deviation is rather high as well in all observed variants. The usefulness of the tool to predict performance values is therefore dependent on the task these predictions would be utilized for and whether it requires very accurate predictions of performance values. On the other hand, judging by the fact that the regression-based approach shows to be superior to the preference-based approach, these imperfect predictions seem still seem to be accurate enough to result in a rather good ranking.

Ranker	Root Mean Square Error			
	Min	Max	Mean	Stdv
LinearRegression	2.269	454695.445	1028.757	21505.865
M5P	1.487	50757.174	121.914	2400.412
RandomForest	0.938	23.788	6.625	3.873
REPTree	2.276	23.676	7.616	4.227
LinearRegression	3.503	372808.391	850.213	17632.512
M5P	1.934	561871.676	1275.212	26574.835
RandomForest	0.868	67.892	10.692	7.429
REPTree	1.573	78.048	13.492	8.815

Tab. 4.6.: Root mean square error of predicted performance values for the regression based rankers with full meta features (top) and no probing (bottom).

The second interesting thing to consider is the ranking which is created by the best algorithm baseline. This ranking constitutes meta-knowledge about the learning process itself in that it shows which algorithms are generally a good choice for many data sets. This ranking is depicted in Table 4.7. The table is to be read top to bottom; of the 22 algorithms, the algorithm in the i-th row is placed first on most data sets when compared to the next 22-i algorithms. The associated number represents the number of data sets on which the algorithm was placed first. It is, however, not a particularly surprising result, as for example the placement of random forest as the first algorithm, and ZeroR in the lower ranks was to be expected. Random forest is a learning algorithm that is known to often deliver good performances, which was proven again in the results discussed previously - out of the reviewed ranking approaches, the RandomForest ranker was the best choice. On the other hand, ZeroR is a simple classifier often used as a baseline, that predicts a static value based on the majority class (or average numeric value of the target feature) [Dev+11].

Classifier	Number of Data Sets Placed First
RandomForest	80
MultilayerPerceptron	66
BayesNet	51
Naive Bayes	53
Logistic	50
SGD	65
SMO	93
SimpleLogistic	131
LMT	120
IBk	66
KStar	95
DecisionTable	88
JRip	117
PART	104
J48	140
REPTree	110
RandomTree	133
OneR	177
DecisionStump	253
VotedPerceptron	255
ZeroR	378
NaiveBayesMultinomial	448

Tab. 4.7.: The ranking of classifiers generated by the baseline.

5 Related Work

The demand for aid in the process of selecting an algorithm has already led to the development of a number of solutions that automate machine-learning (AutoML). In the following sections, the functionality of a few such tools that are related to this work are outlined briefly, organized by their scope of operation. Each tool's usage of meta knowledge is discussed shortly.

5.1 Predicting Rankings

In contrast to ranking solely based on classifier performances, Abdulrahman et al. investigated an approach to extend existing ranking methods by incorporating the time needed for the evaluation of the classifiers [Abd+18]. They call this combined measure of accuracy and time A3R, and integrate it into two different ranking approaches. In both cases, the authors find that this leads to a better mean interval loss, which is a measure that expresses the mean loss experienced with respect to the time that has been allowed for the construction of the ranking, that is the time taken to evaluate top-ranked classifiers.

One of those two approaches is called average-ranking. It utilizes meta knowledge to suggest a ranking of classifiers for a new data set, which in this case consists of recorded past performances of classifiers on a number of data sets, meaning this algorithm aggregates the performances of classifiers across all data sets once and then always recommends the same ranking. The new measure is integrated here by, instead of ordering only according to performance values, ordering classifiers according to the combined measure A3R.

The second approach does not initially use meta knowledge and is called active testing. Active testing is a strategy that 'intelligently selects the most useful tests' [Abd+18] to perform on a given data sets. Tests in this context mean the evaluation of a classifier that is considered for the ranking on the given data set. The authors improved this strategy by reducing the time needed to identify a good algorithm by, similarly to the rank aggregation variant, incorporating time in the estimations made by the strategy which previously only compared accuracy values.

5.2 Algorithm and Hyperparameter Selection

Taking it a step further than predicting rankings of classification algorithms with fixed hyperparameters is additionally optimizing the classifier's hyperparameters, which has been defined as 'the combined algorithm selection and hyperparameter optimization problem (short: CASH)' [Tho+13]. Two widely used approaches of this kind are Auto-WEKA and AUTO-SKLEARN. It has to be noted that these approaches also go further than the solution proposed in this thesis, which currently only takes into account one fixed hyperparameter configuration for each classification algorithm.

Auto-WEKA is an AutoML tool that both selects a machine learning algorithm and optimizes its hyperparameters by using Bayesian optimization [Tho+13]. It was first released in 2013 as an extension to the popular data mining software WEKA [Hal+09], which also offers a user-friendly GUI in addition to a command-line interface and an API, to assist the large number of novice users of the software in selecting parametrized algorithms for their problems. The tool has since grown in popularity and is in version 2.0 as of March 2016 [Kot+16]. In Auto-WEKA, the problem of selecting an algorithm and its hyperparameters is combined by treating the algorithm itself as a hyperparameter and searching the joint space of algorithms and hyperparameters for the best solution. An input data set is first preprocessed by means of feature selection. Then, Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) is used to 'iterate [...] between fitting models and using them to make choices about which configurations to investigate' [HHL11]. In the case of Auto-WEKA, this means that during the optimization process, a model is built, a configuration of hyperparameters that is promising regarding the current model and training data is tried out, and the result is fed back to the model. This cycle is then repeated until the allocated time has run out. Auto-WEKA exploits hard-coded meta-knowledge, that is considering past performances of algorithms, to make decisions by always trying algorithms like Random Forest, which perform well on a large number of data sets, first.

AUTO-SKLEARN has been described as a sister-package to Auto-WEKA and is an AutoML tool which is based on scikit-learn, a machine learning library for Python [Feu+15]. It works very similar to AutoML but extends it by adding a meta-learning pre-processing step to warmstart the Bayesian optimization and automatically constructing ensembles during optimization. During the pre-processing phase, performance values for the classifiers available in AUTO-SKLEARN are recorded on a set of data sets. For each data set, the algorithm which shows the best empirical performance is noted. Then, certain meta-features are calculated for each data set.

The first step of the tool when given a new problem is to calculate meta-features of the data set. Then, the Manhattan distance to the other data sets is determined according to the meta-features, and the algorithms that are associated with the k-nearest data sets are used as a starting point for further optimization. The authors observe that the additional meta-learning and ensemble construction result in a more efficient and robust application. Their results show that meta-learning can be used to improve the overall AutoML process.

5.3 Constructing pipelines

Before a classifier is evaluated on a data set, often a number of pre-processing steps are executed first. This includes, for example, selecting promising features and discarding others, and normalizing the data. The 'sequence of tasks that need to be performed to classify instances belonging to a given dataset into a set of predefined categories' [Sá+17] can therefore be defined as a classification pipeline. Two tools which construct such complete pipelines for data sets are ML-Plan and the RECIPE framework.

ML-Plan is an AutoML tool that instead of concentrating on hyperparameter optimization, aims to optimize the whole machine-learning pipeline [WMH]. This is achieved by viewing machine-learning as a task, building a hierarchical task network out of the resulting subtasks, and then searching for a solution in the induced tree structure. In the tree, the root node contains the complex task of building a machine learning pipeline, inner nodes represent incomplete pipelines consisting of complex and possibly also primitive tasks, and leaf nodes are complete pipelines that include only primitive tasks. An example of this might be 'classify' as the root node, with an intermediate node on some level that contains the tasks 'build NN', 'train NN' and 'predict from NN'. The complex task 'build NN' would then further be decomposed, and could lead to a leaf node with n tasks 'Add layer', 'train NN' and 'predict from NN', which are all primitive tasks that do not need to be further decomposed. A best-first search algorithm in a modified variant is then used to find good solutions in this task network. For the actual implementation of the learning algorithms, WEKA is used. Like Auto-WEKA, ML-Plan uses hard-coded meta-knowledge by using a fixed orders as to which classifiers are tried first. However, the authors find that their results exceed those achieved by Auto-WEKA.

Similar to ML-PLAN, the RECIPE framework constructs whole classification pipelines for new problems [Sá+17]. However, this is done by means of grammar-based

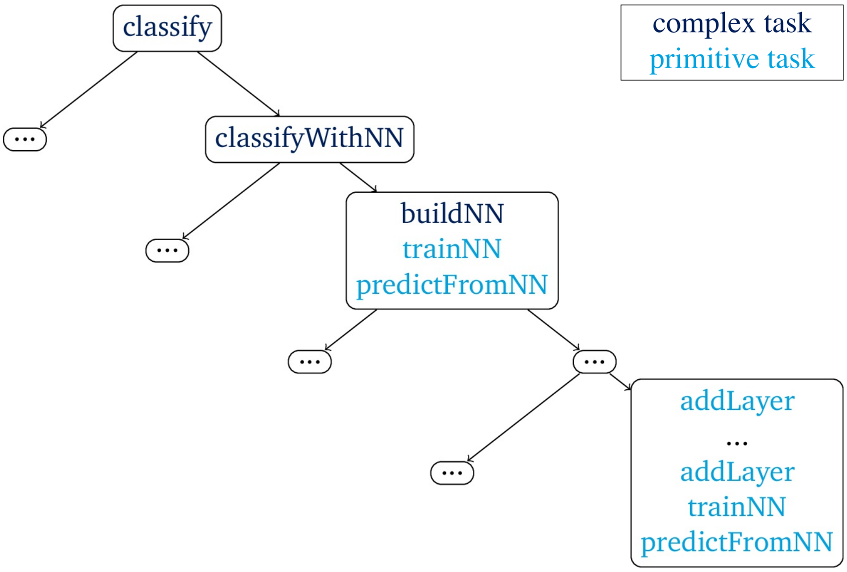

Fig. 5.1.: An example for how the complex task 'classify' might be broken down by ML-Plan, adapted from [WMH]. Nodes containing '...' represent an undefined number of subtrees.

genetic programming: the tasks that the pipeline is composed of are represented by a grammar, which 'is used to generate the initial population, as well as to constrain the crossover and mutation operations, which always need to be valid according to the grammar' [Sá+17]. This means that generated individuals representing complete pipelines can never be invalid. So for a new data set combined with a grammar that represents the valid pipelines, the tool first initializes the first generation according to the grammar. The fitness of individuals is then evaluated by mapping them to their respective implementations in the scikit-learn framework, and a new generation is created that takes this newly gained information into account. Evaluation by the authors indicates that the RECIPE framework is able to compete with AUTO-SKLEARN and a different evolutionary approach, although it does not incorporate meta-knowledge in the search. It is notable that another evolutionary approach, Tree-based Pipeline Optimization Tool (TPOT) [Ols+16] exists, although in comparison to the RECIPE framework, it can generate individuals that represent invalid pipelines.

6 Conclusion

In this thesis, the problem of predicting a ranking of classification algorithms for a new data set on the basis of meta features of the data set and past performances of the algorithms has been considered. Being able to predict a ranking of such algorithms is desirable since this potentially speeds up and simplifies the process of algorithm selection, which is relevant due to a rapidly increasing amount of available data, and more importantly, data that is available but has not yet been analyzed. This problem has been addressed by implementing two different approaches, regression-based and preference-based ranking. Both have been evaluated extensively against a baseline and an oracle.

The evaluation showed a significant advantage regarding Kendall's rank correlation coefficient for all tested methods in comparison to the best algorithm baseline, with the exception of the InstanceBased ranker in the standard configuration and the PairwiseComparison ranker for the full meta data. However, regarding the practically possibly more important measure of B3L, only the three regression-based variants with linear regression, M5P and random forest as the regression learners can hold a significant advantage. When evaluating B3L on reduced meta data that does not incorporate probing, although the evaluated data indicates a slight advantage of those same regression models over the baseline still, it is not statistically significant anymore. Furthermore, some label ranking methods even show a significant disadvantage in comparison to the baseline for the B3L and loss for both evaluation on the full and reduced meta data. Thus it can be concluded that in general, a regression-based approach seems to be more promising than a preference-based approach, but is not better than the baseline regarding the B3L when evaluated on the selected meta data without probing. Furthermore, since the regression-based approach shows an advantage over the preference-based approach in respect to the problem considered in this thesis, one can conclude, that here, the preference-based approach generalizes too much. The additional information available to the regression-based models, the performance values of classifiers, seems to pose an advantage over the abstraction of only considering rankings of classifiers in this case. Out of the regression-based variants, the RandomForest ranker seems to be the most promising one.

The main limitation of these findings is that the data sets used in the evaluation process were not very large, being limited to a maximum of 100 features and 1000 instances. Often, data sets can get much larger and the question whether this ranking approach would then still pose an efficient solution regarding the calculation of meta features and the resulting prediction times and, maybe even more importantly, an accurate enough solution for predicting classifiers, remains.

On the basis of these results, it can be concluded that there is an indication of a connection between certain meta features of a data set and the predictive accuracy of classification algorithms for this data set, which can be exploited to a degree by regression models and label ranking models to predict a ranking of classification algorithms. A practical application of these findings, apart from the direct use as a ranker, may be to incorporate the implementation or parts of it in another AutoML tool as a search heuristic, similar to how some AutoML solutions like AUTO-SKLEARN already benefits from meta learning [Feu+15]. However, some additional work may have to be done in extension to this thesis for such an integration to be sensible.

6.1 Future Work

As the results have shown that the approach taken in this thesis has worked out reasonably well, for example when considering the regression-based approach using random forest, many possibilities for additional work arise. An intuitive thought would be to extend the evaluation of the tool, for instance in terms of meta features. Additional sets of meta features could be evaluated, like adding implementations for additional meta features, or considering using only light probing by incorporating the performance values for usually cheap algorithms in the meta features. Since the significant advantage of any approach was lost regarding B3L when moving from the full meta data to the reduced meta data without probing, this might be especially interesting. One could consider adding the meta feature groups for DecisionStump, RandomTreeDepth2, REPTreeDepth2, and J48.0001, which would add approximately 12 milliseconds to the already existing 3 milliseconds of calculation time for the meta features in the mean, which seems reasonable, and even on the large MNIST data set only represented 8.5 minutes out of a computation time of almost five hours for the full set of meta features. An effect on the time required for and quality of predictions for other larger data sets would have to be examined still.

Furthermore, more insights could be gained about whether this approach is competitive by comparing it with other ranking approaches, for example with the one proposed by Abdulrahman et al. [Abd+18]. In this regard it would also be interesting to implement the loss and time-loss curves proposed by the authors, which show how loss decreases as one advances in the predicted ranking and tries out the recommended algorithms, and at what cost in the sense of time the sampling of algorithms comes respectively.

Additionally, so far the evaluation of the tool has been confined to predicting algorithms for classification. Since this has been relatively successful, it could be tested whether similar predictions can also be made for other machine learning tasks like regression or clustering with the same approach by extending the tool to handling these functions. Also, one could try to include a few parameterized versions of algorithms, as so far only one variant of each classifier with standard hyperparameters has been used.

In a similar manner to extending the algorithms for which a performance measure is predicted, one could use the tool to predict different performance measures. For this, the tool does not even need to be adapted, as it is agnostic to the semantics behind the measure it predicts, it only needs to be numeric. One possibility for this would be the prediction of build or prediction times of learning algorithms, or a measure that combines time and accuracy like A3R [Abd+18].

Depending on the fact whether one is willing to trade off longer build and prediction times for improved accuracy, it would also be possible to fit more sophisticated regression models for the regression based approach. This could include feature pre-processing, automating the choice of regression algorithm with one of the available tools like AUTOWEKA [Tho+13] or AUTO-SKLEARN [Feu+15], or even using a whole pipeline prediction tool, for example ML-Plan [WMH] or the RECIPE framework [Sá+17].

Last, it could be interesting to implement an additional baseline that averages ranks instead of counting how many times an algorithm is ranked first. This would help to determine if the ranking returned by the best algorithm baseline (see Fig. 4.7) still holds when using a different baseline. Naturally, this new baseline then could also be used to re-evaluate the implemented ranking concepts.

A Appendix

Meta Feature Group
NominalAttDistinctValues
MaxNominalAttDistinctValues, MinNominalAttDistinctValues, MeanNominalAttDistinctValues, StdvNominalAttDistinctValues
SimpleMetaFeatures
NumberOfInstances, NumberOfFeatures, NumberOfClasses, Dimensionality, NumberOfInstancesWithMissingValues, NumberOfMissingValues, PercentageOfInstancesWithMissingValues, PercentageOfMissingValues, NumberOfNumericFeatures, NumberOfSymbolicFeatures, NumberOfBinaryFeatures, PercentageOfNumericFeatures, PercentageOfSymbolicFeatures, PercentageOfBinaryFeatures, MajorityClassSize, MinorityClassSize, MajorityClassPercentage, MinorityClassPercentage, AutoCorrelation
Cardinality
MeanCardinalityOfNumericAttributes, StdevCardinalityOfNumericAttributes, MinCardinalityOfNumericAttributes, MaxCardinalityOfNumericAttributes, MeanCardinalityOfNominalAttributes, StdevCardinalityOfNominalAttributes, MinCardinalityOfNominalAttributes, MaxCardinalityOfNominalAttributes, CardinalityAtTwo, CardinalityAtThree, CardinalityAtFour
Statistical
MeanMeansOfNumericAtts, MeanStdDevOfNumericAtts, MeanKurtosisOfNumericAtts, MeanSkewnessOfNumericAtts, MinMeansOfNumericAtts, MinStdDevOfNumericAtts, MinKurtosisOfNumericAtts, MinSkewnessOfNumericAtts, MaxMeansOfNumericAtts, MaxStdDevOfNumericAtts, MaxKurtosisOfNumericAtts, MaxSkewnessOfNumericAtts, Quartile1MeansOfNumericAtts, Quartile1StdDevOfNumericAtts, Quartile1KurtosisOfNumericAtts, Quartile1SkewnessOfNumericAtts, Quartile2MeansOfNumericAtts, Quartile2StdDevOfNumericAtts, Quartile2KurtosisOfNumericAtts, Quartile2SkewnessOfNumericAtts, Quartile3MeansOfNumericAtts, Quartile3StdDevOfNumericAtts, Quartile3KurtosisOfNumericAtts, Quartile3SkewnessOfNumericAtts
Landmarkers[1]
DecisionStump, RandomTreeDepth1, RandomTreeDepth2, RandomTreeDepth3, REPTreeDepth1, REPTreeDepth2, REPTreeDepth3, J48.001., J48.0001., J48.00001., NaiveBayes, kNN1N, CfsSubsetEval kNN1N, CfsSubsetEval NaiveBayes, CfsSubsetEval DecisionStump

Tab. A.1.: The meta features computed by each meta feature group. More details can be found on www.openml.org.

[1] Each landmarker represents its own group of meta features consisting of AUC, ErrRate and Kappa. All are computed with 2-fold crossvalidation.

Bibliography

[Abd+18] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N. van Rijn, and Joaquin Vanschoren. „Speeding up algorithm selection using average ranking and active testing by introducing runtime“. In: *Machine Learning* 107.1 (2018), pp. 79–108 (cit. on pp. 35, 40, 41).

[ALM12] YS Abu-Mostafa, HT Lin, and M Magdon-Ismail. „Learning from Data: A Short Course: AMLbook“. In: *View Article PubMed/NCBI Google Scholar* (2012) (cit. on pp. 5–7).

[Bra+08] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008 (cit. on p. 9).

[Dev+11] C Lakshmi Devasena, T Sumathi, VV Gomathi, and M Hemalatha. „Effectiveness evaluation of rule based classifiers for the classification of iris data set“. In: *Bonfring International Journal of Man Machine Interface* 1 (2011), p. 5 (cit. on p. 34).

[Egg+15] Katharina Eggensperger, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. „Efficient Benchmarking of Hyperparameter Optimizers via Surrogates“. In: *AAAI*. AAAI Press, 2015, pp. 1114–1120 (cit. on p. 2).

[Feu+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, et al. „Efficient and robust automated machine learning“. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2962–2970 (cit. on pp. 36, 40, 41).

[FH10] Johannes Fürnkranz and Eyke Hüllermeier, eds. *Preference Learning*. Springer, 2010 (cit. on pp. 7, 8).

[GR12] John Gantz and David Reinsel. *THE DITIGAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. Tech. rep. IDC 1414_v3. International Data Corporation, Nov. 2012 (cit. on p. 1).

[GR17] John Gantz and John Rydning. *Data Age 2025: The Evolution of Data to Life-Critical. Don't Focus on Big Data; Focus on the Data That's Big*. Tech. rep. International Data Corporation, Apr. 2017 (cit. on p. 1).

[Hal+09] Mark Hall, Eibe Frank, Geoffrey Holmes, et al. „The WEKA data mining software: an update“. In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18 (cit. on pp. 19, 26, 36).

[Har99] Mark Harris. *Splice-2 comparative evaluation: Electricity pricing*. Tech. rep. The University of South Wales, 1999 (cit. on p. 1).

[HHL11] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. „Sequential Model-Based Optimization for General Algorithm Configuration.“ In: *LION* 5 (2011), pp. 507–523 (cit. on p. 36).

[Ken38] Maurice G Kendall. „A new measure of rank correlation“. In: *Biometrika* 30.1/2 (1938), pp. 81–93 (cit. on p. 11).

[Koh+95] Ron Kohavi et al. „A study of cross-validation and bootstrap for accuracy estimation and model selection“. In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145 (cit. on p. 10).

[Kot+16] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. „Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA“. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–5 (cit. on p. 36).

[LBV12] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. „Selecting Classification Algorithms with Active Testing“. In: *MLDM*. Vol. 7376. Lecture Notes in Computer Science. Springer, 2012, pp. 117–131 (cit. on p. 12).

[MW47] Henry B Mann and Donald R Whitney. „On a test of whether one of two random variables is stochastically larger than the other“. In: *The annals of mathematical statistics* (1947), pp. 50–60 (cit. on p. 14).

[Ols+16] Randal S Olson, Ryan J Urbanowicz, Peter C Andrews, Nicole A Lavender, Jason H Moore, et al. „Automating biomedical data science through tree-based pipeline optimization“. In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2016, pp. 123–137 (cit. on p. 38).

[Ric76] John R Rice. „The algorithm selection problem“. In: *Advances in computers*. Vol. 15. Elsevier, 1976, pp. 65–118 (cit. on p. 9).

[Rij+14] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. „Algorithm selection on data streams“. In: *International Conference on Discovery Science*. Springer. 2014, pp. 325–336 (cit. on p. 25).

[Sá+17] Alex Guimarães Cardoso de Sá, Walter José G. S. Pinto, Luiz Otávio Vilas Boas Oliveira, and Gisele L. Pappa. „RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines“. In: *EuroGP*. Vol. 10196. Lecture Notes in Computer Science. 2017, pp. 246–261 (cit. on pp. 37, 38, 41).

[Sie87] J Paul Siebert. *Vehicle recognition using rule based methods*. Tech. rep. TIRM87018. Turing Institute, 1987 (cit. on p. 1).

[Tho+13] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. „Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms“. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 847–855 (cit. on pp. 36, 41).

[Van+13] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. „OpenML: Networked Science in Machine Learning“. In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60 (cit. on p. 20).

[WMH] Marcel Wever, Felix Mohr, and Eyke Hüllermeier. „Automatic Machine Learning: Hierarchical Planning Versus Evolutionary Optimization“. Unpublished: Proc. 27. Workshop Computational Intelligence, Dortmund, 23. -24.11.2017 (cit. on pp. 37, 38, 41).

Webpages

[36] *iris*. 1936. URL: `https://www.openml.org/d/61` (visited on Feb. 11, 2018) (cit. on p. 6).

[Cac17a] Miguel Cachada. *Run 2210139*. May 17, 2017. URL: `https://www.openml.org/r/2210139` (visited on Nov. 30, 2017) (cit. on p. 1).

[Cac17b] Miguel Cachada. *Run 2221382*. May 18, 2017. URL: `https://www.openml.org/r/2221382` (visited on Nov. 30, 2017) (cit. on p. 1).

[Cac17c] Miguel Cachada. *Run 2290623*. May 20, 2017. URL: `https://www.openml.org/r/2290623` (visited on Nov. 30, 2017) (cit. on p. 1).

[Cac17d] Miguel Cachada. *Run 2290766*. May 20, 2017. URL: `https://www.openml.org/r/2290766` (visited on Nov. 30, 2017) (cit. on p. 1).

[Int17] IntelligentSystemsGroup. *jPL framework*. June 30, 2017. URL: `https://github.com/Intelligent-Systems-Group/jpl-framework` (visited on Nov. 30, 2017) (cit. on pp. 19, 26).

[Int18] IntelligentSystemsGroup. *Evaluation Engine*. Jan. 17, 2018. URL: `https://github.com/openml/EvaluationEngine` (visited on Feb. 11, 2018) (cit. on p. 20).

[LCBnd] Yann LeCun, Corinna Cortes, and Christopher J.C. Burgers. *THE MNIST DATABASE of handwritten digits*. n.d. URL: `http://yann.lecun.com/exdb/mnist/` (visited on Mar. 13, 2018) (cit. on p. 32).

[nd] *OpenML Bootcamp*. n.d. URL: `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm` (visited on Feb. 11, 2018) (cit. on p. 24).

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

I hereby declare that I prepared this thesis independently and without illicit assistance and have not used any sources without declaration. Any statements that have been adopted literally or analogously have been identified as such. This thesis has not been submitted in the same or substantially similar version, not even in part, to another authrity for grading and has not been published elsewhere.

Declaration (German)

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Paderborn, March 14, 2018

Helena Graf