Meta-Learning and the Full Model Selection Problem

A thesis

submitted in fulfilment

of the requirements for the degree

of

Doctor of Philosophy in Computer Science

at the

University of Waikato

by

Quan Sun

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2014

Abstract

When working as a data analyst, one of my daily tasks is to select appropriate tools from a set of existing data analysis techniques in my toolbox, including data preprocessing, outlier detection, feature selection, learning algorithm and evaluation techniques, for a given data project. This indeed was an enjoyable job at the beginning, because to me finding patterns and valuable information from data is always fun. Things become tricky when several projects needed to be done in a relatively short time.

Naturally, as a computer science graduate, I started to ask myself, "What can be automated here?"; because, intuitively, part of my work is more or less a loop that can be programmed. Literally, the loop is "choose, run, test and choose again... until some criterion/goals are met".

In other words, I use my experience or knowledge about machine learning and data mining to guide and speed up the process of selecting and applying techniques in order to build a relatively good predictive model for a given dataset for some purpose. So the following questions arise:

"Is it possible to design and implement a system that helps a data analyst to choose from a set of data mining tools? Or at least that provides a useful recommendation about tools that potentially save some time for a human analyst."

To answer these questions, I decided to undertake a long-term study on this topic, to think, define, research, and simulate this problem before coding my dream system. This thesis presents research results, including new methods, algorithms, and theoretical and empirical analysis from two directions, both of which try to propose systematic and efficient solutions to the questions above, using different resource requirements, namely, the meta-learning-based algorithm/parameter ranking approach and the meta-heuristic search-based full-model selection approach.

Some of the results have been published in research papers; thus, this thesis also serves as a coherent collection of results in a single volume.

Acknowledgements

This thesis would not exist without the monumental support, insightful supervision and consistent encouragement of my chief supervisor, Associate Professor Bernhard Pfahringer. I sincerely thank him for leading me into the interesting field of meta-learning. Words cannot express my gratitude. Thank you, Bernhard!

I am grateful to my co-supervisor, Dr Michael Mayo, for his invaluable knowledge and for his support in a number of ways. Bernhard and Mike have been a pleasure to work with, a supervision team from whom I really could not have asked more.

I would like to thank the members of the Machine Learning group and the Technical Support group, especially Associate Professor Eibe Frank, Dr Albert Bifet, Dr Sam Sarjant, Dr Mark Hall, Dr Remco Bouckaert, Dr Tony Smith, Jacqui Elphick, Clint Dilks, Mike Vallabh, Peter Reutemann, Dale Fletcher, Antti Puurula, Professor Geoff Holmes, Professor Ian Witten, and the countless students who have come and gone over the years. It is they who make the Machine Learning lab a fun environment to work in. I would also like to thank Dr Lyn Hunt and Dr Bob Durrant of the Statistics department for their interesting discussions about statistics.

During the period of my study, the University of Waikato has provided me with much-appreciated financial support through a Waikato doctoral scholarship, a PhD study award and has provided funding for several conference trips. I would like to thank all the people in the Faculty of Computing and Mathematical Sciences.

I also thank my support system outside university. Thanks go to my colleagues and friends, Dr Richard Kirkby, David Selby, Tom Fuyala, Bruce Bowering, Nick Waterhouse, Duncan Mackintosh and John Moriarty. I would like to thank Associate Professor Carlos Soares and Professor John I. Marden for their insightful suggestions and comments on my research.

Finally, I would like to thank my family for endless support and love. This thesis is dedicated to you.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Through decades of development, machine learning has came out from its incunabulum period, and is becoming more sound and well-defined. A rich set of data analysis techniques, including algorithms and methods, have been and are being developed on demand. Nowadays users of machine learning are facing a new problem: **how to choose an appropriate set of tools for a given data mining problem in order to achieve better data mining than simply using an arbitrary selection of tools**. This thesis presents research results from two approaches, both of which are systematic and efficient solutions to the above question, namely the meta-learning-based algorithm/parameter recommendation approach (the meta-learning approach) and the meta-heuristic search-based Full Model Selection approach (the FMS approach).

1.1 Applications

The main difference between the two approaches is due to the time required for making a useful recommendation. A concept graph is given in Figure 1.1. The meta-learning approach has the ability to make recommendations instantly, e.g., within a second, but its recommendation might not be optimal for data

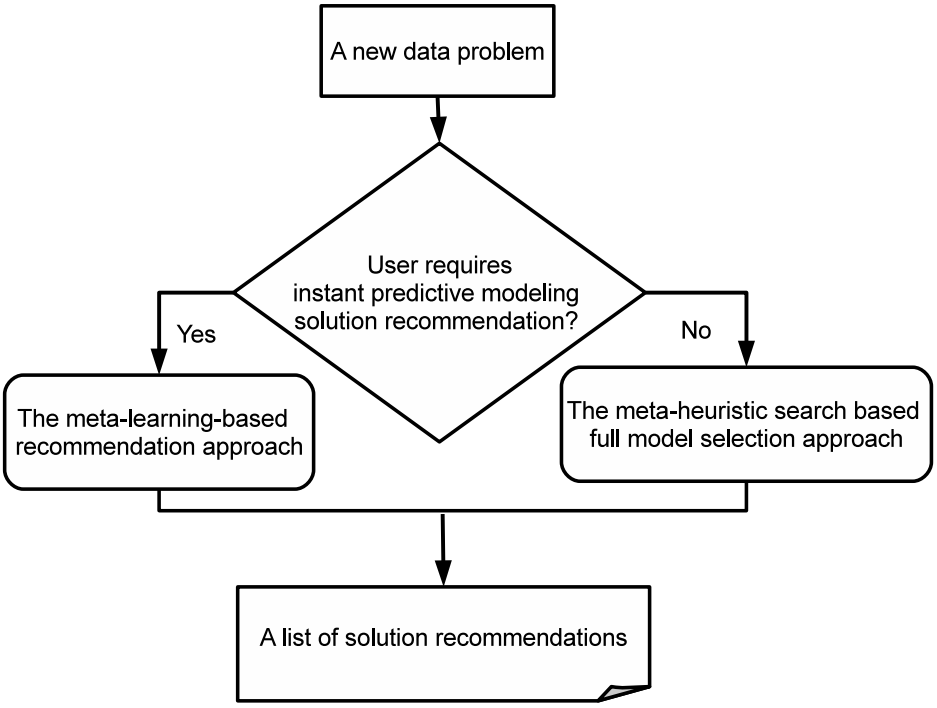


Figure 1.1: The meta-learning approach and the FMS approach

characteristics that it has not seen before. By contrast, the FMS approach does not suffer the "unknown" data characteristics issue since the candidate models and solutions are evaluated directly on a target dataset. However, it can be much slower than meta-learning in terms of making a recommendation, because FMS is based on a procedure guided by some heuristics looking for a suitable solution in a large search space. Therefore, when fast recommendation is essential, the meta-learning approach is recommended; whereas when meta-learning's recommendation is not good enough, the FMS is a reasonable alternative.

Potentially, the new techniques, algorithms and methods proposed in this thesis can be used as building blocks in a data mining recommendation engine for a given data problem.

1.2 Contributions

This research led to general contributions to the field of data mining and machine learning. These include the following:

- A new meta-feature generation method for meta-learning-based algorithm ranking, which simultaneously improves the predictive performances of different meta-learners. (machine learning contribution).
- A new meta-learner called Approximate Ranking Tree Forests (ART forests) that performs very competitively when compared with several state-of-the-art meta-learners. We present both theoretical and empirical contributions. (machine learning contribution).
- A new experimental configuration that is more realistic than previous configurations for applications of meta-learning. We also present a default algorithm ranking list of 20 machine-learning algorithms based on their optimized performances over 466 datasets. (data mining contribution).
- A new ensemble learning algorithm for classification, regression and ranking with extensive empirical evaluation and theoretical analysis (machine learning contribution).
- A framework which can be used for designing new FMS (Full Model Selection) algorithms.
- A novel FMS algorithm (called GPS), which can be seen as a realization and an application of the proposed framework. Our experiments on real-world problems show that the proposed algorithm performs very competitively with PSMS, the state-of-the-art PSO-based FMS algorithm.
- We also examined the feasibility of using the model tree idea for speeding up the GPS algorithm. Our experimental results suggest that using the

perfect binary tree as the internal tree structure for GPS-based Full Model Tree is a viable approach when the empirical training complexity of GPS is worse than linear.

Significant parts of the research presented in this thesis have appeared in the following publications.

- Quan Sun and Bernhard Pfahringer. *Pairwise Meta-Rules for Better Meta-learning-based Algorithm Ranking.* Machine Learning, 93 (1), 141-161, Springer US, 2013.

- Quan Sun, Bernhard Pfahringer and Michael Mayo. *Towards a Framework for Designing Full Model Selection and Optimization Systems.* In Proceedings of the 11th International Workshop on Multiple Classifier Systems (MCS'13), Nanjing, China, LNCS 7872, pp. 259–270. Springer, Heidelberg, 2013.

- Quan Sun, Bernhard Pfahringer and Michael Mayo. *Full Model Selection in the Space of Data Mining Operators.* In Companion Material Proceedings of the ACM Conference on Genetic and Evolutionary Computation (GECCO'12), Philadelphia, United States, 2012.

- Quan Sun and Bernhard Pfahringer. *Bagging Ensemble Selection for Regression.* In Proceedings of the 25th Australasian Joint Conference on Artificial Intelligence (AI'12), Sydney, Australia, pages 695–706. Springer, 2012.

- Quan Sun. *Full Model Selection and Optimization.* In Proceedings of the 10th New Zealand Computer Science Research Student Conference (NZCSRSC'12), Dunedin, New Zealand, 2012.

- Quan Sun and Bernhard Pfahringer. *Bagging Ensemble Selection.* In

Proceedings of the 24th Australasian Joint Conference on Artificial Intelligence (AI'11), Perth, Australia, pages 251–260. Springer, 2011.

- Quan Sun. *Getting Even More Out of Ensemble Selection.* In Proceedings of the 9th New Zealand Computer Science Research Student Conference (NZCSRSC'11), Palmerston North, New Zealand, 2011.

1.3 Thesis Outline

This thesis is organised as follows:

- Chapter 2 gives an overview of existing meta-learning and meta-heuristic search-based full-model selection (FMS) methods and related works.
- In Chapter 3, we introduce a new meta-feature generation method and a novel meta-learning algorithm with both theoretical and empirical analysis.
- In Chapter 4, we discuss and propose alternative ensemble learning algorithms for meta-learning-based ranking problems.
- In Chapter 5 we discuss the ideas of the FMS approach. In particular, a novel FMS algorithm (GPS) is proposed and evaluated.
- Chapter 6 concludes the thesis with additional remarks and addresses some potentially important research directions for future work.

Chapter 2

Background

This chapter provides an introduction to the works that are related to the techniques proposed in this thesis. In this section, we also introduce the basic ideas of meta-learning and meta-heuristic search-based full-model selection techniques.

2.1 Meta-learning Systems

We first list several representative contributions that are related to meta-learning in general. The list is eclectic and by no means exhaustive. We discuss these contributions with an emphasis on the degree of relevance to the work presented in this thesis.

Early meta-learning systems are studied under the bias management framework. In [127], the authors proposed the Variable Bias Management System (VBMS) that dynamically alters evolving hypotheses, concept representations, and algorithms by selecting biases based on characteristics of a particular problem. The VBMS system and a similar system called STABB [153], which stands for "shift to a better bias", are among the earliest machine learning approaches to meta-learning.

Later in the signal processing community, the Hierarchical Mixtures of

Experts (HME) architecture was proposed to employ probabilistic methods in both the way it transfers the feature space and the way it combines expert predictions [161]. This type of system is usually regarded as meta-learning system because a meta-level procedure is applied to transfer the original feature or parameter space to a "higher" level.

Another early related direction is on concept learning and its relationship to data characteristics [126]. The idea is to view concepts as functions over instance space, which leads to geometric characteristics such as concept size and concentration. The authors conducted experiments to show that some data characteristics drastically affect the accuracy of concept learning.

Early meta-learning systems were also proposed under the lifelong reinforcement learning framework. The success-story algorithm proposed in [76] has been applied to complex maze tasks with a partially observable environment.

[158] discussed the term *inductive transfer* as a variant of meta-learning. The idea is that by transferring knowledge across related learning tasks, a learner can become "more experienced" and generalize better.

The Web-based Data Mining Advisor (DMA) system [64, 62, 21], is a research outcome of the METAL project (ESPRIT Nr. 26.357), which is a meta-learning system for automatically selecting learning algorithms for classification problems. The DMA system is also an attempt to apply the concept of data envelopment analysis (a nonparametric method in operations research and economics) [8, 4] to meta-learning. A rich set of meta-learning techniques was proposed by researchers involved in the METAL project, including new data characteristics (meta-features) and meta-learners [20, 141, 118, 22, 31, 23].

[13] proposed the concept of Intelligent Discovery Assistants (IDAs), which provides users with systematic enumerations of valid DM processes. The authors also illustrated a demonstration using data from the 1998 KDD CUP

competition[1]. Recently, [156] proposed the concept of experiment databases. The goal is to easily share machine learning experiments with the community, and automatically organize them in public databases. Currently, the system holds over 650,000 classification experiments.

2.1.1 Meta-learning Approaches

In the previous section, we have briefly reviewed several meta-learning systems with diverse rationales and views. Indeed, meta-learning is a rich field, usually explained as "learning to learn". Different researchers hold different views of exactly what the term "meta-learning" means. [158] has given a comprehensive review on the different perspectives on meta-learning. As a summary, we here list some of the views and approaches to meta-learning:

Meta-data based algorithm recommendation (ranking): in this approach, a meta-dataset is constructed by using various characteristics of a given dataset collection, such as general, statistical, information-theoretic characteristics, termed "meta-features". Another type of meta-feature is called "landmarkers", in which some properties (e.g., predictive performance) of a model learnt by an algorithm are used as meta-features. Usually landmarker-based features need to be computed relatively quickly (such as in $O(nlogn)$ [118]) otherwise one could simply run the candidate algorithms on the dataset. Then, the performances of the candidate algorithms are measured for each dataset in the collection. Given a meta-dataset, another algorithm, usually called a meta-learner, learns a model using the given meta-features. Given a new dataset, firstly the meta-features are calculated, and the expected or relative predictive performances of different algorithms can be predicted.

Ensemble learning can be viewed as a type of meta-learning. For example, the stacking generalization method [164, 136, 49] works by combining

[1]http://www.kdd.org/kdd-cup-1998-direct-marketing-profit-optimization

a number of base-level learning algorithms and using a meta-level learner to learn a linear function of the models produced by the base-level algorithms. Given a new dataset, the predictions of the base-level algorithms are combined (e.g., weighted voting) to provide the final prediction. Boosting [134] is another popular ensemble learning strategy in which the same base-level algorithm is used multiple times, where in each boosting iteration, a re-weighted training set is used. The final boosting prediction is also a combination of base-level models. Cascade generalization [60] is also an interesting ensemble learning strategy. The basic idea of cascade generalization is to use sequentially the set of classifiers, at each step performing an extension of the original data by the insertion of new attributes. The new attributes are derived from the probability class distribution given by a base classifier. Extensive theoretical and empirical studies have been done in the previous 20 years. Comprehensive reviews on ensemble techniques are given in [48, 171].

Dynamic bias selection works by altering the inductive bias of a learning algorithm to match the given problem. In this approach, the key properties of a learning algorithm are changed, such as the kernel type of the support vector machine algorithm, the splitting heuristic of a decision tree algorithm or a transformation to the input feature space. There are many different methods available in this approach. As an example, [7] described a framework for dynamic bias selection as a case-based meta-learning system. Another type of dynamic bias selection approach focuses on changing the representation of the feature space [153, 73]. More examples are given in [159].

Inductive transfer uses the idea of transferring acquired knowledge from one domain to help learning in another domain. This idea has been extensively studied in multi-task learning [36, 11]. Applications of inductive transfer and multi-task learning can be found in spam filtering [5] and Web search [38].

2.1.2 Inductive Bias

It is commonly accepted in the machine learning community that each algorithm has its own specific strengths and weaknesses or a restricted hypothesis space bias [26, 111, 84]. This is due to the assumptions each algorithm makes (based on its own inductive bias) in order to learn a model for a given dataset. This phenomenon has also been confirmed by a series of empirical studies [163, 137, 10, 35, 166]. In [111], the term "inductive bias" is described as the set of all the assumptions that, together with the training data, deductively justify the predictions that the algorithm makes for unseen instances. There are at least four types of inductive bias, namely search bias, language bias, the "discriminative or generative" bias and validation bias.

Search bias—Consider the ID3 [121] decision tree algorithm as shown in Algorithm 2.1. Given a classification dataset, there are many decision trees (hypotheses) consistent with the examples in the data. The ID3 algorithm uses the information gain heuristic so it is biased to favour trees that place high information gain features close to the root over those that do not. This type of bias is usually called the preference or search bias [111].

Language bias—Assume we have a regression dataset consisting of the following features:

- f_1: the number of sales of product A
- f_2: the number of sales of product B
- f_3: the number of sales of product C

The target (response) variable is the number of sales of product X. If we use the linear regression algorithm, then the learned model is a linear function of the form:

$$\hat{H} = w_0 + w_1 f_1 + w_2 f_2 + w_3 f_3,$$

Algorithm 2.1 The ID3 Algorithm [121, 105]

if all examples have the same class label **then**
  return a leaf with that class label
else if there are no features left to test **then**
  return a leaf with the most common class label
else
  Choose the feature $\hat{F}$ that maximises the information gain of the set of examples S to be the next node
  Add a branch from the node for each possible value f of $\hat{F}$
  for each branch: **do**
    calculate S_f by removing $\hat{F}$ from the set of features
    recursively call the algorithm with S_f, to compute the gain relative to the current set of examples
  end for
end if

where w_0 through w_3 are numerical coefficients. In this case, the decision to use the multiple (linear) regression algorithm introduces a restriction bias or language bias because nonlinear combinations of the features are eliminated by the algorithm.

"Discriminative or generative" bias—Consider a simple binary classification dataset as shown in Table 2.1.2. In the *discriminative* approach, we could use a rule learner to build a binary classification model. For example, the RIPPER rule learner [42] gives us the following rules:

- **If** (weight $\leq$ 39) **Then** class=A
- **If** (weight $\geq$ 45) **Then** class=A
- **Else** class=B

However, in the *generative* approach, assuming we know the values of the

Table 2.1: Weight (in grams) of a kiwifruit from two farms

Instances	38	40	42	39	40	44	41	37	41	42	45	46	45
Class label	A	B	B	A	B	B	A	A	B	B	A	A	B

population means μ_A and μ_B, and the weight of a kiwifruit is normally distributed in both farms (and with equal variance), then we can easily obtain an optimal Bayesian classifier [149] predicting "A" if weight $\leq \frac{\mu_A+\mu_B}{2}$, and "B" otherwise. As pointed out in [134], when the assumptions we have made about the data are valid, good generalization is assured. In this case, the decision to use a certain algorithm introduces a "discriminative or generative" bias.

Validation bias—[111] notes that some algorithms combine more than one type of bias. Therefore, end-users of machine learning are supposed to be able to choose a proper algorithm for a given task. The selection procedure is usually guided by an evaluation mechanism [27], e.g., cross-validation [46, 61]. However, [6] provided empirical evidence showing that the bootstrap estimator has lower variance than cross-validation has for estimating the true accuracy of the FOIL rule learner [122], but that it is also biased. The estimate of accuracy provided by cross-validation has high variance but is approximately unbiased. This implies that the choice of performance estimation method is related to the algorithm being evaluated [84]. Thus, the decision to use a certain performance estimation method also introduces a validation bias.

2.1.3 Theoretical Considerations

Theoretical motivations as well as some arguments on meta-learning research are due to the No Free Lunch (NFL) theorem [165] and the Law of Conservation for Generalization Performance (LCG) [132]. The basic idea is: "*When taken across all learning tasks, the generalization performance of any learner sums*

to 0". A detailed review on NFL and LCG for meta-learning can be found in [62]. Here, we briefly discuss the basic concepts.

For a given learning algorithm, a model M is induced, which defines a class probability distribution p over the instances space. An Ultimate Learning Algorithm (ULA) is a learning algorithm that induces a model M^*, such that:

$$\forall M' \neq M^* \quad E(D(p^*, p^\Omega)) \leq E(D(p', p^\Omega)), \tag{2.1}$$

where the expectation is computed for a given training/test set sample of the instance space, over the entire function space, D is some appropriate distance measure and p^*, p^Ω and p' are class probability distributions. As pointed out in [62], here we are interested in an "ultimate" but not a "universal" algorithm because "universal" generally means either:

1. (mathematically) applicable independent of any assumptions, or

2. applicable throughout the entire universe.

When asking about the real world (*our universe*) it is the second definition that is important; what could happen in other conceivable universes is of no possible interest to us. As pointed out in multiple research papers [63, 62]: "*In our universe, we embrace the weak assumption of Machine Learning, namely that the process that presents us with learning problems induces a non-uniform probability distribution over the possible functions. Hence, assuming the existence of a ULA is not in contradiction with the NFL theorem*".

2.1.4 Learning to Recommend (Rank) Algorithms

There are many other approaches in meta-learning in which the meta-data or meta-knowledge is used to improve the performance of a learning system. As we have mentioned in the previous sections, the term "meta-learning" has been used by various researchers in different ways, and it is therefore difficult to give a concise definition that everyone would applaud. Our work

in this thesis focuses on meta-learning for algorithm/parameter ranking and related algorithms (meta-learners). Next, we discuss the general ideas of meta-learning-based algorithm ranking and motivations.

As we have discussed in Section 2.1.2, there is no single best algorithm to be used in all problems since most algorithms have an inductive bias, theoretically due to the NFL theorem [165]. The brute-force approach is to try all the algorithms (with different parameter settings) for a given dataset at hand. In practice this is usually not feasible if there are too many alternative algorithms available.

Consider the use case of a data analyst. One daily task of the data analyst normally involves preparation of a dataset that can be processed by a learning algorithm. Usually, there are several algorithms available, so the analyst needs to select one of them for a business goal, e.g., a binary classification algorithm for a customer churn study or a regression algorithm for online sales prediction. Given the selected algorithm, the analyst also needs to further fine tune the parameters in order to obtain a stable and accurate model. The choice of algorithm and parameter is guided by the performance estimation methodology that the analyst uses.

One common practice is to use the "trial and error (generate and test)" strategy. Taking algorithm selection as an example, the analyst could get the performance estimations of the algorithms based on cross-validation, and then use statistical tests to determine the best algorithm to use. As long as the performance estimation method is sound and valid, the analyst would usually get a relatively good model for practical use. Although feasible, this strategy may still require a reasonable amount of computing time, especially when there are many algorithms available. Also, in a business or industrial environment, the end-users of machine learning techniques are not necessarily machine learning experts. Therefore, the choice of which algorithm(s) to use depends on the

dataset at hand, and systems that can provide such recommendations would be very useful [111, 21].

The next question is "which type of recommendation should a meta-learning system provide to the end-user?". In this thesis we follow the reasons and motivations described in [23, 21]: "*... when searching a topic on the Web, one may investigate several links. The same can also apply to a data analysis task if enough resources are available to try out more than one algorithm. Since we do not know how many algorithms the user might actually want to select, we provide a ranking of all the algorithms...*".

In [21], the user's goal is stated as: "*saving time by reducing the number of alternative algorithms tried out on a given problem with minimal loss in the quality of the results obtained when compared to the best possible ones*". Meta-learning for algorithm ranking uses a general machine learning approach to generate meta-knowledge mapping the characteristics of a dataset, captured by meta-features, to the relative performances of the available algorithms.

The advantage of the above approach is that high-quality algorithm or parameter ranking can be done on the fly, e.g., in seconds. This promise is particularly important for business domains that require rapid deployment of analytical techniques, e.g., stock market prediction and customer response prediction, since high-quality models need to be built every day or even every minute.

Apart from the industrial demand, in machine learning research, meta-learning has also been used for initializing and boosting the performance of evolutionary-algorithm-based model selection techniques. Recent work can be found in [124]. The work by [138] discusses cross-disciplinary perspectives on meta-learning. Recent research seems to suggest that by making explicit the assumptions underlying machine learning research, it is possible to show that meta-learning offers a viable mechanism to build a "general-purpose" learning

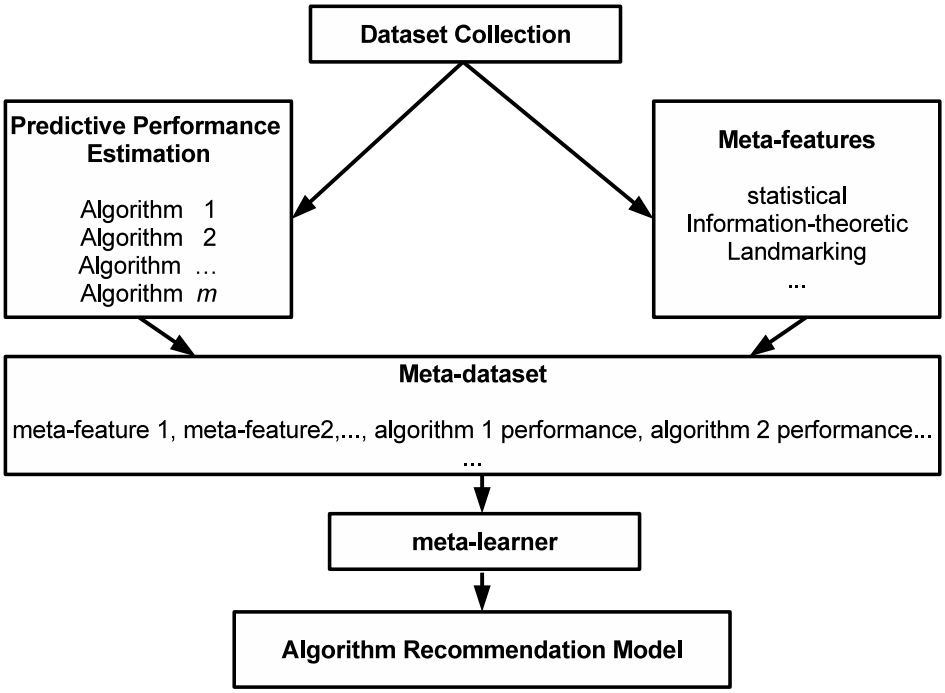


Figure 2.1: Meta-learning-based algorithm recommendation

algorithm [63, 62]. For a comprehensive review of meta-learning research and its applications, we refer the reader to [158, 62, 21].

2.1.5 Instance-based Learning for Algorithm Ranking

Figure 2.1 illustrates the conceptual graph of meta-learning-based algorithm ranking (recommendation). Next we briefly introduce a simple meta-learning procedure which is based on the k-nearest neighbours (k-NN) ranking method. This approach has been used in [23], where it is called the instance-based learning (IBL) approach, since k-NN is a form of IBL.

In [23], the standard Euclidean distance function is employed to determine the quantitative similarity between examples. Algorithm 2.2 shows the pseudocode for the k-NN ranking algorithm for algorithm recommendation. This algorithm is used as a benchmark meta-learner in our meta-learning experiments. We will give more details about this algorithm in Section 3.

Algorithm 2.2 The k-NN ranking algorithm for meta-learning-based algorithm recommendation [23, 21]

1. Find the subset of k datasets containing the ones that are most "similar" to the new dataset in terms of distances in the meta-feature space.
2. Combine the target values (e.g., accuracy measurements or ranking positions) of the k examples (datasets) to generate a prediction for a new example.

2.1.6 Meta-features

Over the past two decades, researchers in meta-learning have designed a rich set of meta-features. These meta-features are used to describe the characteristics of a dataset. In this thesis, we use the term "attribute" or "base-level feature" to refer to a feature in the original dataset. We also use the terms "data characteristics" and "meta-features" interchangeably.

Traditionally, in terms of attributes, meta-features are clustered in the following categories [84, 21]:

- features that describe the nature of attributes
- features that describe attributes
- features that describe relationships between attributes
- features that describe relationships between attributes and the target.

As is true in general for machine learning, the performance of meta-learning depends crucially on the quality of the meta-features available. In terms of the meta-features, existing meta-learning systems are mainly based on four types of meta-features: simple, statistical, information-theoretic and landmarking-based meta-features, or **SIL** for short.

- **Simple**—Number of instances, number of attributes, number of classes or targets...

- **Statistical**—Mean kurtosis of attributes, mean skewness of attributes...
- **Information-theoretic**—Class entropy, mean entropy of attributes, noise-signal ratio [84]...
- **Landmarking**—Performance of a simple classifier [118], e.g., a decision stump [81] or linear discriminant [56].

Thorough reviews and explanations of these meta-features are given in [20, 118, 84, 140, 62, 21]. Recently, more meta-feature sets have been developed and tested, for example, the learning-curve-based meta-features [96], tree-structure-information-based [12] and histogram-based meta-features [84].

In Section 3, we propose a novel method that generates "meta-level" meta-features from the "base-level" meta-features via rule induction, where the "base-level" meta-features can come from any of the meta-feature sets mentioned above.

Table 2.2 gives a summary of meta-features that are used as the base-level meta-features in this thesis.

2.1.7 Parameter and Full Model Recommendation

In the previous sections, we have discussed using the meta-learning approach for algorithm recommendation. For parameter or full model recommendation, the idea is similar. Theoretically, given a set of data mining tools (or a set of parameters of an algorithm) that can be used for building a predictive model on a particular dataset, each combination of the available tools can be viewed as a single "algorithm".

For practical meta-learning applications, if the number of tools (algorithms) is relatively large, then simply evaluating each combination is not feasible. For example, if we had 10 tools, there will be $10! = 3,628,800$ possible permutations! Although we could reduce the possible permutations by introducing

ID	Name	ID	Name
1	RandomTreeDepth1AUC_K=0	41	J48.0001.kappa
2	RandomTreeDepth2AUC_K=0	42	J48.001.kappa
3	RandomTreeDepth3AUC_K=0	43	ObliviousTree.depth.1.ErrRate
4	RandomTreeDepth1AUC_K=0	44	ObliviousTree.depth.1.AUC
5	RandomTreeDepth2AUC_K=0	45	ObliviousTree.depth.2.ErrRate
6	RandomTreeDepth3AUC_K=0	46	ObliviousTree.depth.2.AUC
7	RandomTreeDepth1ErrRate_K=1	47	ObliviousTree.depth.3.ErrRate
8	RandomTreeDepth1Kappa_K=1	48	ObliviousTree.depth.3.AUC
9	RandomTreeDepth2ErrRate_K=1	49	ObliviousTree.depth.1.kappa
10	RandomTreeDepth2Kappa_K=1	50	ObliviousTree.depth.2.kappa
11	RandomTreeDepth3ErrRate_K=1	51	ObliviousTree.depth.3.kappa
12	RandomTreeDepth3Kappa_K=1	52	MeanMeansOfNumericAtts
13	RandomTreeDepth1ErrRate_K=1	53	MeanStdDevOfNumericAtts
14	RandomTreeDepth1Kappa_K=1	54	MeanKurtosisOfNumericAtts
15	RandomTreeDepth2ErrRate_K=1	55	MeanSkewnessOfNumericAtts
16	RandomTreeDepth2Kappa_K=1	56	NumAttributes
17	RandomTreeDepth3ErrRate_K=1	57	Dimensionality
18	RandomTreeDepth3Kappa_K=1	58	NumNominalAtts
19	DecisionStumpErrRate	59	NumNumericAtts
20	DecisionStumpAUC	60	PercentageOfNominalAtts
21	NBErrRate	61	PercentageOfNumericAtts
22	NBAUC2fCV	62	NumBinaryAtts
23	NBAUC5fCV	63	PercentageOfBinaryAtts
24	NBAUC10fCV	64	ClassCount
25	REPTreeDepth1ErrRate	65	PositivePercentage
26	REPTreeDepth1AUC	66	NegativePercentage
27	REPTreeDepth2ErrRate	67	DefaultAccuracy
28	REPTreeDepth2AUC	68	IncompleteInstanceCount
29	REPTreeDepth3ErrRate	69	InstanceCount
30	REPTreeDepth3AUC	70	NumMissingValues
31	REPTreeDepth1Kappa	71	PercentageOfMissingValues
32	REPTreeDepth2Kappa	72	MaxNominalAttDistinctValues
33	REPTreeDepth3Kappa	73	MinNominalAttDistinctValues
34	J48.00001.ErrRate	74	MeanNominalAttDistinctValues
35	J48.00001.AUC	75	StdvNominalAttDistinctValues
36	J48.0001.ErrRate	76	ClassEntropy
37	J48.0001.AUC	77	MeanAttributeEntropy
38	J48.001.ErrRate	78	MeanMutualInformation
39	J48.001.AUC	79	EquivalentNumberOfAtts
40	J48.00001.kappa	80	NoiseToSignalRatio

Figure 2.2: The 80 SIL meta-features that are used as the base-level meta-features.

some hard constraints, the space could still be large. In this situation, we could only manually or randomly select a sample of combinations of tools and evaluate their performances. If the sample size is still too large, then we could use a subset (e.g., the top n best solutions) to generate a meta-learning dataset.

Another approach is to "reduce" the permutation space by defining some templates. For example, assuming we have five tools $\{A,\ B,\ C,\ D,\ E\}$, the total number of permutations is $5! = 120$. If we know that solutions (permutations) with the two objects $< A \rightharpoonup B >$ (applying A and then immediately

B) are likely to yield a good model, then we could use $< A \rightharpoonup B >$ as a new object reducing the space to $|\{AB,\ C,\ D,\ E\}| = 4! = 24$.

2.2 Meta-heuristic Search based Full Model Selection

In Section 2.1, we discussed the meta-learning approach to algorithm and parameter recommendation. In this section, we focus on the meta-heuristic-search[2] based full model selection (FMS) approach.

Here, the FMS problem is defined in the sense of choosing a subset of (machine learning and data mining) tools, including preprocessing, feature selection, learning algorithms and so on, from a set of available tools in order to produce a good predictive model. In this approach, we assume the end-users of the system do not require instant recommendation, i.e., for a relatively large dataset (hundreds of thousands of instances and 20 or 30 features), making a useful recommendation in several hours is acceptable.

Although, the computation cost of the FMS approach can be significantly reduced using parallel computing techniques, the emphasis in the current thesis is on the predictive performance of the solutions found by the proposed FMS techniques.

2.2.1 Solving FMS as an Optimization Problem

In its nature, FMS is a standard optimization problem. Given $g : S \mapsto \mathbb{R}$, the problem is to find:

$$\arg\min_{v \in S} \ \mathrm{g(v)}, \tag{2.2}$$

where S is a search space, g is the objective function (such as a predictive performance estimation function) and v is a vector of decision variables (such

[2]Some researchers also call it *meta-heuristics* [102]

as a subset of data mining and machine learning tools). When S is *discrete*, FMS also belongs to the combinatorial optimization problems (COPs):

The FMS Problem

Instance: A set of data mining tools.

Task: Find a subset of the tools such that its estimated predictive performance on a given dataset is maxima.

2.2.2 The Search Space

We define a search space that consists of all data mining actions (operators) that are available to a given dataset for a user-specified predictive modelling goal, such as a set of outlier filters, a set of feature selection methods, a set of data transformation techniques and a set of base learning algorithms. This search space is called "the space of data mining operators (DMO)", or simply "the DMO space".

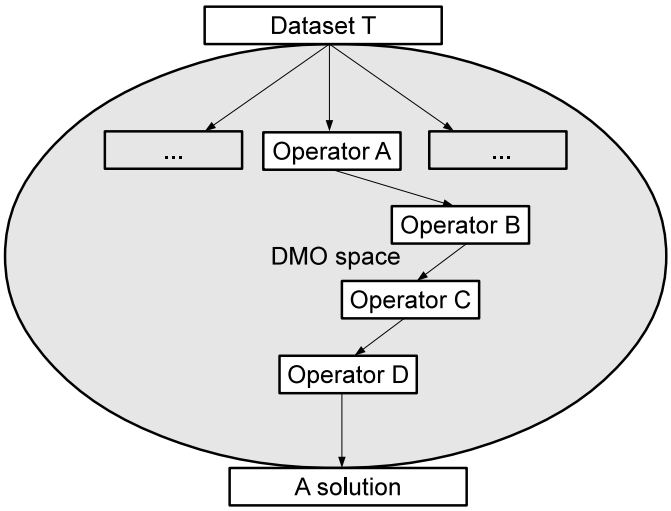


Figure 2.3: An illustration of the DMO space

In the DMO space, a data mining solution is abstracted as a directed acyclic graph (DAG) consisting of DMOs that are connected based on some relations: see Figure 2.3 for an illustration. For simplicity, in Figure 2.3 we

consider that an optimal data mining solution is given by a DAG defined by four DMOs (A, B, C and D) for dataset T. The DMO space is represented by the largest oval, which consists of all DMOs applicable to T. The directed arrows represent the relationships (action rules) in the DAG. If Operator A is an outlier filter, Operator B is a feature reduction method, Operator C is a decision tree algorithm, and Operator D is a post-processing method, the DAG can be interpreted as follows: given a dataset T, in an optimal solution we first use the outlier detection method (DMO A) to remove outliers, and then we employ the feature selection method (DMO B) to remove useless features, and then build a decision tree model (DMO C), and finally, we use a probability calibration method (DMO D) to calculate the model outputs. This is a very large search space because in theory there exists an arbitrary number of DMOs (including an arbitrary number of link directions, node orders and arrangements). Therefore brute force based search will definitely fail.

2.2.3 Meta-heuristic Search Algorithms for FMS

At the concept level, meta-heuristic search employs heuristics to choose for its actions. The heuristics lead to a solution and help to reduce the number of alternatives. The advantages of using a heuristic search approach for problem solving include:

- Finding a solution for an average problem efficiently.

- Finding a reasonably good, but not necessarily optimal solution efficiently

Algorithm 2.3 shows the pseudocode of a simple meta-heuristic algorithm.

Algorithm 2.3 A typical meta-heuristic search procedure [113]

Obtain an initial solution set θ_0 and set $k = 0$.

Repeat

Identify the neighbourhood $N(\theta_k)$ of the current solution(s).

Select candidate solution(s) $\{\theta^c\} \subset N(\theta_k)$ from the neighborhood.

Accept the candidate(s) and set $\theta_{k+1} = \theta^c$ or reject it and set $\theta_{k+1} = \theta_k$.

Increment $k = k + 1$.

Until stopping criterion is satisfied

Next we briefly review two commonly used meta-heuristic search algorithms.

Genetic Algorithms—Genetic algorithms (GAs) [77, 69, 110, 133, 111, 9] have proved to be a versatile and effective approach for solving combinatorial optimization problems. Rather than searching from the simple to the complex, as many other solutions do, GAs generate successor hypotheses by repeated mutation and recombination of parts of the existing best hypotheses. At each iteration, the current population (a collection of hypotheses) is updated by replacing certain hypotheses by offsprings of the best hypotheses. Algorithm 2.4 gives a simple version of the genetic algorithm.

In machine learning, a large number of applications of GAs can be found in the parameter tuning [70, 69, 169, 99, 101] and the feature selection domains [155, 154, 80, 32, 95]. GA also has a wide range of industry applications, including bioinformatics [89, 72], computer-automated design [100], finance and economics [106, 107], power electronics design [170], rare event analysis [44] and many others.

In this research, GAs will be used to search optimal data mining solutions in the DMO space, and their performance will be compared to the performance of other heuristic search algorithms, such as particle swarm optimization (PSO) algorithms.

Algorithm 2.4 Basic Genetic Algorithm Steps

Generate a population of structures

Repeat

    Test the structures for quality

    Select structures to reproduce

    Produce new variations of selected structures

    Replace old structures with new ones

Until stopping criterion is satisfied

Particle Swarm Optimization (PSO) [87]—PSO is inspired by the behaviour of biological communities that exhibit both individual and social behavior (for example, flocks of birds and swarms of bees). Members of such societies share common goals that are realized by interacting with their environment. In PSO, each solution is called a particle, and a set of particles is called a swarm. In the abstract sense, particles have a velocity value that they use for flying in the search space. Algorithm 2.5 is a simple version of PSO. The differences between PSO and evolutionary algorithms have been highlighted by several authors [51]. Inspired by the ideas of GAs and PSO, in this thesis, we will attempt to propose new meta-heuristic search algorithms for FMS.

2.2.4 Alternative Meta-heuristic Search Algorithms

In this section we briefly review several alternative meta-heuristic search algorithms that could also be used for the FMS problem.

Tabu Search—Tabu search [68, 65, 67] is a local search algorithm. It uses a neighbourhood procedure to generate an improved solution in its neighbourhood space, until stopping criterion has been satisfied. Tabu search keeps a "tabu" list of solutions in memory so that the algorithm does not consider those solutions in the list until they are sufficiently far in the past. At least

Algorithm 2.5 Basic Particle Swarm Optimization Steps

```
Initialize swarm
Compute particle scores
Locate leader and set personal bests
    For each particle in the swarm
        Calculate velocity
        Update particle position
        Compute particle score
        Update
    Update global particle
Until a stopping criterion is satisfied
```

three types of memory structures have been studied for tabu search, namely, short-term, intermediate-term and long-term [66]. Tabu Search is usually implemented as a variant of steepest ascent with replacement [102].

Simulated Annealing—Simulated annealing [90, 37] gets its name from annealing, a process of cooling molten metal. The key to the process is the rate at which temperature decreases as the metal is cooled. This notion of slow cooling is implemented in the simulated annealing algorithm as a slow decrease in the chance of accepting worse solutions. The simulated annealing algorithm is an adaptation of the Metropolis-Hastings algorithm [108]. The basic idea of the algorithm is that if a newly generated solution is worse than the original solution, it may still replace the original with a certain probability controlled by a temperature parameter t (the rate of decreasing t is called the algorithm's schedule).

Bayesian Optimization Algorithm (BOA)—The general procedure of the BOA algorithm is similar to a GA, but the recombination operator of the GA is replaced by probability estimation and sampling [116, 115, 83]. First an initial population is generated randomly. In each iteration, the algorithm

estimates the probability distribution of a selected set of promising solutions. A new set of solutions is then generated according to the estimate. Precisely, BOA employs Bayesian techniques to estimate the probability distribution.

As we have discussed earlier, the FMS problem belongs to the field of combinational optimization problems where many algorithms are available, including the algorithms we have mentioned above. In Section 5 we will focus on the GA and the PSO algorithms, and we leave the other algorithms for future research.

2.2.5 Hyperparameter Selection

In this section, we review two hyperparameter selection techniques.

Grid Search—Grid search is an exhaustive approach which does not apply any optimization during the search process. It searches through a manually specified subset of the hyperparameter space of a learning algorithm guided by a predictive performance estimation method, e.g., cross-validation. For continuous hyperparameters, one needs to set bounds and transfer them into a discretized space. Grid Search has the obvious advantage that every hyperparameter vector in the grid is evaluated before the best one can be determined. Thus if the grid is chosen appropriately, it guarantees that the best hyperparameter vector is close to the globally optimal parameters rather than in a suboptimal area [103].

Online Gaussian Process—The idea of using an online Gaussian process model for hyperparameter selection was introduced in [59]. The idea is to sample new points (a vector of hyperparameters), where the expected improvement (given already evaluated hyperparameter vectors) is maximal, in order to find good hyperparameters efficiently. The performance of this method and some variants has been empirically examined in [59, 103]. In general the Gaussian process based approach finds hyperparameters that are not significantly worse

than those found by grid search, in much shorter time. Recently, [139] extended the Gaussian process framework in a parallel computing environment.

The above techniques are designed for hyperparameter tuning for single algorithms. Next, we review techniques that have been applied to FMS.

2.2.6 The PSMS system

The PSMS (particle swarm full model selection) system proposed in [51], is an application of particle swarm optimisation (PSO) to the problem of full model selection for classification problems. In total, 3 feature transformation objects, 13 feature selection objects and 10 classifier objects are used in the PSMS system. A full model is defined as a 16-dimensional particle position [51]:

$$\mathbf{x}_i =< x_{i,pre}, y_{i,1...Npre}, x_{i,fs}, y_{i,1,...N_{fs}}, x_{i,sel}, x_{i,class}, y_{i,1...Nclass} >,$$

where $x_{i,pre\in\{1,...,8\}}$ represents a combination of feature transformation methods. Each combination is represented by a binary vector of size 3 (i.e., the number of feature transformation methods considered), and there are $2^3 = 8$ possible combinations. Each element of the binary vector represents a single feature transformation method. Variable $y_{i,1...N_{pre}}$ codifies the hyperparameters for the selected combination of feature transformation methods, and $N_{pre} = 3$ because each feature transformation method has a single hyperparameter. The order of the feature transformation methods is fixed in the PSMS system. $x_{i,fs\in\{0,...,12\}}$ represents the index (ID) of the feature selection method used by the model, and $y_{i,1...N_{fs}}$ its respective hyperparameters; N_{fs} is set to the maximum number of hyperparameters that any feature selection method can take. $x_{i,sel}$ is a binary variable that indicates whether feature transformation should be performed before feature selection or vice versa. $x_{i,class} \in \{1, ..., 10\}$ represents the classifier selected and $y_{i,1,...N_{class}}$ its respective hyperparameters. N_{class} is the the maximum number of hyperparameters that a classifier can

take. The above numerical codification is supposed to be decoded and used with the chain grouping object to obtain a full model from a particle position $\mathbf{x}_i$.

Based on the experimental results in [51], the PSMS system shows promising results when it is compared with the Pattern Search (PS) strategy [112, 14] for the FMS problem. The system also shows competitive performance compared with other search strategies in a model selection competition.

We have briefly introduced the PSMS system. From the system architecture point of view, PSMS assumes a full model has three components: feature transformation, feature selection, and algorithm, which is a simplified presentation of a full model, because a full model may have other components, such as data cleansing and data sampling.

In Section 5, we will introduce a novel search strategy for the FMS problem, which covers five data mining components, namely, data cleansing, data sampling, feature transformation, feature selection and algorithm DMOs.

2.3 Model Selection and Performance Estimation Methodologies

Both the meta-learning and the meta-heuristic search approaches to the FMS problem need to estimate the predictive performance of an algorithm (or a set of data mining tools). In this section, we review several performance estimation methodologies that are commonly used in data mining and machine learning.

2.3.1 VC Dimension and Structural Risk Minimization

The VC (Vapnik-Chervonenkis) dimension [157] is a measure of the capacity of a statistical classification algorithm, defined as the cardinality of the largest set of points that the algorithm can shatter. In terms of model selection, the

theory of VC dimension can be used to predict a probabilistic upper bound on the test error of a classification model [157, 17].

Let d be the VC dimension of a hypothesis h, $err(h)$ be the true error and $\widehat{err(h)}$ be the training error. One of the core results of VC dimension is that with probability $1 - \eta$:

$$err(h) \leq \widehat{err(h)} + \sqrt{\frac{d(log(2N/d) + 1) - log(\eta/4)}{N}},$$

where N is the number of training instances.

The above bound provides a way to estimate the true error (or error on unseen data) based on $\widehat{err(h)}$ and d. Although VC dimension offers a theoretically sound approach for estimating the true error (generalisation error) of a model, in practice, it is not always possible to compute d. Also, there is an implicit assumption that has to be made in order to derive the above bound, which is $N \gg d$. The VC dimension theory based model selection framework is called structural risk minimization (SRM), which was introduced by Vapnik. [30] has introduced the use of SRM for support vector machine model selection. In this thesis, we do not use the SRM framework for performance estimation because the VC dimensions of the algorithms (or solutions consisting of a set of tools) are not readily available.

2.3.2 Akaike and Bayesian Information Criterion

The Akaike information criterion (AIC) [1] is a measure of the relative goodness of fit of a statistical model. The general formula is:

$$AIC = 2k - 2log(L),$$

where k is the number of parameters in a statistical model, and L is the maximized value of the likelihood function for the estimated model. Another closely related criterion is the Bayesian information criterion (BIC) [135] which

penalizes the number of parameters more strongly than AIC. The formula for the BIC is:

$$BIC = -2log(L) + klog(N),$$

where k and L are as same as in AIC, and N is the sample size.

Both AIC and BIC have been widely used for model selection in linear regression and time series applications. As they are designed for statistical and (exponential family) probabilistic models, they need to be combined with maximum-likelihood estimation (MLE) approaches. However, the algorithms (solutions) examined in this thesis are not only conventional statistical models but also other types of models (e.g., trees, rules), which may or may not be solvable using MLE.

2.3.3 Cross-validation

Cross-validation [46, 61, 91] is a statistical method for estimating the true error of algorithms by dividing data into folds. One fold is used for testing, while the remaining ones are used for training. There are several types of cross-validation, such as k-fold cross-validation, repeated sub-sampling validation and leave-one-out cross-validation. For example, in k-fold cross-validation, the dataset F is partitioned into k folds, F_i, for $i = 1, ..., k$. Each fold is in turn left out in training and used as a test set. The k-fold cross-validation estimate of the true error is the overall proportion of error committed on all folds [18]. The procedure can be repeated and scores can be averaged. Cross-validation has been used widely in model selection [119], model/algorithm comparison [35] and feature selection [78]. Cross-validation estimators are often pessimistic [18], since the algorithm under testing uses smaller training sets for training. Another drawback of cross-validation is due to its variance for small sample sizes [19] and the computational cost.

As there is no single best true error estimation method in data mining

and machine learning, and datasets used in this thesis are not small sample datasets, in this thesis, cross-validation is used as the main methodology for true error and predictive performance estimation of algorithms.

2.4 Summary

In this chapter, we gave an overview of existing meta-learning and meta-heuristic search-based full-model selection (FMS) methods and related works. In the next chapter, we will introduce a new meta-feature generation method and a novel meta-learning algorithm with both theoretical and empirical analysis.

Chapter 3

Pairwise Meta-Rules and Approximate Ranking Trees

In this chapter[1], we present a novel meta-feature generation method in the context of meta-learning, which is based on rules that compare the performance of individual base learners in a one-against-one manner. In addition to these new meta-features, we also introduce a new meta-learner called Approximate Ranking Tree Forests (ART forests) that performs very competitively when compared with several state-of-the-art meta-learners. Our experimental results are based on a large collection of datasets and show that the proposed new techniques can improve the overall performance of meta-learning for algorithm ranking significantly. A key point in our approach is that each performance figure of any base learner for any specific dataset is generated by optimising the parameters of the base learner separately for each dataset.

This chapter makes three contributions, a novel meta-feature generator for meta-learning, a new meta-learner, and a more appropriate experimental configuration, for improving the overall performance of meta-learning for

[1]Part of the research presented in this chapter has appeared in [146]: Quan Sun and Bernhard Pfahringer. *Pairwise Meta-Rules for Better Meta-Learning-Based Algorithm Ranking.* Machine Learning, 93(1):141-161, 2013

algorithm ranking.

3.1 Background

"Meta-learning" is a broad field; in this chapter, for simplicity, the term meta-learning is used in the sense of "meta-learning for algorithm ranking or recommendation". We first introduce the mechanics of meta-learning. The basic steps of a meta-learning task are as follows. Firstly, a set of datasets is collected; secondly, we need to define some meta-features as the characteristics of each dataset, e.g., number of instances, number of numeric/categorical features and so on (more details on meta-features are given in Section 3.2). Thirdly, we estimate the predictive performance of the available algorithms (e.g., using cross-validation) for every dataset in the dataset collection. Thus, for each dataset we get a list of available algorithms with their performance estimations.

Given the above information, we can construct a meta-dataset, which is a $n \times m$ data matrix, where $m = m_f + m_t$. Here, m is the sum of the number of meta-features m_f and the number of algorithms m_t, and n is the number of datasets. Figure 3.1 (left-hand side table) shows an example of a meta-dataset in a general multi-target regression setting.

For algorithm recommendation, our goal is not to predict the absolute expected performance of any algorithm, but rather the relative performance between algorithms. Thus, the meta-dataset can be transformed to represent the rankings of the algorithms. Figure 3.1 (from left-hand side to right-hand side table) shows an example of how a "raw" meta-dataset is transformed. In this case, ranking is a special case of the general multi-target regression setting. Then, we need a meta-learner, here called a "ranker", which takes the $n \times m$ matrix as a training dataset to learn a ranking model. Given a new dataset, we first calculate its meta-features and use the meta-features (e.g., the f_1, f_2, f_3 values for the example in Figure 3.1) as input to the ranker. The

	f1	f2	f3	t1	t2	t3
d1	100	0.52	-1.0	0.85	0.95	0.72
d2	300	0.45	2.0	0.55	0.50	0.70
d3	450	0.77	1.5	0.71	0.83	0.90

	f1	f2	f3	t1	t2	t3
d1	100	0.52	-1.0	2	1	3
d2	300	0.45	2.0	2	3	1
d3	450	0.77	1.5	3	2	1

Figure 3.1: Two formats of a meta-dataset constructed from three datasets (d_1, d_2, d_3), three meta-features (f_1, f_2, f_3) and three algorithms (t_1, t_2, t_3). Left-hand side table shows the format using the actual performance values (higher is better) as targets; the right-hand side table shows a format using the ranks (lower is better) as targets.

ranker finally returns a ranked list of all algorithms. As is true in general for machine learning, the performance of meta-learning depends crucially on the quality of both the meta-features and the meta-learners available. In addition, the performance of the base-level learners must also be estimated to a high quality to enable successful meta-learning.

Existing meta-learning systems are mainly based on three types of meta-features: Statistical, Information-theoretic and Landmarking-based meta-features, or **SIL** for short. Thorough reviews and explanations of these meta-features can be found in [20, 118, 84, 140]. Recently, more meta-feature sets have been developed and tested, for example, the learning-curve-based meta-features [96], tree-structure-information-based [12] and histogram-based meta-features [84].

In this chapter, we propose a novel method for meta-learning that generates "meta-level" meta-features from the "base-level" meta-features via rule induction that tries to predict the better algorithm for each pair of algorithms. Here the "base-level" meta-features can come from any of the meta-feature sets mentioned above. Meta-features generated by the proposed method (described in Section 3.2) contain pairwise information between algorithms. Adding them to the original feature space can help to improve the performance of many meta-learners.

Next, we will discuss several ranking approaches from an *algorithmic* perspective as the underlying multi-target or ranking problem is being studied in

various fields, including social sciences, economics and mathematical sciences. Although the terminologies and presentations of the same models can be quite different for different fields, the algorithmic properties are similar in concept and relatively easy to describe.

3.1.1 The *k*-Nearest Neighbors Approach

The k-NN ranking approach has two steps: the nearest neighbor search step and the ranking generation step. In the first step, given a new dataset, we first calculate its meta-features to construct an instance as a query (an m_f-value array). Then, we select a set of instances (nearest neighbors) in the training set (the $n \times m_f$ data matrix) that are similar to the query instance. The similarity between instances is usually based on a distance function, e.g., Euclidean distance.

In the second step, we combine the rankings of the nearest neighbors to generate an aggregated algorithm ranking for the new dataset. For our experiments, we use the average ranks method described in [21]. Let $R_{i,j}$ be the rank of algorithm $T_j, j = 1, ..., t$ on dataset i, where t is the number of algorithms. The average rank for each algorithm T_j is defined as: $\bar{R}_j = (\sum_{i=1}^{k} R_{i,j})/k$, where k is the number of nearest neighbors. The k-NN ranker's performance is related to the value of k, and appropriate values for k are usually determined by cross-validation. The k-NN ranker is often used as a benchmark learner for testing the performance of different meta-feature sets.

3.1.2 The Binary Pairwise Classification Approach

Given the $n \times m$ data matrix as the training data, multiple binary (pairwise) classification models can be used to construct a ranking model. For example, if there were three meta-features and three algorithms in the training set (e.g., Figure 3.1 right-hand side table), one could build three binary classification

models for each pair of algorithms: Algorithm-1 vs. Algorithm-2, Algorithm-1 vs. Algorithm-3, and Algorithm-2 vs. Algorithm-3. The training data for the three binary classification models are the same, which is the $n \times m_f$ data matrix. Given a new dataset, we first calculate its meta-features, again an m_f-value array as a query. Then, we use the three binary classification models to classify the query. The final algorithm ranking list for the new dataset is computed based on how many times each algorithm has been predicted as "is better". If there were more than three algorithms to rank, then there might be ties in the list. Several tie breaking techniques have been examined in the literature [84, 21], but usually this is not a strong performance factor for meta-learning systems.

The advantage of the binary classification ranking approach is that existing binary classification algorithms can be employed directly. However, if there are many algorithms to rank, then the number of binary models required can be large and hard to manage in practice, e.g., for 20 algorithms, this would require $\frac{T\times(T-1)}{2} = \frac{20\times19}{2} = 190$ binary classification models to be built, which could be costly if one also considered using different (and fine tuned) classification algorithms for each of the 190 binary classification problems. The binary pairwise classification approach has also been studied in label ranking [79].

3.1.3 The Learning to Rank Approach

From the end-user's perspective, algorithm ranking is similar to the ranking problem in a search engine scenario, where a ranked list is returned for responding to a query. The search engine scenario has been extensively studied in learning to rank and information retrieval (IR). One obvious candidate for rank prediction is the AdaRank algorithm proposed in [167]. AdaRank is a boosting algorithm that minimizes a loss function directly defined on the ranking performance measure. The algorithm repeatedly constructs "weak rankers"

on the basis of re-weighted training data and finally linearly combines the weak rankers for making ranking predictions. In addition to supplying algorithms, the IR literature is also an excellent source for ranking evaluation metrics. Similar to search engine users, meta-learning users are usually mainly interested in the top candidates, be it websites or algorithms. An IR measure which captures this bias towards the top ranked items well is the normalized discounted cumulative gain (NDCG) metric [82], which has not been used in meta-learning evaluation before.

3.1.4 The Label Ranking Approach

Label ranking can be seen as an extension of the conventional setting of classification. The former can be obtained from the latter by replacing single class labels by complete label rankings [41, 40]. From the algorithmic view point, one type of label ranking approaches, such as the ranking by pairwise comparison (RPC) algorithm [79], is based on extending the pairwise binary classification approach by using more sophisticated ranking aggregation methods. Another type of label ranking algorithms, such the label ranking trees (LRT) algorithm [40], tries to apply probabilistic models under the label ranking setting. Meta-learning for algorithm ranking using the multi-target regression setting can also be transformed to a label ranking problem, so label ranking algorithms can be used directly.

3.1.5 The Multi-Target Regression Approach

As we can see in Figure 3.1, the algorithm ranking problem can also be seen as a multi-target (multi-response) regression problem, where the rank position values of each algorithm are the targets in the multi-target regression setting. In multi-target regression, the task is to estimate several target (response) variables using a common set of input variables (features). Two approaches

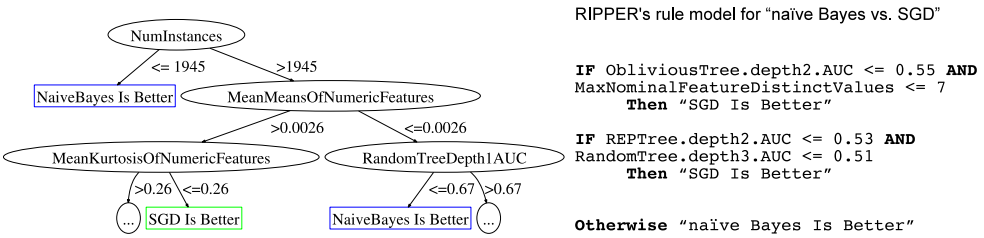
are usually employed. One is to build separate single-target regression models for each target variable; the other is to use a single model to estimate all the targets simultaneously. For meta-learning, the former approach is similar to the binary classification ranking approach but requiring fewer models to be built, e.g., for 20 algorithms, only 20 single-target regression models are needed. In this section, we focus on the latter approach and start from the linear model. A linear multi-target regression model can be expressed as:

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^* + \mathbf{E}, \tag{3.1}$$

where $\mathbf{W}^*$ is the regression coefficient matrix (an $n \times m_f$ matrix), $\mathbf{X}$ is the data (feature value) matrix, $\mathbf{Y}$ is the target matrix (can be rankings) and $\mathbf{E}$ is the noise term (matrix). The problem is usually formulated as a single convex optimisation problem of the form:

$$\underset{\mathbf{W}}{\text{minimize}} \quad f(\mathbf{W}) \text{ subject to } g(\mathbf{W}) \leq r, \tag{3.2}$$

where $f(\mathbf{W})$ is a loss function, e.g., squared loss, $g(\mathbf{W})$ is the regularization term, e.g., l^2-norm-based, and r is a constant. Although this kind of formulation is mathematically straightforward, efficient numerical techniques and relatively easy-to-implement algorithms are not readily available for sophisticated ranking-based loss/regularization functions. Also, algorithms in this category are usually designed for relatively low-dimensional problems or it is assumed that the number of training instances is much greater than the number of features (or the number of model parameters to estimate). In addition, we know that the various kinds of meta-features are "logically" related, so a nonlinear model might be more appropriate. Figure 3.2 shows two model output examples from a decision tree learner and a propositional rule learner for a binary-classification-based meta-learning problem, where only two algorithms are considered. In this example, three types of meta-features (SIL) are used. We can see that whether an algorithm is preferred depends on some logical relationships between meta-features. For example, the second rule in Figure 3.2



(a) A decision tree model (b) A rule model

Figure 3.2: Relationships between base-level meta-features.

(b) suggests that when there are no strong features (indicated by the "weak" tree-based landmarkers' AUCs being low) then SGD (WEKA implementation of an algorithm based on stochastic gradient descent) is likely to work better than Naive Bayes. For these reasons we leave the linear (single-model) multi-target model approach for future research.

Here, we focus on a special class of algorithms that are based on a nonlinear multi-target model:

$$\mathbf{Y} = z(\mathbf{X}), \tag{3.3}$$

here z is a decision-tree-based function. A well-known algorithm of this kind is called predictive clustering trees (PCT), which have been adapted for ranking problems in [150]. There are several advantages in using decision-tree-based rankers, including: a decision tree naturally identifies partitions of the data, where "similar" individuals are supposed to be in the same "cluster". This leads to a relatively natural interpretation for the meta-learning mechanism at the conceptual level; decision tree algorithms are reasonably fast (their training complexity is $\mathcal{O}(nlogn)$, where n is the number of training instances). In Section 3.3, we introduce a new ranker, which is an ensemble of decision-tree-based rankers.

3.2 Pairwise Meta-Rules

In this section, we introduce a novel meta-feature generation method in the context of meta-learning. The main motivation comes from our observation that existing meta-feature sets have ignored one potential source of information: the logical pairwise relationships between each pair of the target algorithms to rank. Explicitly adding this information to the meta-feature space might improve a meta-learner's predictive accuracy.

Of course, when predicting for a new dataset, this information will not be available, as otherwise we would not try to predict rankings in the first place. Therefore we propose to use a rule learner to learn pairwise rules first, and then use these rules as new meta-features. Two steps are involved: the first step is similar to the binary classification ranking approach, where $\frac{T\times(T-1)}{2}$ (T is the number of algorithms to rank) binary classification training datasets are constructed from the original $n \times m$ data matrix. Each binary dataset has two class labels: {"algorithm t_i is better", "algorithm t_j is better"}. Whether an algorithm is better than the other is determined by their ranking position in the ranked list of a particular dataset.

Taking the dataset illustrated in the right-hand side table of Figure 3.1 as an example, here we need to construct $\frac{3\times(3-1)}{2} = 3$ binary datasets as shown in Figure 3.3. Then, we build 3 rule-based binary classification models based on the 3 binary classification meta-datasets. In principle any rule learner could be used. We choose the RIPPER algorithm [42] here, because it is relatively fast and its rule models are generally very compact compared to other rule learners. Figure 3.2 (b) shows a RIPPER rule model. For each pair of algorithms, we compute a ruleset describing in which situation an algorithm is to be preferred over another. We call these rules the Pairwise Meta-Rules.

Next, we discuss how these rules can be used as new meta-features. Using Figure 3.2 (b) as an example, this RIPPER ruleset comprises three rules. The

t1 vs. t2 dataset	t1 vs. t3 dataset	t2 vs. t3 dataset
{100, 0.52, -1.0, "t2 is better"} {300, 0.45, 2.0, "t1 is better"} {450, 0.77, 1.5, "t2 is better"}	{100, 0.52, -1.0, "t1 is better"} {300, 0.45, 2.0, "t1 is better"} {450, 0.77, 1.5, "t3 is better"}	{100, 0.52, -1.0, "t2 is better"} {300, 0.45, 2.0, "t3 is better"} {450, 0.77, 1.5, "t3 is better"}

Figure 3.3: Example datasets for learning pairwise meta-rules.

first two are individual rules, whereas the third one is a default catch-all rule. We generate two different sets of meta-features from Pairwise Meta-Rules.

Method 1 turns each individual rule into one boolean meta-feature. For example, we may have a Pairwise Meta-Rule:

`If BaseMetaFeature-X` $\leq$ `0.5 AND BaseMetaFeature-Y` $\geq$ `0,`

`Then Algorithm A is better than Algorithm B`.

The value of the new meta-feature constructed from this Meta-Rule will be determined by looking at the (base-level) meta-feature values of a new dataset defined by a Meta-Rule. For a new dataset, the Meta-Rule-based meta-feature value is set to *true* if the rule condition "`BaseMetaFeature-X` $\leq$ `0.5 AND BaseMetaFeature-Y` $\geq$ `0`" is met, otherwise *false*. For the meta-learning problem, the RIPPER algorithm returns on average about 2 individual rules for each of the 190 algorithm pairs.

Method 2 creates just one boolean meta-feature representing the outcome of applying the full ruleset. For a new dataset, the Meta-Rule-based meta-feature value is set to *true* if the the ruleset conditions are all true, otherwise *false*. For the meta-learning problem, the RIPPER algorithm returns 190 rulesets for each of the 190 algorithm pairs, so in Method 2, 190 Meta-Rule-based meta-features are added to the original feature space.

The Pairwise Meta-Rules method is different from the standard stacking [164] method. We do not use the predicted complete ranking of base models, instead we use the RIPPER rule sets to construct new meta-features. A meta-learner will use all meta-features (including both base-level and Meta-Rule-based meta-features) for building the final model, which is also different from

stacking.

In the experiments reported below we compare three different meta-feature sets:

SIL-Only: 80 different SIL meta-features only;

SIL+Meta-Rules-1: the 80 SIL meta-features plus meta-features generated by Method 1;

SIL+Meta-Rules-2: the 80 SIL meta-features plus meta-features generated by Method 2.

Next, we propose a novel meta-learner (ranker), specifically designed for the meta-learning-based algorithm ranking problem.

3.3 Approximate Ranking Tree Forests for Meta-Learning

In preliminary experiments we found that the predictive performances of the predictive clustering trees for ranking (PCTR) [150] and the label ranking trees (LRT) [40] algorithms usually are worse than the optimised k-NN ranker for our meta-learning problem, where the number of objects to rank is relatively large (20 algorithms). Inspired by the bootstrap aggregating (bagging) [50**?**] strategy and the random forests framework [25], we here propose a new ranker for the meta-learning problem, called Approximate Ranking Tree Forests (ART Forests), which is a random forest ensemble of approximate ranking trees. The motivations for proposing ART Forests include:

1. as more and more meta-features are being developed and added to the feature space, we believe that a relatively fast algorithm (e.g., decision-tree-based learners) that has "built-in" feature selection capacity would be useful;

2. the meta-learner should not be constrained by the "number of training

examples must be much greater than the number of parameters to estimate" restriction, because for the meta-learning problem the number of datasets is usually not very large;

3. recent theoretical work [15] seems to suggest that irrelevant features do not significantly decrease random forest's predictive performance, so ideally the meta-learner should be capable of using as many meta-features as possible;

4. ensemble algorithms usually outperform base algorithms. Theoretically, statistics and models aggregated from bootstrap samples are nonparametric estimates [50], so we do not have to make parametric assumptions about the form of the underlying population, which provides an automatic approximation layer to our new algorithm. In the literature, an empirical study by [85] has shown that boosting-based meta-learner outperformed the k-NN based meta-learner in the pairwise binary classification setting (Section 3.1.2) for meta-learning.

3.3.1 Approximate Ranking Trees (ART)

In this section, we propose the ART algorithm which is used as the base learner for ART forests. The pseudocode is given in Algorithm 3.1. ART is a recursive partitioning algorithm that splits data into smaller sub-partitions which are increasingly and relatively more similar in terms of a homogeneous test (statistic) for examples in different partitions.

Next, we introduce the rationale underlying ART. We first review some basic concepts from permutation theory that will be used in this section, adopting the notation in [104] for analysing rank data in the multi-target setting. The basic unit of analysis consists of n datasets as judges to rank a set of m algorithms (objects). The set of objects can be denoted by $\mathcal{O} = \{O_1, O_2, ..., O_m\}$. For the meta-learning problem, we have a complete ranking of algorithms for

Algorithm 3.1 Approximate Ranking Trees (ART)

Input:

D (training data, a $n \times m$ data matrix);

u (number of features to use, default $log_2 M+1$, M is the number of features)

C (splitting and stopping criterions, details are given in Sec 3.3.2 and 3.3.4)

bestSplit $\leftarrow$ Randomly choose u features and test them based on the splitting criterion C. Use the best feature among the u features.

if *stopping criterion* (e.g., $n^{(D)} = 1$ or $R^{2(D_{bestSplit})} \geq \theta = 0.95$) is met **then**

Return a leaf node with the corresponding leaf ranking when $n^{(D)} = 1$ (or a ranking calculated from the average rank vector when $n^{(D)} > 1$).

else

// D^+ and D^- are two sub-partitions.

leftSubtree $\leftarrow$ **ART**($D^+_{bestSplit}$, u, C)

rightSubtree $\leftarrow$ **ART**($D^-_{bestSplit}$, u, C)

Return (*bestSplit*, *leftSubtree*, *rightSubtree*)

end if

each dataset: there is a best algorithm, second best,..., and finally a worst one, with ranks of $1, 2, ..., m$, respectively. We here use $\mathcal{S}_m$ to denote the permutations of m ranks:

$$\mathcal{S}_m \equiv \{\text{All permutations of the ranks } \{1, 2, ..., m\}\}, \tag{3.4}$$

so the (multi-) target part of our meta-dataset can be expressed as a sample of (complete) ranking vectors:

$$y^{<1>}, ..., y^{<n>} \in \mathcal{S}_m. \tag{3.5}$$

For example, the targets part of Figure 3.1 right-hand side table consists of three rankings: $y^{<1>} = [2, 1, 3]', y^{<2>} = [2, 3, 1]', y^{<3>} = [3, 2, 1]'$.

3.3.2 ART's Splitting Criterion

The splitting criterion is a measure that quantifies the quality of a given split point. In general, a local model is fitted into different partitions. In regression trees, the local model is usually the mean (target) value of the examples in a partition. This approach is fast and relatively stable, especially when regression trees are used as base models for a tree ensemble, since the mean is an optimal estimator for i.i.d. examples. The analogy in ranking trees is that we need to define or estimate a central ranking $\hat{z}$ to be used as the local model so that $\hat{z}$ minimizes a ranking-based dispersion measure, such as the average distance corresponding to a distance function $d(.,.)$ of ranking vectors:

$$\bar{d}(z) \equiv \frac{1}{n}\sum_{i=1}^{n} d(y^{<i>}, z), z \in \mathcal{S}_m. \tag{3.6}$$

When n is large, given a proper distance function, the sample central ranking could be estimated using maximum-likelihood estimation (MLE) or Bayesian methods for certain ranking models, such as the Mallows models [104], but these methods are usually computationally heavy, hence not suitable for ensembling. For ART, we use the average ranking calculated from the average rank vector as an approximation to the central ranking of rankings in a partition: $\hat{z} = averageRanking(\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^{<i>})$, in terms of the Spearman's distance:

$$d(y^*, y) = \sum_{i=1}^{m} |y_i^* - y_i|^2. \tag{3.7}$$

When d is the Spearman's distance (Eq. 3.7), based on Theorem 2.2 of [104], the average ranking $averageRanking(\bar{y})$ is the sample central rankings, which provides a theoretical foundation for ART.

Alternative distance functions could also be used but we will show later (Eq. 3.11) that the Spearman's distance has some convenient mathematical properties. To get a feeling about what an average ranking is in our meta-learning problem, Figure 3.5 shows an example of the average ranks of 20 algorithms over 466 datasets.

In the ART algorithm, we use the median value of a (meta-) feature domain as a (binary) split point to split the data D (the current partition) into two sub-partitions D^+ and D^-. The best split point is determined by the one that maximises the R^2 statistic:

$$R^2 = 1 - \frac{\sum_{l=1}^{L} \sum_{i=1}^{n^{(l)}} d(y^{(li)}, \hat{z}^{(l)})}{\sum_{l=1}^{L} \sum_{i=1}^{n^{(l)}} d(y^{(li)}, \hat{z}^{(D)})}, \tag{3.8}$$

where L is the number of populations (partitions), and $n^{(l)}$ is the number of examples in partition l. R^2 is originally designed to measure the proportion of the spread explained by the differences in the partitions (by partitioning) [104]. There are two special cases: if the sample central rankings of different partitions are equal, such as

$$\hat{z}^{(1)} = ... = \hat{z}^{(L)}, \tag{3.9}$$

then $R^2 = 0$ implies that partitioning is not necessary; if there is only one distinct ranking within each partition, and they are not all the same between partitions, then $R^2 = 1$. Here we estimate R^2 and employ it as a heuristic for ART induction. Next, we will derive an "one-step" formula for calculating R^2 without calculating the distance between the central ranking and the other rankings in a partition.

At inner nodes, ART tests two partitions D^+ and D^- so we have $L = 2$, and we can rewrite Eq. (3.8) to:

$$\begin{aligned} R^2 &= 1 - \frac{\sum_{i=1}^{n^{(D^+)}} d(y^{(D^+i)}, \hat{z}^{(D^+)}) + \sum_{i=1}^{n^{(D^-)}} d(y^{(D^-i)}, \hat{z}^{(D^-)})}{\sum_{i=1}^{n^{(D^+)}} d(y^{(D^+i)}, \hat{z}^{(D)}) + \sum_{i=1}^{n^{(D^-)}} d(y^{(D^-i)}, \hat{z}^{(D)})} \\ &= 1 - \frac{n^{(D^+)} \bar{d}_{Spearman}^{(D^+)} + n^{(D^-)} \bar{d}_{Spearman}^{(D^-)}}{n^{(D^+)} \bar{d}_{Spearman}^{(D)} + n^{(D^-)} \bar{d}_{Spearman}^{(D)}} \\ &= 1 - \frac{n^{(D^+)} \bar{d}_{Spearman}^{(D^+)} + n^{(D^-)} \bar{d}_{Spearman}^{(D^-)}}{n^{(D)} \bar{d}_{Spearman}^{(D)}}, \end{aligned} \tag{3.10}$$

where $D = D^+ \cup D^-$.

For rank data the average distance is also a proper measure of spread $\hat{\alpha}$: $\hat{\alpha}_{Spearman} \equiv \bar{d}_{Spearman}$ [104]. [3] and [104] present a convenient mathematical

result with detailed derivations for Spearman' distance (a brief derivation is given in the next section):

$$\hat{\alpha}_{Spearman} = \frac{m(m+1)(2m+1)}{3} - 2||\bar{y}||^2. \tag{3.11}$$

Here we replace $\bar{d}_{Spearman}$ with $\hat{\alpha}_{Spearman}$, and rewrite Eq. (3.10) to:

$$\begin{aligned} R^2 &= 1 - \frac{n^{(D^+)}\hat{\alpha}^{(D^+)}_{Spearman} + n^{(D^-)}\hat{\alpha}^{(D^-)}_{Spearman}}{n^{(D^+)}\hat{\alpha}^{(D)}_{Spearman} + n^{(D^-)}\hat{\alpha}^{(D)}_{Spearman}} \\ &= 1 - \frac{n^{(D^+)}\hat{\alpha}^{(D^+)}_{Spearman} + n^{(D^-)}\hat{\alpha}^{(D^-)}_{Spearman}}{n^{(D)}\hat{\alpha}^{(D)}_{Spearman}}. \end{aligned} \tag{3.12}$$

Combining everything together, we can compute R^2 (Eq. 3.8) efficiently:

$$R^2 = 1 - \frac{n^{(D^+)}(h - 2||\bar{y}^{(D^+)}||^2) + n^{(D^-)}(h - 2||\bar{y}^{(D^-)}||^2)}{n^{(D)}(h - 2||\bar{y}^{(D)}||^2)}, \tag{3.13}$$

where $h = \frac{m(m+1)(2m+1)}{3}$.

3.3.3 A Brief Derivation of Eq. 3.11

Let Q_m be the set of all $m \times m$ permutation matrices, where m is the number of objects (e.g., algorithms) to rank. The sets $\mathcal{S}_m$ (in Eq. 3.4) and Q_m are in one-to-one correspondence:

$$Q^{(y)}e_m = y, Q^{(y)} \in Q_m, y \in \mathcal{S}_m,$$

where $e'_m = [1, 2, 3, ..., m]$. For example, $Q_{ij} = 1$ iff $y_i = j$.

Define the Marginals matrix $\hat{M}$, an $m \times m$ matrix whose $(ij)^{th}$ element is:

$$\hat{M}_{ij} = \sum_{k=1}^{n} I[y_i^{(k)} = j],$$

where $I[A] = 1$ if event A occurs and 0 otherwise.

Let d be a Hoeffding distance of the form:

$$\begin{aligned} d(x, y) &= \textstyle\sum_{i=1}^{m} a(x_i, y_i) \\ &= tr(Q^{(x)}\Delta Q^{(y)'}), \end{aligned}$$

where $a(.,.)$ is a function on $\{1,...,m\} \times \{1,...,m\}$ that satisfies $a(i,i) = 0$ and $a(i,j) = a(j,i)$.

For Spearman distance (which is a form of Hoeffding distance):

$$\Delta = f_m 1'_m + 1_m f'_m - 2e_m e'_m,$$

where $f'_m = [1, 2^2, 3^2, ..., m^2]$ and $1'_m = [1, 1, 1, ..., 1]$.

The spread in Eq. 3.11 is calculated as:

$$\begin{aligned} \hat{\alpha}_{Spearman} &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d(y^{(i)}, y^{(j)})}{n^2} \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} tr(Q_i \Delta Q'_j)}{n^2} \\ &= tr(\hat{M}\Delta\hat{M}'), \end{aligned}$$

where $\hat{M}1_m = \hat{M}'1_m = 1_m$. Since it is well know that the sum of the squares of integers $1^2 + 2^2 + 3^2 + ... + m^2 = \frac{m(m+1)(2m+1)}{6}$, therefore

$$\begin{aligned} \hat{\alpha}_{Spearman} &= 2(1'_m f_m) - 2(\hat{M}e_m)'(\hat{M}e_m) \\ &= \frac{m(m+1)(2m+1)}{3} - 2||\bar{y}||^2. \end{aligned}$$

Above we give the core steps of the derivation. For more detailed about the Marginals matrix and intermediate-step derivations, we refer the reader to [3, 104].

3.3.4 ART's Stopping Criterion

In order to prevent overfitting, a stopping criterion is usually used to determine if it is worthwhile to split the current node. A natural stopping criterion can be introduced by adding a regularization parameter θ, e.g., ART will stop growing the current node if:

$$R^{2^{(bestSplit)}} \geq \theta, \theta \in (0, 1], \tag{3.14}$$

Another stopping criterion is the minimum number of examples at a leaf node, γ. Setting γ to a small number usually outperform trees using larger γ

Algorithm 3.2 Approximate Ranking Tree Forests (ART forests)

Input:

T (number of ART to use)

D (training data, a $n \times m$ data matrix);

u (number of features to use, default $log_2 M+1$, M is the total number of features)

C (splitting and stopping criterions, details are given in Sec 3.3.2 and 3.3.4)

$ART_{ensemble} \leftarrow \emptyset$

for $i = 1$ **to** T **do**

  $D_i \leftarrow$ *getBootstrapSample(D)*

  $ART_i \leftarrow$ **ART**(D_i, u, C)

  $ART_{ensemble} \leftarrow ART_{ensemble} \cup ART_i$

end for

Return $ART_{ensemble}$

values when the tree algorithm is used as the base learner for random forests [25]. ART uses both of these criterias to limit tree sizes.

Based on our experimental results (in the next section), we suggest using the following default parameters for ART:

- The minimum number of instances γ at a leaf node: 1.
- The "pruning" parameter θ: 0.95.

3.3.5 ART Forests and Rank Aggregation

We use the random forests framework described in [25] for ART forests induction. Algorithm 3.2 shows the pseudocode. An ART forest is grown as follows:

1. The training set is a bootstrap sample from the original training set;

2. An integer u is set by the user. At each node, u features are selected at random and the node is split on the best feature among the selected u;

In prediction, as a test feature vector $\mathbf{x}$ is put down each tree it is assigned the average ranking vector of the rankings at the node it stops at. The average ranking vector of these over all approximate ranking trees in the forest is the predicted ranking for $\mathbf{x}$. Alternative rank aggregation methods, such as other types of Borda count [125, 93], graph theory models and binary linear program [93] might perform better (with extra computational cost), and will be investigated in future research.

3.4 Experiment Setup and Results

This experimental comparison investigates two related questions.

- The first question concerns the new meta-feature generation method: can it consistently improve the performance of known meta-learners?

- The second question concerns the new algorithm: are ART forests competitive with current state-of-the-art rankers?

In this section, we focus on the performance of meta-learning for algorithm ranking on binary classification datasets only. Previous studies were limited by the small number of available datasets, usually fewer than 100 datasets were used in reported meta-learning experiments. To be able to draw statistically significant conclusions, we chose to use as many datasets as possible from various public data sources, including the UCI[2], StatLib[3], KDD[4] and WEKA[5] repositories. In total slightly more than 1,000 classification and regression

[2]http://archive.ics.uci.edu/ml/

[3]http://lib.stat.cmu.edu/

[4]http://kdd.ics.uci.edu/

[5]http://www.cs.waikato.ac.nz/ml/weka/

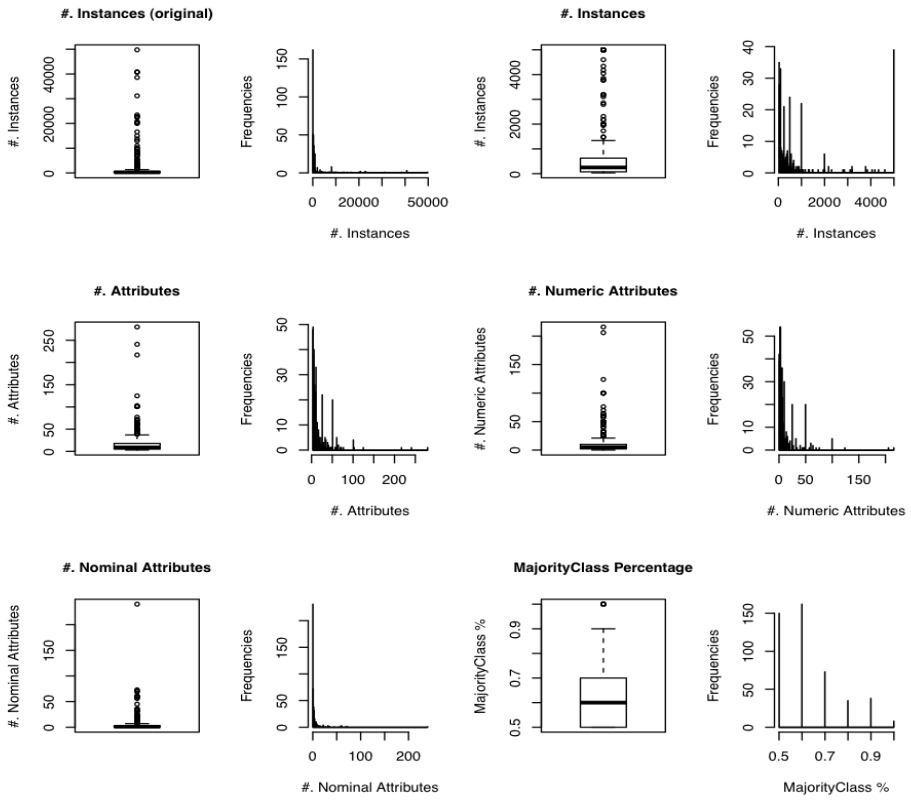


Figure 3.4: Some aggregated properties of the 466 datasets.

datasets were collected. However, due to the varying quality of the public datasets, not all of them could be used directly. After removing duplications and very small datasets (less than 10 instances), and converting multiple-class classification (by keeping the top two majority classes) and regression (by using the mean as a binary splitting point to transfer the numeric target to a binary target) datasets to binary classification datasets, we finally obtained 466 datasets for experiments. Also, to speed up our experiments we reduced relatively large datasets to a subset of 5,000 instances using random sampling. Figure 3.4 shows some of the aggregated properties of the 466 datasets.

3.4.1 EA-Based Performance Estimation

For simplicity and in order to speed up experiments, many previous meta-learning experiments have estimated algorithm performances using some pre-

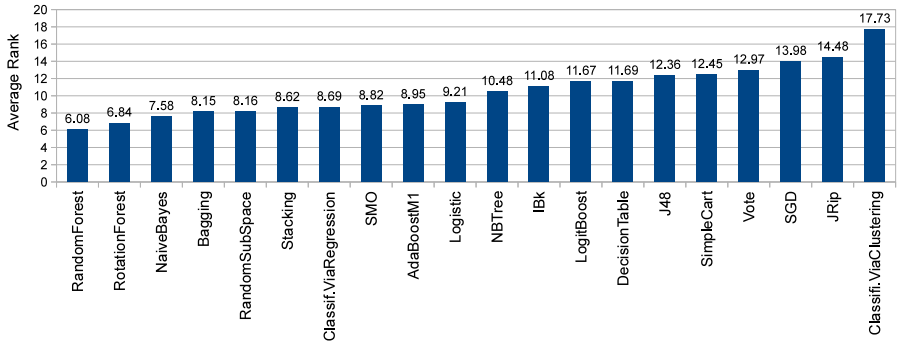


Figure 3.5: Sorted average ranks of the 20 algorithms over 466 datasets based on PSO-optimised performances. This list can be transformed to a ranking, e.g., (optimised) RandomForest is ranked 1, RotationForest 2,..., ClassificationViaClustering 20. The ranking can be seen as a default ranker over the 466 datasets.

specified default parameter settings across all base-level datasets. We claim that this approach is bound to be suboptimal in practice because to be able to make useful predictions, most algorithms have to be optimised for each specific dataset. Technically, predicting the full combination of algorithm plus optimal parameter settings is not feasible.

We therefore propose an intermediate approach, where we assume that a procedure is available for optimising each algorithm for each dataset, and then predict the ranking of the optimised algorithms. Given a new dataset, the recommended ranking is based on optimal parameter settings, and that the actual parameter setting of the algorithm(s) (selected by a meta-learning end-user) is optimised afterwards.

Algorithm	Parameters for PSO to tune
AdaBoostM1	I {10, 30, 100} H {1.0, 0.8, 0.6, 0.4, 0.2} Q {True, False} REPTree depth {-1 to 6}
Bagging	I {10, 30, 100} REPTree depth {-1 to 6} REPTree prune {True, False}
ClassificationViaClustering	numClusters {2 to 30}
ClassificationViaRegression	M5P (with different parameters) REPTree (with different parameters) LinearRegression (with different parameters)
DecisionTable	Evaluation measure {Acc,RMSE,MAE,AUC} numFolds {1 to 5}
IBk	K {1 to numInstances - 1}
J48	BinarySplits {True, False} ConfFactor {0.00001 to 0.5} minInstances {2 to 0.5 * numInstances}
JRip	Folds {3 to 10} minNo {2 to 10} pruning {True, False}
Logistic	R {0.0000001 to 100000.0} M {-1, 300, 500, 1000}
LogitBoost	REPTree depth {-1 to 6} H {0.2 to 1.0} I {10, 30, 100}
NBTree	D {True, False}
NaiveBayes	K {True, False}
RandomForest	I {10, 30, 100} K {0 to numAtts}
RandomSubSpace	I {10, 30, 100} P {0.1 to 0.9} REPTree depth {-1 to 6}
RotationForest	I {10, 30, 100} P {50 t0 90} RandomProjection {Gaussian, sparse1, sparse2}
SGD	F {log loss, hinge loss} N {True, False} M {True, False} R {0.00001 to 100.0} E {100, 500, 1000}
SMO	C {0.00001 to 10000.0} N {0,1,2} kernel {PolyKernel (E 1.0, 2.0), RBF(G 0.0001 to 1000.0)}
SimpleCart	H {True, False} M {2 to 0.3 x numInstances} U {True, False}
Stacking	Meta-classifier: LinearRegression X {2 to 5} 10 base classifiers: NaiveBayes Logistic SMO SGD IBk J48 NBTree SimpleCart DecisionTable JRip
Vote	R {AVG,PROD,MAJ,MIN,MAX,MED} 10 base classifiers: NaiveBayes Logistic SMO SGD IBk J48 NBTree SimpleCart DecisionTable JRip

Figure 3.6: WEKA algorithms and their parameter settings that are considered in the PSO-based parameter optimization procedure.

The parameter optimization procedure used here is based on evolutionary algorithms. Grid-search would be an alternative, but based on recent research [124], EA-based techniques seem more efficient. Specifically, we employ the particle swarm optimisation (PSO) based parameter selection technique described in [51, 147, 148]. Although EA-based performance estimation is more appropriate, it is time-consuming, especially for large datasets. Therefore, as mentioned above, large datasets were downsampled to 5,000 instances. In

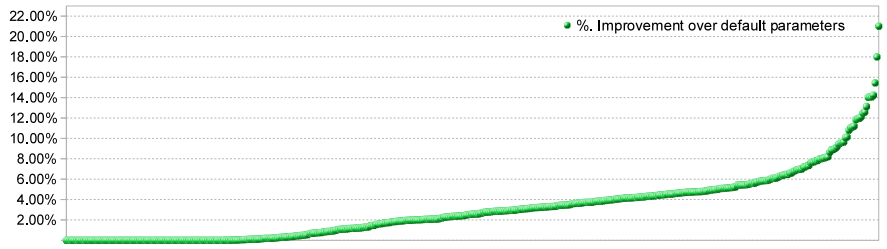


Figure 3.7: Percentage of improvement of the best AUC performance among 20 PSO-optimised algorithms for each dataset over the same 20 algorithm using their default parameters. Sorted by percentage of improvement over 466 datasets.

this experiment, meta-learning is used to rank 20 supervised machine learning algorithms (WEKA [74] implementations are used). When generating the meta-dataset from the 466 datasets, for each of the 20 algorithms, we manually choose different parameters and value ranges for PSO to optimise. Taking the support vector machine algorithm (SMO in WEKA) as an example, we set PSO to optimise the kernel type, all kernel parameters, and the complexity constant. Figure 3.6 shows the details of the WEKA algorithms and their parameter settings that are considered in the PSO-based parameter optimization procedure.

The class distributions of the 466 datasets vary a lot, with some of the datasets being very skewed, which can cause high variance on zero-one loss estimation. Consequently, we choose the area under the receiver operating characteristic curve (AUC) metric as the main performance measure, as it is less affected by class skew. We run the 20 algorithms (with PSO-based parameter optimisation) on the 466 binary classification datasets and use 10-fold cross-validation based AUC scores for ranking generation. Building up this meta-dataset is the most expensive part of our meta-learning experiment: it took roughly three weeks (about 6,000 single-core CPU hours) to complete on two 2.8GHz 16G RAM PCs, with 6 threads running on each machine.

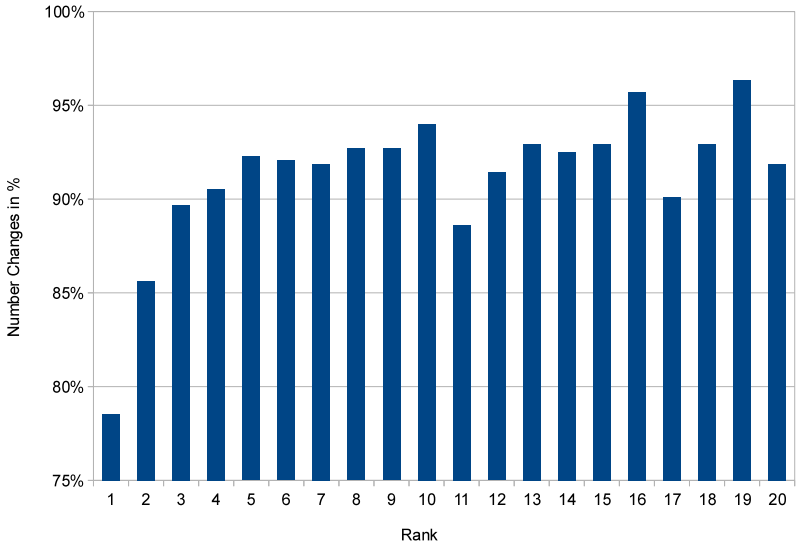


Figure 3.8: Number of datasets (in % of 466) did the default parameter setting cause one algorithm to be ranked at the respective rank value on the x-axis, whereas the PSO-version caused a different algorithm to be ranked at the respective rank.

Figure 3.7 shows the percentage of improvement of the best AUC score among the 20 PSO-optimised algorithms for each dataset over the best AUC score of the same 20 algorithms using their default parameters. The result demonstrates the benefit of using the performances of optimised algorithms for generating algorithm rankings. We claim this is a more appropriate experimental setting for meta-learning than simply using a fixed set of parameters for every dataset.

Figure 3.8 shows the number of datasets (in % with respect to 466) for which the default parameter setting cause one algorithm to be ranked at the respective rank value on the x-axis, whereas the PSO-version caused a different algorithm to be ranked at the same rank. For instance, the bar at rank 1 means for 78% of the 466 datasets, the default parameter setting cause one algorithm to win over all 20 algorithms, whereas the PSO-optimised parameter setting caused a different algorithm to win. In other words, for $100\% - 78\% = 22\%$

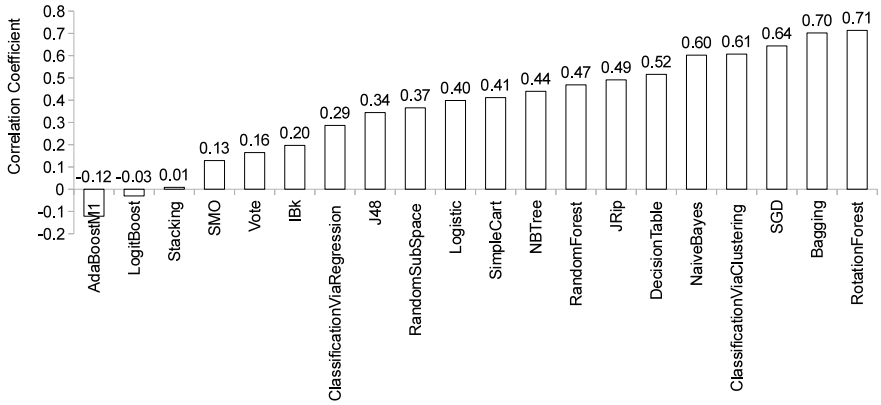


Figure 3.9: Correlation coefficient for each pair of "default parameter setting vs. PSO-optimised parameter setting" over the 466 datasets.

of the 466 datasets, the default parameter setting and the PSO-optimised parameter setting are matched.

Figure 3.9 shows the correlation coefficient for each pair of "default parameter setting vs. PSO-optimised parameter setting" ranking over 466 datasets. Higher correlation coefficient values mean that the default and the PSO-optimised parameter settings of the respective algorithm are more consistent in terms of correlation coefficient. This figure also reveals which algorithm is more sensitive to parameter optimisation. Algorithms with relatively lower scores means the model produced from their default parameter setting cannot represent the full power of those algorithms, such as AdaBoostM1, LogitBoost, Stacking and SMO. For those algorithms, parameter optimisation should be done for each dataset. Algorithms with relatively high scores mean that default parameter settings usually yield good models. This includes algorithms such as RotationForest, Bagging and SGD.

3.4.2 Meta-Learners (Rankers) in Comparison

We test the new pairwise meta-rule based meta-features with different meta-learners. In total 7 rankers are used for our experiments.

DefaultRanker—The default ranker follows a very simple yet powerful philosophy: if an algorithm has worked well on previous datasets, then it is likely to be used as the first algorithm to try on a new dataset. So, the default ranker used in our experiments is simply using the average rank of each algorithm over all the training data (meta-dataset). Thus the ranking predicted by the default ranker is always the same for every test dataset, which is the average ranking of the training examples. This ranker can also be seen as a special case of an ART model with only one node. One distinguishing feature of the algorithm ranking problem is that the (average-ranking-based) default ranker is relatively strong compared with common preference learning problems, such as movie/book recommendation and survey data involving human subjects.

***k*-NN**—The k-NN ranker, as described in Section 3.1, uses standard Euclidean distance and we set $k = 15$ (we will show later that 15 is a relatively good value for our problem).

LRT—The label ranking trees ranker, we use the WEKA-LR[6] implementation with default parameters. LRT is based on the Mallows model for rank data.

RPC—The RPC ranker, is the ranking by pairwise comparison algorithm proposed in [79]. We use the default setting of the RPC implementation in WEKA-LR in which the logistic regression algorithm is used as the base learner.

PCTR—The PCTR ranker, is the predictive clustering trees for ranking (PCTR) ranker [150]. The minimum number of instances at a leaf node is set

[6]http://www.uni-marburg.de/fb12/kebi/research/software/weka-lr-page/

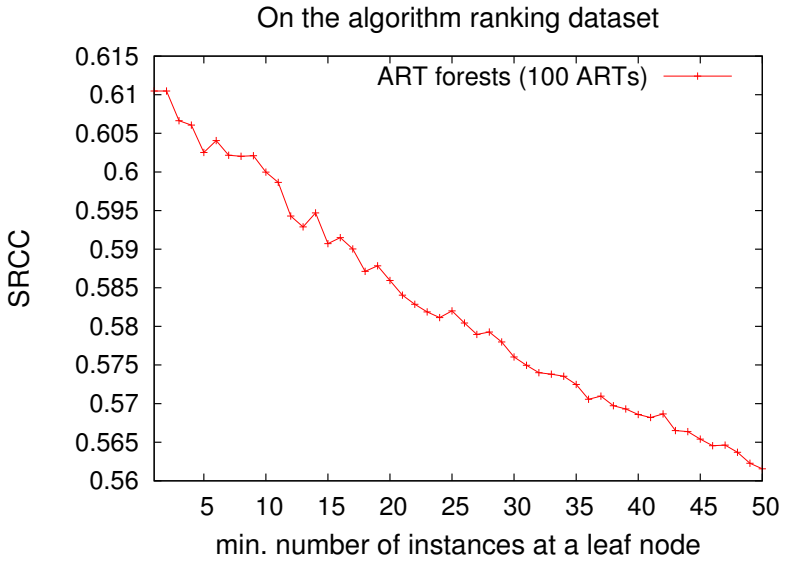


Figure 3.10: The min. number of instances at a leaf node vs. ART forests's performance

to 15.

AdaRank—The AdaRank ranker uses 100 PCTR rankers as its base models and the minimum number of instances at a leaf node is set to 30 (to simulate a relatively weak learner as in [167]). Please note that boosting algorithms can stop early so the final AdaRank ensemble might not have exactly 100 base models.

ARTForests—The ARTForests ranker, we use 100 approximate ranking trees for ART forests, and the minimum number of instances γ at a leaf node is set to 1 for each ART (using small γ such as 1 leads to a good bagging ensemble, which is suggested in [25] and is also confirmed by our preliminary experiment (please see Figure 3.10); The "pruning" parameter θ is set to 0.95. The number of random features to use is set to $log_2 M+1$, where M is the total number of features

3.4.3 Evaluation of Ranking Accuracy

We assess ranking accuracy by comparing the rankings predicted by a ranker for a given dataset with the corresponding target rankings. Given two sets of m-value rankings: $T = [T_1, T_2, ..., T_{m-1}, T_m]$ and $P = [P_1, P_2, ..., P_{m-1}, P_m]$, which are targets and predictions, respectively, and letting $d_i^2 = (T_i - P_i)^2$, the following ranking evaluation metrics and functions are used in our experiments.

Spearman's Rank Correlation Coefficient (SRCC)—SRCC is defined as:

$$\rho_{SRCC} = 1 - \frac{6\sum_{i=1}^{m} d_i^2}{m(m^2-1)}, \tag{3.15}$$

which assesses how well the relationship between the true and predicted rankings can be described using a monotonic function [86].

Weighted Rank Correlation—WRC is defined as:

$$\rho_{WRC} = 1 - \frac{6\sum_{i=1}^{m} d_i^2((m - T_i + 1) + (m - P_i + 1))}{m^4 + m^3 - m^2 - m}. \tag{3.16}$$

The WRC metric puts more weight on the top candidates. It has been used for meta-learning in [140, 120].

Loose Accuracy—The loose accuracy (LA@X) measurement considers ranking accuracy of the top X candidates only. LA@1 is also called the restricted accuracy metric (count 1.0 if the top one is correct, otherwise count 0); Similarly, LA@3 means we count 1.0 if one of the predicted top three candidates matched the true top one, otherwise count 0. LA has been used for meta-learning in [84]. We report results for LA@1, 3 and 5.

Normalized Discounted Cumulative Gain—Discounted cumulative gain (DCG) is a measure of effectiveness of a search engine algorithm or related applications by using a graded relevance scale of items in a result list. The gain is accumulated from the top of the list to the bottom with the gain of each result discounted at lower ranks [82]. The DCG@X is defined as: $\text{DCG@X} = \sum_{i=1}^{X} \frac{2^{g_i}-1}{\log_2(i+1)}$, where g_i is a grade value. The Normalized DCG at

position X is defined as:

$$\text{NDCG@X} = \frac{DCG@X}{IDCG@X}, \tag{3.17}$$

here IDCG is the ideal DCG at X. We report results for NDCG@1, 3 and 5.

The actual evaluation scores for the above ranking evaluation functions of each ranker were estimated based on multiple runs of training/testing split evaluations. We use the average scores obtained from 10 runs of 90% vs. 10% training/testing evaluation for result visualization. For each run, 419 (90% of 466) datasets were randomly selected for building a meta-learning system using the corresponding ranker, and the remaining 47 datasets were used for testing.

To avoid information leaking, we make sure that when meta-rule based meta-features are used by a ranker, the rules are generated using only the corresponding training meta-dataset of each run.

3.4.4 Experimental Results

We present and analyse two sets of experimental results. One is a comparison of meta-feature sets based on k-NN performance curves; the other one is a comparison of ranking performances of multiple rankers on two meta-feature sets.

Figure 3.11 shows the performances of the k-NN ranker using three meta-feature sets under different numbers of nearest neighbors (k). The performance of the default ranker is also included. Overall, k values between 10 and 20 usually produce relatively good performance across all eight ranking metrics. Regarding the choice of meta-feature sets, we can see that k-NN using the SIL+Meta-rules-1 set outperforms the SIL-only and the SIL+Meta-rules-2 meta-feature sets when appropriate k values are chosen. Taking the LA@1 metric as an example, the best k-NN performance is about 0.26, which means that on average, if an optimal k value is used, the probability for the k-NN

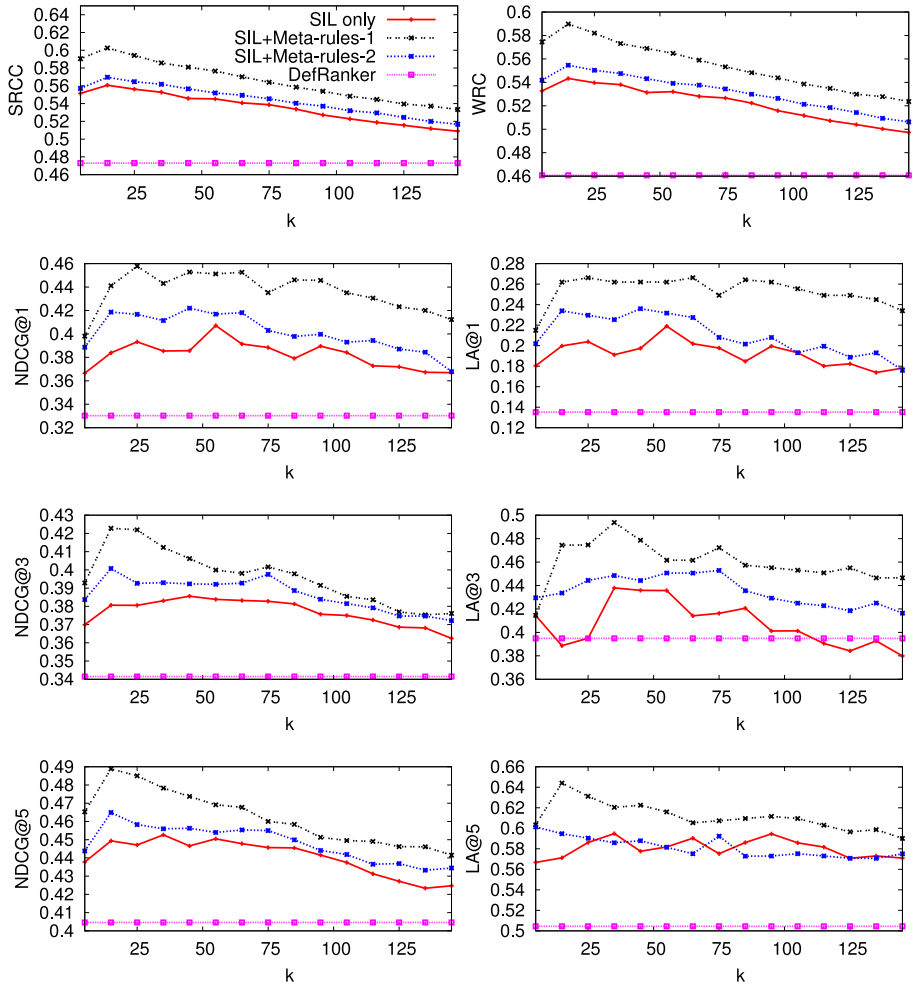


Figure 3.11: Comparison of meta-feature sets by k-NN curves. Higher scores are better.

ranker to predict correctly the best algorithm to use from 20 algorithms is 26%, where the default ranker achieves about 14%, and random guessing would only result in $1/20 = 5\%$. The LA@3 and LA@5 metrics behave similarly. Regarding the other five metrics, they all usually outperform the baseline, albeit to varying degrees, always clearly depending on a proper choice of the value of k.

With the second set of results, we compare different rankers on two sets of meta-features: the SIL-only set and the SIL+Meta-rules-1 set. For simplicity, we use SIL+Meta-rules to refer the SIL+Meta-rules-1 set. Figure 3.12 shows the performances of the seven rankers (with and without using the meta-rules) on the eight metrics. A "*" besides a ranker's name means the SIL+Meta-

rules set is significantly better than the SIL-only set (using a paired t-test significance level of 0.05). In total, the SIL+Meta-rules set significantly wins 38 out of 48 (about 79.1%) comparison tests across all rankers (please note that the default ranker is not counted since it does not use any meta-features). Overall, we can see that the ART forests ranker with the SIL+Meta-rules set consistently produces performance gains for all different metrics, and is placed as the best ranker for 7 out of 8 metrics. Although increasing ART Forests' ensemble size may further improve its performance, using 100 as a default setting seems to work well for our experiments.

Table 3.1 shows a summary of the top one ranker for each metric and how much performance gain the respective ranker can achieve. We can see that all the best rankers used the SIL+Meta-rules set. The greatest performance gain (93%) over the default ranker is from the k-NN ranker on the LA@1 measurement. The "Gain §" values show how much improvement the top rankers using the SIL+Meta-rules meta-feature set can achieve over the default ranker; and the "Gain †" values show how much improvement the Meta-Rule method (Method 1) introduced in this chapter can achieve over the respective best rankers using the SIL-only meta-feature set.

In terms of runtime, the RIPPER-based pairwise meta-rules method on average takes 30 seconds to generate rulesets for 190 algorithm pairs.

Figure 3.13 shows the runtime of different rankers (exclude the runtime for learning pairwise meta-rules). We can see that the ART Forests algorithm (with 100 ART models) is relatively efficient compared with other state-of-the-art rankers. When rapid modeling is required for an application, the k-NN ranker is a reasonable alternative because it is fast and relatively accurate.

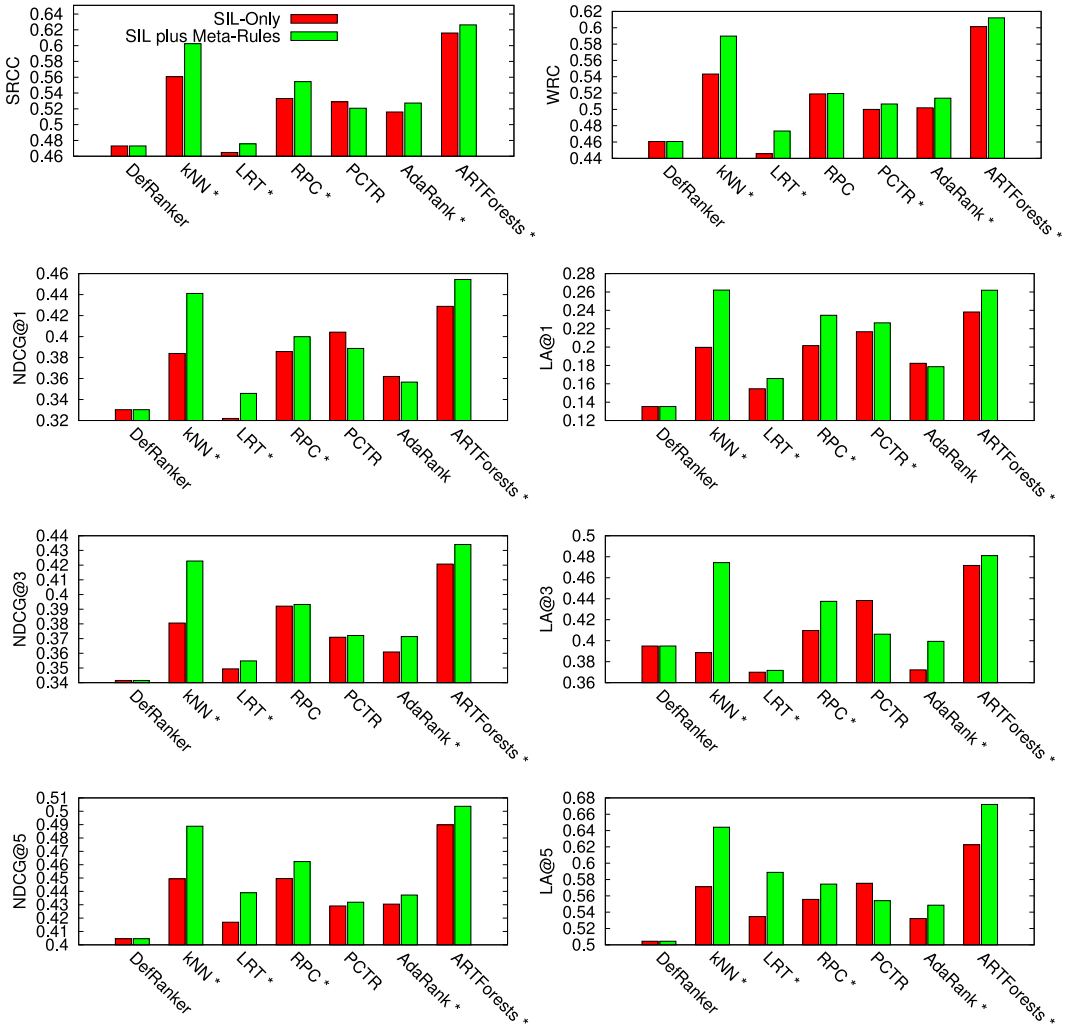


Figure 3.12: Comparison of meta-feature sets by ranker performances.

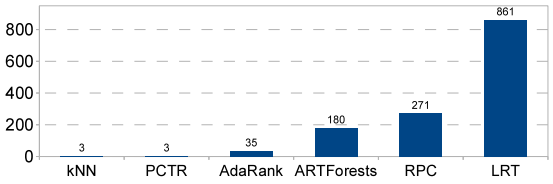


Figure 3.13: Single-core runtime in seconds including both training and prediction stages.

3.5 Discussions

In this section, we discuss the limitations of the proposed techniques and future work for extending and improving the system.

Evaluation	Best Ranker	Used ‡	Gain §	Gain †
SRCC	ART Forests	Yes	32.41%	1.67%
WRC	ART Forests	Yes	32.83%	1.77%
LA@1	k-NN	Yes	93.82%	31.1%
LA@3	ART Forests	Yes	21.85%	2.00%
LA@5	ART Forests	Yes	33.23%	7.94%
NDCG@1	ART Forests	Yes	37.60%	5.97%
NDCG@3	ART Forests	Yes	27.06%	3.16%
NDCG@5	ART Forests	Yes	24.50%	2.84%

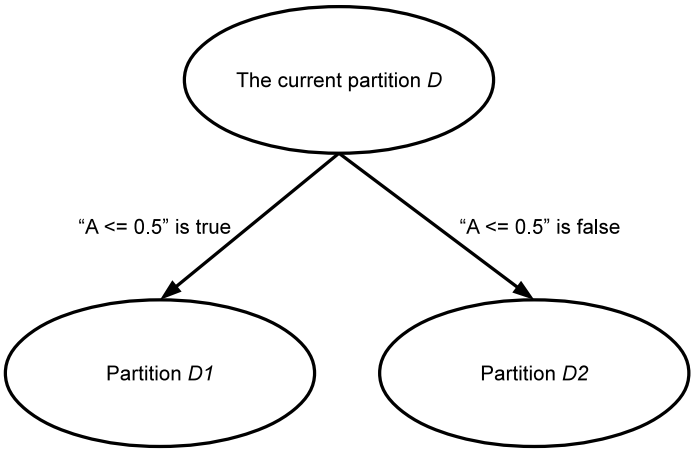
Table 3.1: Summary of the top one ranker performances under different ranking evaluation metrics and functions. **Used** ‡: meta-rules (SIL+Meta-Rule set) is used, or not. **Gain** §: performance gain of the best ranker over the default ranker. **Gain** †: performance gain of the best ranker using the SIL+Meta-Rule set over the same ranker using the SIL-only set.

3.5.1 Limitations

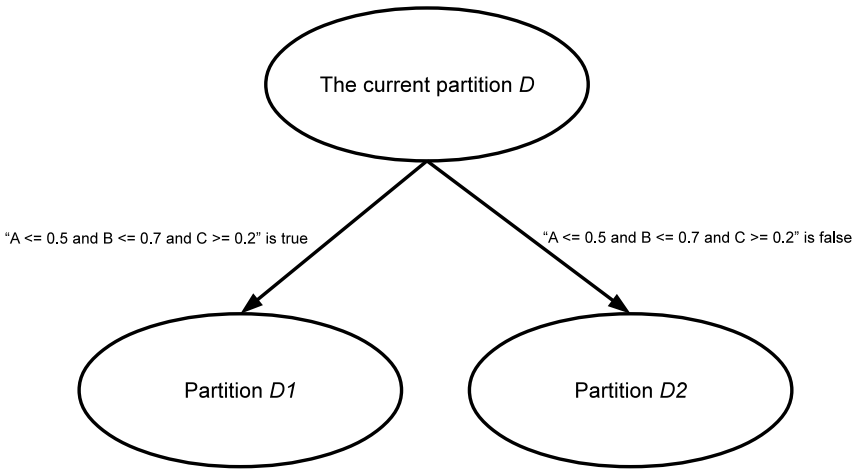
For the pairwise meta-rule method, although the RIPPER algorithm works well, we believe that alternative rule learners are still worth investigating. An important direction for future work is to design efficient rule learners for generating pairwise meta-rules. Because the number of pairwise rule models required increases quadratically with the number of objects (algorithms) to rank. Therefore, the training complexity of the chosen rule learner is critical to the scalability of the proposed method. Let z be the average number of rules returned by the rule learner for an algorithm pair, the total number of pairwise meta-rule based meta-features generated for m algorithms is $\frac{zm(m-1)}{2}$. So we can see that when using the pairwise meta-rule method, the number of meta-features also grows quadratically with the number of algorithms to rank. Although this is the reason for proposing the ART forests algorithm,

the performance and benefit of applying the pairwise meta-rule method on the other meta-learners would be affected when the number of algorithms to rank increases. To overcome this problem, in future work, we will investigate: (1) rule pruning techniques for reducing the total number of pairwise meta-rules; (2) rule set fusion techniques, such as group-wise meta-rules. Another interesting future direction would be using the predicted rank differences (or posterior probabilities) instead of the purely boolean pairwise meta-features, as the actual differences could be more informative. For the ART algorithm, in this chapter we only considered and evaluated using the median value of a meta-feature domain as a *binary* split. A future work direction would be comparing this to multi-point splitting strategies.

In this chapter, we employed the EA-based parameter selection technique, which is a relatively expensive approach for generating meta-dataset. In the literature, there are some attempts to use meta-learning itself to parameter selection. [142] and [2] have shown promising results using meta-learning for selecting the kernel type and kernel parameters for support vector machines. There is also previous work that proposed hybrid systems that combine meta-learning and optimisation techniques [124, 71, 43]. These systems could also be used as alternative parameter selection procedures for the meta-learning experiment setup proposed in the current research. In Section 3.4.1, we mentioned that predicting the full combination of algorithm plus optimal parameter settings is not feasible. While this is technically true, [97, 98] has introduced the active testing based meta-learning framework that is able to return good parameter setting together with the recommended algorithm. In their experiments, a set of 292 algorithm-parameter combinations was evaluated.



(a) ART-0's 1-condition splitting point



(b) ART-1's multiple-condition splitting point

Figure 3.14: 1-condition splitting vs. multiple-condition splitting

3.5.2 Single-Condition vs. Multiple-Condition Splitting

In the original ART algorithm, the median value of a meta-feature domain is used as the splitting point for testing. For example, if the median value of meta-feature A is 0.5, the splitting point (rule) used for splitting the current partition is then "`A` $\leq$ `0.5`". Figure 3.14 (a) shows an example. For simplicity, in this section, we call this algorithm **ART-1c**. We can see that ART-1c uses only one condition for splitting.

In the following experiment, ART-1c is compared to a variant of ART which uses multiple randomly generated conditions as the splitting point. Figure 3.14

(b) shows an example. For simplicity, we call this variant—**ART-mc**. Next, we describe how to generate a rule-like splitting point with multiple random conditions.

Algorithm 3.3 Generating a multiple-condition rule-like splitting point

$R \leftarrow \emptyset$

Repeat M times

Randomly choose a feature A from an "active set" of features

Randomly choose a value from the feature domain (between $min(A)$ and $max(A)$)

Randomly select an operator from $\{\leq, \geq\}$

Now, we have a condition c generated, e.g., $A \leq 0.5$. Add c to R

return R

Let M be the number of conditions to be used in a rule-like splitting point. Algorithm 3.3 shows the pseudocode for generating a random multiple-condition rule-like splitting point.

ART-mc works as follows. At an inner node, ART-mc generates K random multiple-condition rule-like splitting points, and then R^2 is calculated for each splitting point. The one with the highest R^2 score is used as the final splitting point to split the current partition into two sub-partitions. Next, we show some experimental results.

Figure 3.15 shows a comparison between single-condition splitting (ART-1c) and multiple-condition splitting (ART-mc) for a single tree on the meta-learning dataset we have used in the previous sections. The y-axis values on the 8 figures are CV-based performance measures of a single ART. Labels in the x-axis indicate the splitting method that is used by the respective ART variant. For example, ART-mc (1) means the respective ART variant uses 1 random condition for splitting; ART-mc (2) means the respective ART variant uses 2 random conditions for splitting. Similarly, ART-mc (6) means the

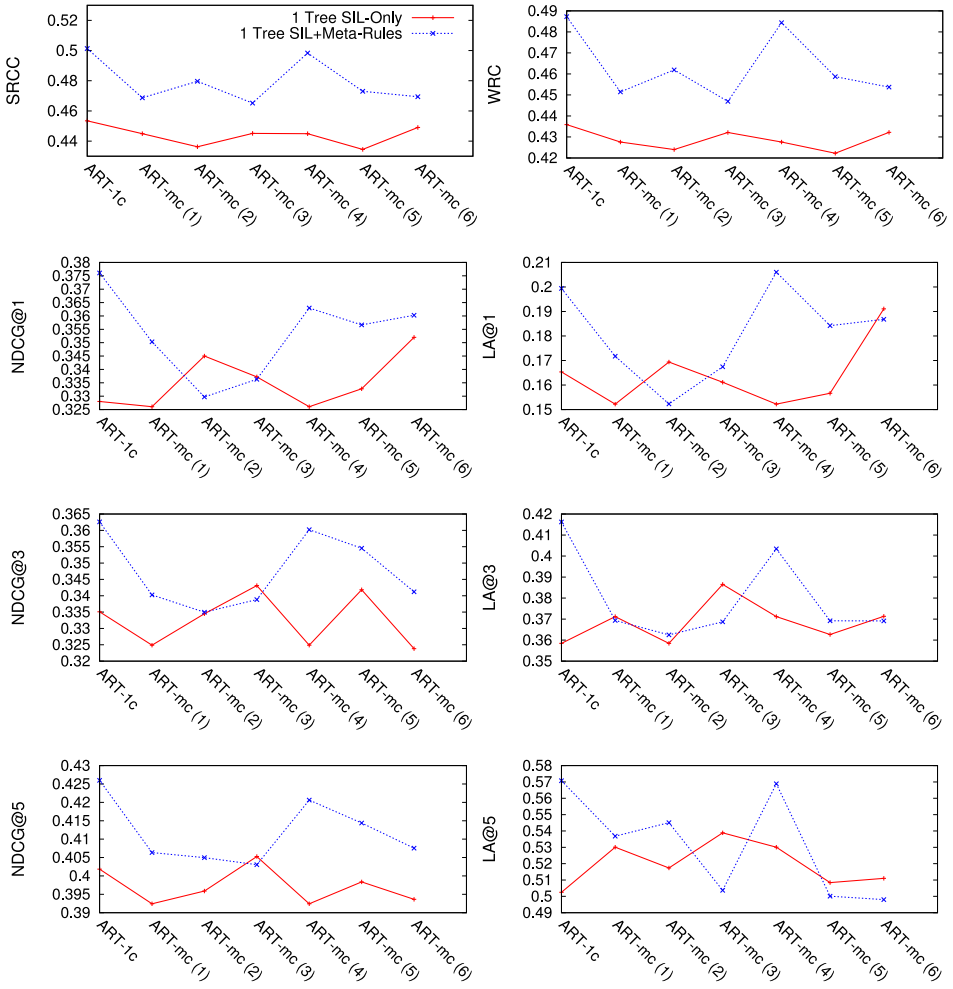


Figure 3.15: 1-Tree; SIL-only vs. SIL+MetaRules. Higher scores are better. Labels in the x-axis indicate the splitting method that is used by the respective ART variant. For example, ART-mc (1) means the respective ART variant uses 1 random condition for splitting

respective ART variant uses 6 random conditions for splitting. Overall, we can see that the meta-rule-based ART variants are more likely to achieve a higher predictive performance compared to ART variants that did not use meta-rules. When comparing single vs. multiple conditions for splitting, we see that increasing the number of random conditions does not always improve the predictive performance.

Figure 3.16 shows a similar comparison between single-condition splitting and multiple-condition splitting for ART forests (100 ARTs). Overall, figures

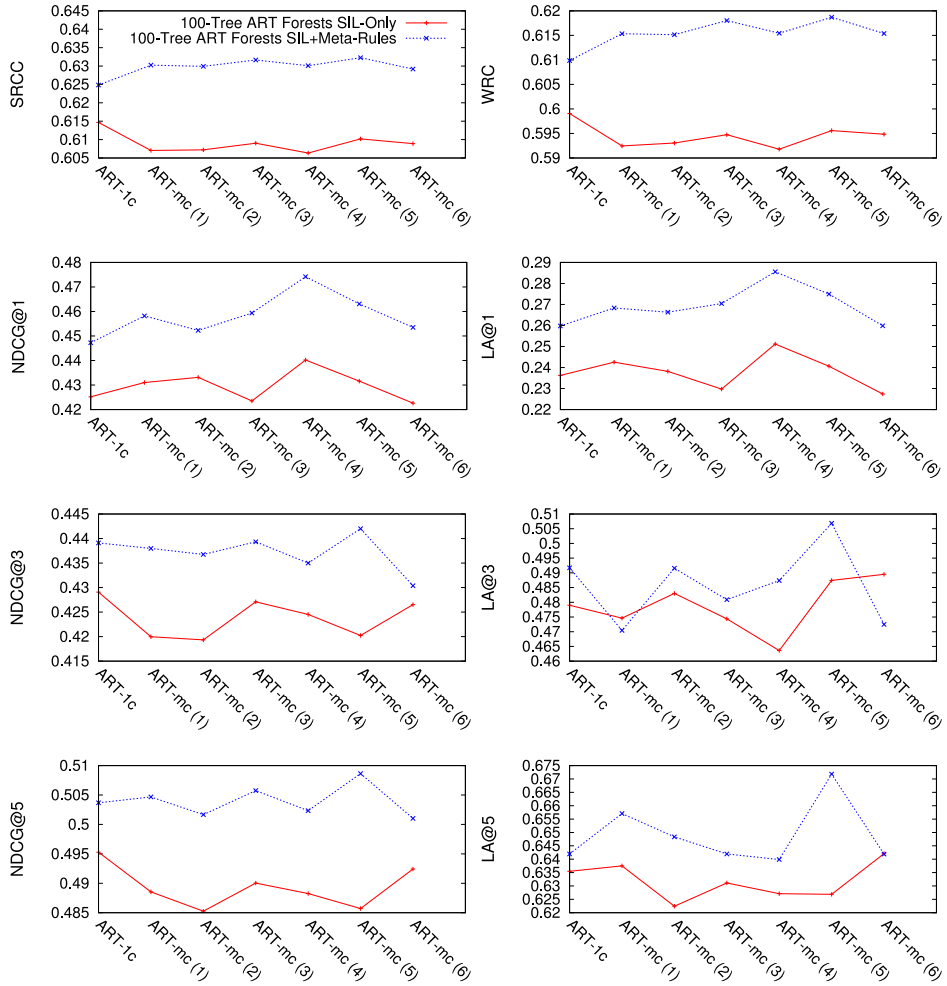


Figure 3.16: 100-tree ART Forests; SIL-only vs. SIL+MetaRules. Higher scores are better. Labels in the x-axis indicate the splitting method that is used by the respective ART variant. For example, ART-mc (1) means the respective ART variant uses 1 random condition for splitting

in Figure 3.16 show that the meta-rule-based ART forests are more likely to achieve a higher predictive performance compared to ART forests that did not use meta-rules. When comparing using single or multiple conditions for splitting, the pattern is similar to Figure 3.15. Again, we can see that increasing the number of random conditions does not always improve the predictive performance.

The above results show again that the meta-rule based algorithms outperform algorithms only use the original features (SIL-only). The results also

indicate that using multiple random conditions for splitting do not necessarily improve the predictive performance over using a median-value-based single rule condition for splitting.

3.6 Preliminary Theoretical Analysis

In this section, we attempt to give further explanation why ART forests works well in practice.

We first review some basic concepts that will be used, adopting the notations used in [52]. In our algorithm ranking problem we have n datasets, and a set of m machine learning algorithms. A performance estimation methodology is used to estimate the predictive performance of each algorithm on each dataset in terms of a metric (e.g., AUC). Given this information, we get a full ranking π (of m algorithms) for each dataset:

$$\pi = (\pi(1), ..., \pi(m)),$$

where $\pi(i)$ is the rank given to algorithm $i, i = 1, ..., m$. Denote the set of all $m!$ rankings by S_m, and a right-invariant discrepancy function $d(\cdot, \cdot)$ on $S_m \times S_m$. Here "right-invariant" means: $d(\mu, \pi) \geq 0$, $d(\mu, \pi) = d(\pi, \mu)$ iff $\pi = \mu$ and $d(\mu\nu, \pi\nu) = d(\pi, \mu)$, for every $\pi, \mu, \nu \in S_m$.

The algorithm rankings for our dataset collection are assumed to be generated from the following model:

$$P_\theta(\pi) = \frac{\exp\{-\theta d(\pi, \pi_0)\}}{\Psi(\theta)}, \pi \in S_m, \theta \in R, \tag{3.18}$$

where π_0 is a fixed ranking (central or modal ranking), and $\Psi(\theta)$ is a normalizing constant.

Using different distance functions, a variety of models can be derived from the above model, for example, the Mallows' θ model (with Spearman's distance) and the Mallows' ϕ model (with Kendall's distance).

The basic idea of model 3.18 is that it gives the maximum probability to the central or modal ranking π_0. The smaller the distance $d(\pi, \pi_0)$, the larger the probability that is assigned to π.

In the ART algorithm, we use the average rank vector as π_0, which turns out to be a good choice (confirmed by our experiments in previous sections). Here, we show that if the rankings of the meta-learning-based algorithm ranking problem are assumed to be generated under a multistage model, namely, the strongly unimodal model proposed in [53], the average rank vector actually is a consistent estimator to the MLE of $\hat{\theta}$, which offers an alternative explanation to the underlying mechanism of ART.

Multistage ranking models decompose a ranking generation process into independent stages. Here we focus only on the forward model because it is more natural to the algorithm selection procedure usually done by a data analyst. In the forward model, given a dataset, a data analyst selects her preferred algorithm at the first stage, the most preferred of the remaining algorithms at the second stage, and so on until the least preferred algorithm is selected. As described in [53], the correctness of the choice of an item at any stage in a multistage model is assessed according to the central ranking π_0. In the algorithm ranking context, the central ranking could be an algorithm ranking in the data analyst's mind (based on her experiences and knowledge or previous statistical analysis).

The following example is adapted from an example described in Section 2.1 of [53], which has been modified for the meta-learning-based algorithm ranking problem. Assuming we have 4 algorithms:

1. logistic regression (lgr)

2. support vector machines (svm)

3. random forests (rf)

4. a decision tree algorithm (cart)

Now, we have $\mathcal{O} = \{\mathbf{lgr}, \mathbf{svm}, \mathbf{rf}, \mathbf{cart}\}$. For a collection of datasets, in terms of the 10-fold CV-based AUC performance, we define $\pi_0 = (2, 3, 1, 4)$ as the modal ranking. Then, in a forward model, the ranking $\pi = (4, 3, 1, 2)$ means a correct choice at the first stage (because **rf** is selected as the best algorithm to use for a new dataset). An incorrect choice was made at the second stage, since among the three remaining algorithms: 1(**lgr**), 2(**svm**), 4(**cart**), **cart** was chosen as best although π_0 indicates that **lgr** is the best of the three. Finally, at the third stage, an incorrect choice is made since among the two remaining algorithms 1(**svm**) and 2(**lgr**), algorithm **svm** is selected, which disagrees with the ordering according to π_0.

In the multistage model, the probability of a ranking is affected both by the correctness of a data analyst's choice, and how close the data analyst comes to selecting the best of the remaining algorithms at each stage. In [53], this is defined as:

Let $V_\beta = \alpha$ if at stage β the $(\alpha + 1)$st best of the remaining algorithms is selected $(\alpha = 0, ..., m - \beta)$. Therefore $V_\beta = 0$ means a correct choice at stage β. In the example above with $\pi_0 = (2, 3, 1, 4)$ and $\pi = (4, 3, 1, 2)$, the values of the V_β are $V_1 = 0, V_2 = 2$, and $V_3 = 1$. Under the assumption of independent choices at each stage in the ranking process, the most general model for independent components of $\mathbf{V} = (V_1, ..., V_{m-1})$ has:

$$Pr(V_\beta = \alpha) = p(\alpha, \beta), \tag{3.19}$$

where $p(\alpha, \beta) \geq 0$ and $\sum_{\alpha=0}^{m-\beta} p(\alpha, \beta) = 1$. The multistage model on S_m is defined as:

$$Pr(\pi) = \prod_{\beta=1}^{k-1} p(V_\beta, \beta), \pi \in S_m. \tag{3.20}$$

Here π_0 can be interpreted as: when a choice has to be made between just two algorithms, the algorithm ranked better according to π_0 is preferred.

More assumptions can be made in regard to the above model. Given the following two:

For each $\beta = 1, ..., m-1, p(0, \beta) > p(1, \beta)$, and

$$p(\alpha, \beta) \text{ is a nonincreasing function of } \alpha, \alpha = 1, ..., m - \beta. \tag{3.21}$$

Model 3.20 becomes the so called strongly unimodal model [53] because conditions 3.21 imply that at each stage an algorithm with lower (better) rank according to π_0 is always at least as likely to be selected as one with a higher rank. In the strongly unimodal model, π_0 is the mode (it is the ranking that is most likely to be sampled).

Theoretical results in [53], such as Definition 6.1, 6.2, Lemma 6.1, 6.2 and Theorem 6.3, justify that for a random sample of n judges (datasets in the meta-learning problem), if $\bar{\pi}(i) = n^{-1} \sum \pi_\nu(i)$ is the average rank assigned to item (algorithm) i, and $\bar{\pi}_0(i)$ is the rank of $\bar{\pi}(i)$ among $\{\bar{\pi}(1), ..., \bar{\pi}(m)\}$, then the vector $\bar{\pi}_0$ is a consistent estimator of π_0. Because both $\bar{\pi}_0$ and the MLE converge almost surely to π_0, they must be converging toward each other [53].

The above theoretical analysis, together with our empirical results, provide further explanation why ART forests works well in practice. We believe this is also a promising direction for future work.

3.7 Conclusions

In this chapter, we have introduced a new approach for generating meta-features to meta learning (please note that the idea of extending the set of base-level features with outputs produced by learning algorithms has also been studied in the context of stacking [164]. The main difference between our method and stacking is given in Section 3.2), and we have introduced a specialised meta-learning algorithm (based on the random forests framework [25]) for predicting algorithm rankings with both theoretical and empirical analysis.

Unlike previous work, experiments in this chapter were based on an unusual large collection of publicly available datasets and a novel (and more appropriate) meta-learning experiment configuration, which does a systematic exploration of the space of parameter values of the algorithms to be ranked. Our experimental results indicate that the pairwise meta-rule generation method (Method 1) consistently improves the performances of different ranking approaches to meta-learning. The new ART forests ranker is always among the top rankers across the ranking metrics and functions we have tested. The success of the proposed methods and algorithms on the meta-learning problem studied in this chapter suggests the applicability of the new techniques to a wide range of meta-learning problems.

Chapter 4

Bagging Ensemble Selection for Classification, Regression and Ranking

In the previous chapter we have proposed a new meta-learner—ART forests, and have examined its predictive performance under different ranking accuracy metrics and functions.

"*What about if a data analyst who had designed a special ranking accuracy metric (function) would like to build a meta-learner optimizing that metric directly?*"

Although ART forests works well for the metrics that have been tested in the previous section, it is not designed for optimizing a target metric directly. In this chapter[1], we propose an ensemble learning strategy named Bagging Ensemble Selection (BES), which takes a set of meta-learners (e.g., the ones described in the previous chapter) as the base-level learners to form an ensemble model. The ensemble construction stage of BES is guided by a user-specified "loss function" which provides a solution to our question.

In the first part of this section, an extensive set of experiments has been

[1]Part of the research presented in this chapter has appeared in [144, 145]

conducted to study the performance of the proposed BES strategy for both classification and regression problems. In later sections, we present results for applying the BES strategy to meta-learning for algorithm and parameter ranking.

4.1 BES for Classification

The problem of constructing an ensemble of classifiers from a library of base classifiers has always been of interest to the data mining community. Usually, compared with individual classifiers, ensemble methods are more accurate and stable. We here reproduce the mathematical expression used in [114] to illustrate the idea of ensemble learning.

Let x be an instance and $m_i, i = 1...k$, a set of base classifiers that output probability distributions $m_i(x, c_j)$ for each class label $c_j, j = 1...n$. The output of the final classifier ensemble $y(x)$ for instance x can be expressed as:

$$y(x) = \arg\max_{c_j} \sum_{i=1}^{k} w_i m_i(x, c_j), \tag{4.1}$$

where w_i is the weight of base classifier m_i. In this particular form, ensemble learning strategies can be seen as methods for calculating optimal weights for each base classifier in terms of a classification goal. Since the mid-90's, many ensemble methods have been proposed. For a more detailed review of recent developments please refer to [21, 129].

Before introducing the new methods, we briefly review bagging (bootstrap aggregating) [24] and the ensemble selection algorithm proposed in [34]. Bagging is based on the instability of base classifiers, which can be exploited to improve the predictive performance of such unstable base classifiers. The basic idea is that, given a training set T of size n and a classifier A, bagging generates m new training sets with replacement, T_i, each of size $n' \leq n$. Then, bagging applies A to each T_i to build m models. The final output of bagging

is based on simple voting [21].

Ensemble selection is a method for constructing ensembles from a library of base classifiers [34]. Firstly, base models are built using many different machine learning algorithms. Then a construction strategy such as forward stepwise selection, guided by some scoring function, extracts a well performing subset of all models. The simple forward model selection based procedure proposed in [34] works as follows: (1) start with an empty ensemble; (2) add to the ensemble the model in the library that maximizes the ensemble's performance will respect to some error metric on a hillclimb set; (3) repeat Step 2 until all models have been examined; (4) return that subset of models that yields maximum performance on the hillclimb set.

One advantage of ensemble selection is that it can be optimised for many common performance metrics, or even a combination of metrics. For variants of the ensemble selection algorithm, the reader is referred to [34, 33]. In the next section, we will describe the proposed bagging ensemble selection algorithms and explain the motivation of combining bagging and ensemble selection.

4.2 The Bagging Ensemble Selection Strategy

Based on the data sets and comparison results from [34], the simple forward model selection based ensemble selection algorithm is superior to many other well-known ensemble learning algorithms, such as stacking with linear regression at the meta-level, bagging decision trees, and boosting decision stumps. However, sometimes ensemble selection overfits the hillclimbing set, reducing the performance of the final ensemble.

Figure 4.1 shows the hillclimb and test set learning curves of running ensemble selection on a data set. The red curve is the hillclimb set performance and the blue curve is the test set performance. It demonstrates that as the number of models in the model library increases, the performance (in terms of

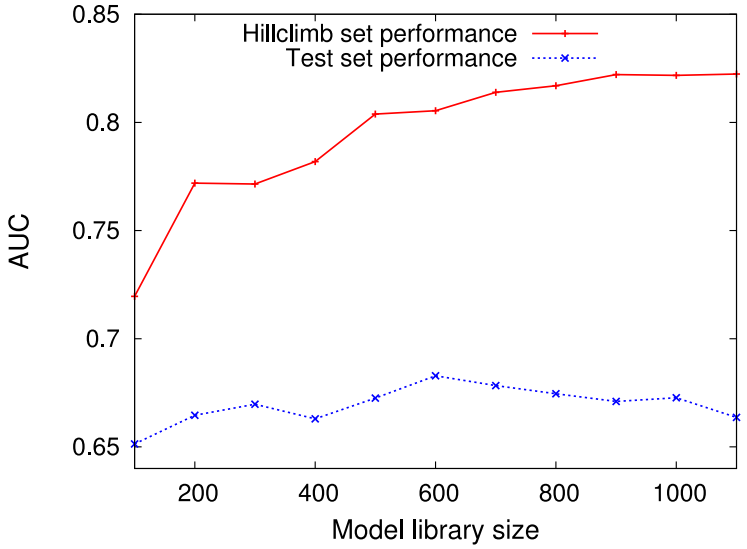


Figure 4.1: The KDD 09 customer churn data

AUC) of ensemble selection on the hillclimb set gradually increases. However, the corresponding performance on the test set does not always increase; it may reach a peak (local or global) and then gradually decline.

Also, as indicated in [34], for certain data sets, the root-mean-squared-error metric sometimes can decline very quickly. To overcome this problem, the authors of [34] proposed three additions to the simple forward selection procedure to reduce the chance of the hillclimb set overfitting. The proposed additions are: (1) selection with replacement, where each individual classifier can be selected multiple times, which means some classifiers get larger weights than others; (2) sorted ensemble initialization, where instead of starting with an empty ensemble, models in the library are sorted by their performance, and the best N models are put into the initial ensemble; (3) "bagged" ensemble selection, where K groups (bags) of models are randomly selected from the model library, and ensemble selection is done inside each bag; the final ensemble is the union of the subsets selected for each of the bags. All three procedures also introduce additional parameters to the simple ensemble selection algorithm.

Furthermore, there is one more issue: how much data should be used for

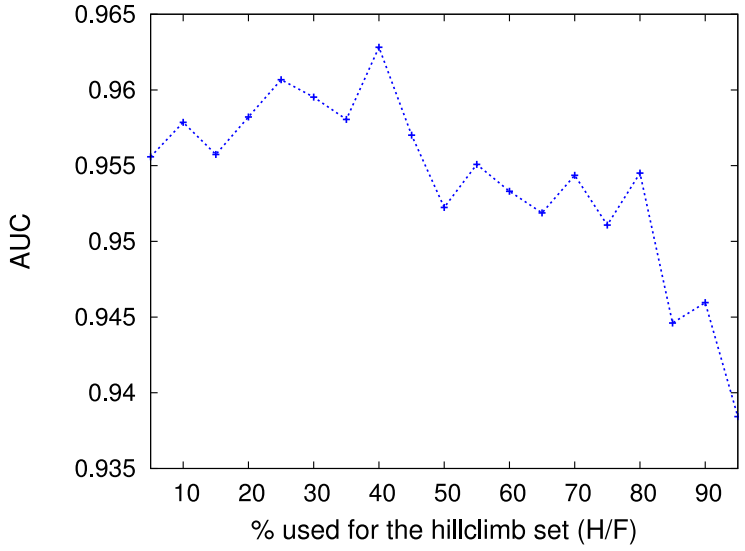


Figure 4.2: The waveform-5000 data

the hillclimb set? Figure 4.2 shows a typical test set learning curve for running ensemble selection with hillclimb sets of varying sizes. Assume the training set is F, and the hillclimb set H is a subset of F. Here, the x-axis shows the ratio H/F and indicates the percentage of F that is used for the hillclimb set. Based on the learning curve, we can see that the performance of ensemble selection is not stable, and is related to how much data is used for H. In the figure, there is a performance peak at $x = 40\%$, but performance starts to drop from $x = 50\%$. Different data sets may have different optimal ratios, which usually can be found only by using cross-validation. Therefore, this parameter indirectly increases the complexity of ensemble selection.

Based on these observations, we propose a new ensemble learning algorithm called Bagging Ensemble Selection (BES): if we view the simple forward ensemble selection algorithm as an unstable base classifier, then we can apply the bagging idea to construct an ensemble of simple ensemble selection classifiers, which should be more robust than an individual ensemble selection classifier. In addition, the respective out-of-bag samples can be used as the hillclimb set. Specifically we will use the following three variations of bagging ensemble

Algorithm 4.1 The BES-OOB algorithm

Inputs Training set S; Ensemble Selection classifier E; Integer T (number of bootstrap samples)

$U \leftarrow \emptyset$

for $i = 1 \rightarrow T$ **do**

S_b = bootstrap sample from S (i.i.d. sample with replacement)

S_{oob} = out of bag sample

train base classifiers (can be a diverse model library) in E on S_b

E_i = do ensemble selection based on base classifiers' performance on S_{oob}

$U = U \bigcup E_i$

end for

return U

selection:

1. The **BaggingES-Simple** algorithm is the straightforward application of bagging to ensemble selection, with ensemble selection being the base classifier inside bagging. In this algorithm, the amount of data used for the hillclimb set is still a user-specified parameter (with a default of 30%). Each bootstrap sample is split into a train and a hillclimbing set according to this parameter.

2. The **BaggingES-OOB** algorithm uses the full bootstrap sample for model generation, and the respective out-of-bag instances as the hillclimb set for selection. The bootstrap sample is expected to contain about $1 - 1/e \approx 63.2\%$ of the unique examples of the training set [10, 24]. Therefore the hillclimb set (out-of-bag sample) is expected to have about $1/e \approx 36.8\%$ unique examples of the training set for each bagging iteration. An advantage of BaggingES-OOB is that the user does not need to choose the size of the hillclimb set. Algorithm 4.1 shows the pseudocode for training the BaggingES-OOB ensemble.

3. The **BaggingES-OOB-EX** algorithm is an extreme case of BaggingES-

OOB, where in each bagging iteration only the single best classifier (in terms of performance on the hillclimb set) is selected. Therefore, if the number of bagging iterations is set M, then the final ensemble size will be exactly M as well.

4.3 Experimental Results

We experiment with ten classification problems. All of them are real world data sets which can be downloaded from the UCI repository [54], the UCSD FICO data mining contest website[2] and the KDD Cup 2009 website[3]. These data sets were selected because they are large enough, and they come from very different research and industrial areas.

Original data sets			Final binary data sets
Data set with release year	#Insts	Atts:Classes	Class distribution (#Insts)
Adult 96	48,842	14:2	23% vs 77% (10,000)
Chess 94	28,056	6:18	48% vs 52% (8,747)
Connect-4 95	67,557	42:3	26% vs 74% (10,000)
Covtype 98	581,012	54:7	43% vs 57% (10,000)
KDD09 Customer Churn 09	50,000	190:2	8% vs 92% (10,000)
Localization Person Activity 10	164,860	8:11	37% vs 63% (10,000)
MAGIC Gamma Telescope 07	19,020	11:2	35% vs 65% (10,000)
MiniBooNE Particle 10	130,065	50:2	28% vs 72% (10,000)
Poker Hand 07	1,025,010	11:10	45% vs 55% (10,000)
UCSD FICO Contest 10	130,475	334:2	9% vs 91% (10,000)

Table 4.1: Data sets: basic characteristics

Table 4.1 shows the basic properties of these data sets.

To make experiments possible for large model libraries, selecting from thousands of base classifiers, all five multiclass data sets were converted to binary problems by keeping only the two largest classes each. After this conversion to binary problems, for data sets that are larger than 10,000 instances, a subset of 10,000 instances is randomly selected for our experiments. Table 4.1 (in the

[2]The University of California, San Diego and FICO 2010 data mining contest, http://mil.ucsd.edu/

[3]The KDD Cup 2009, http://www.kddcup-orange.com/

rightmost column) shows the basic properties of the final data sets.

Ensemble selection is not restricted by the type of base classifiers used. Theoretically, any classifier can be used as a base classifier for ensemble selection. In this experiment, the WEKA [74] implementation of the random tree classifier is used as the base classifier for all experiments. There are two reasons for focussing solely on random trees as base classifiers. The first one is simplicity: just by varying a single parameter, the random seed, we can obtain a large and relatively diverse model library. The second one is fair comparison: most other ensemble methods are limited to uniform base classifiers. To speed up our experiments, parameter K of the random tree, the number of random attributes, is always set to 5, and the minimum number of instances at each leaf node is set to 50.

In [34], the authors have shown that ensemble selection can be optimised to many common evaluation metrics. Bagging ensemble selection inherits this very useful feature; the goal metric is therefore a user-specified parameter. In this experiment, the AUC (area under the ROC curve) metric is used for all experiments. The numeric value of the AUC metric is calculated as the weighted sum across classes.

The following sections present two sets of results. One shows the results from comparing the three bagging ensemble selection algorithms to the simple forward ensemble selection algorithm (ES) and the ES++ algorithm, which is the improved version of ES with the three additions, as described in the introduction. This is followed by an analysis of the final ensemble sizes for these algorithms. The other set of results shows a comparison between bagging ensemble selection and other ensemble learning algorithms.

4.3.1 Comparison to the Forward ES Algorithms

In this experiment the following setup is used: the number of bags (bagging iterations) for BaggingES-Simple, BaggingES-OOB and BaggingES-OOB-EX is set to 50. For each data set, we run 10 experiments per algorithm, increasing the size of the model library per bag by 10 for each successive experiment: from 10 to 20, then to 30 and so on until 100 for the tenth experiment. That is, when the size of the model library is 100, then, in total, 5,000 base classifiers (random trees) are trained.

Accordingly, we run 10 experiments on each data set for the ES algorithm and the ES++ algorithm (hillclimb ratio is set to 30% for both ES and ES++) that we want to compare. The size of the model library increases by 500 in each successive experiment, from a base 500 to 1,000, then 1,500 until it reaches 5,000 in the tenth experiment, which means all five algorithms in the comparison use the same number of base classifiers in each individual experiment. Also, for the ES++ algorithm, the number of subgroups is set to 50.

Figure 4.3 shows the test set learning curves of the ES algorithm, the ES++ algorithm, and the three bagging ensemble selection algorithms based on 500 individual experiments (5 algorithms, 10 data sets, 10 different model library sizes per data set). For each experiment, the algorithms are trained on 66% of the data set and evaluated on the other 34%. We repeated each experiment five times and the mean values were used for generating the figures and comparison. Based on Figure 4.3, we can see that ES and ES++ outperform bagging ensemble selection when the size of the model library is greater than 1,000 on the Adult-96 data set. For all other nine data sets, bagging ensemble selection, particularly BaggingES-OOB (blue curves) and BaggingES-OOB-EX (green curves), clearly outperform the ES algorithm and the ES++ algorithm. For data sets Chess-94, KDD-09 and Localization-10, BaggingES-OOB and BaggingES-OOB-EX gave similar performance.

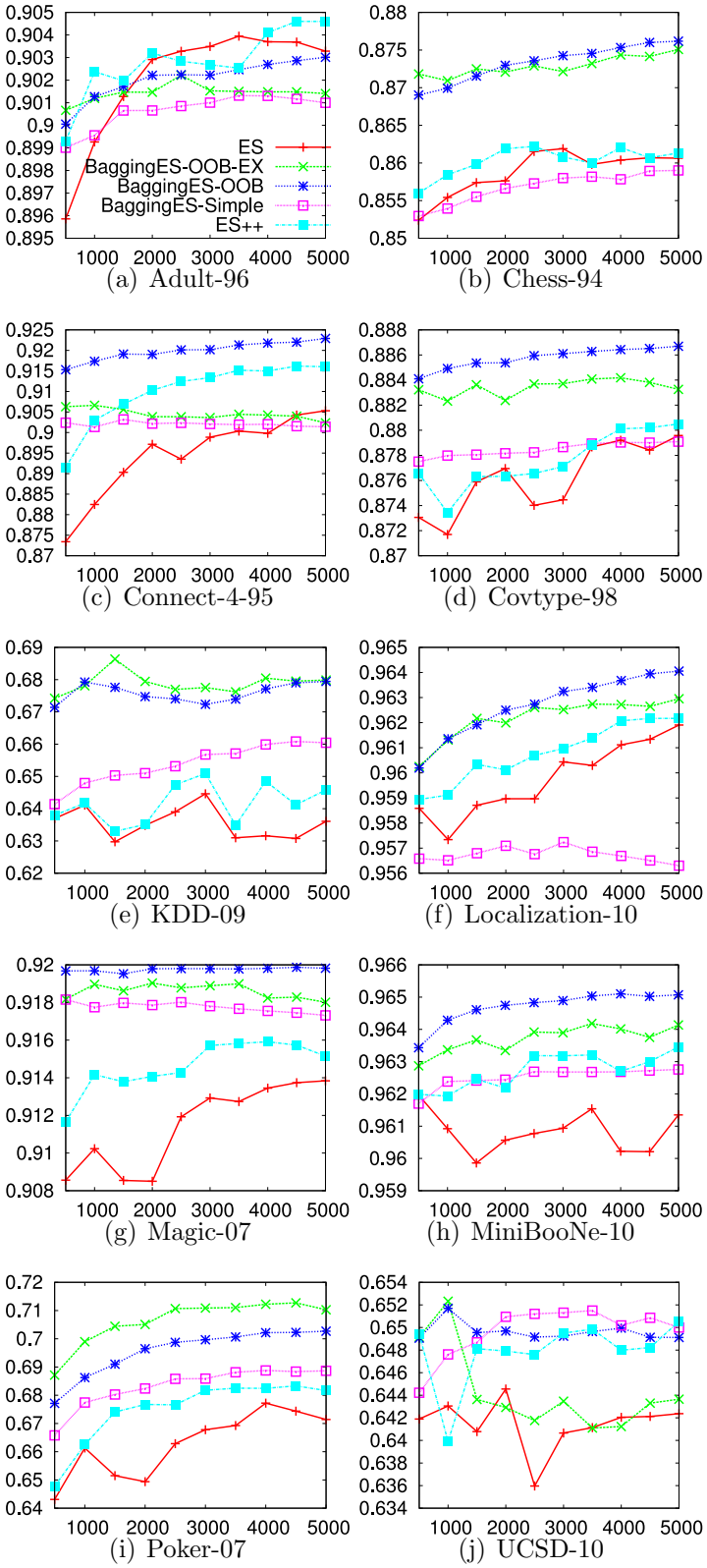


Figure 4.3: Learning curves of ES, ES++ and the three bagging ensemble selection algorithms. X-axis is the model library size; y-axis is the AUC performance

An interesting pattern is that, for data sets Connect-4-95, Magic-07 and UCSD-10, the test performance of BaggingES-OOB-EX declines as the size of the model library increases. This is probably due to the fact that model diversity is more important for these data sets than for others. Thus, as the model library gets larger and larger, the best base classifier of each of the 50 bags of BaggingES-OOB-EX might become more similar to each other, thus losing model diversity.

For 6 out of 10 model library sizes, the BaggingES-Simple algorithm outperforms all other algorithms on the UCSD-10 data set. The ES++ algorithm outperforms other algorithms on the UCSD-10 data set when model library sizes are 500 and 5,000, but had a relatively poor performance when model library size is 1,000. Again, we can see that, for Covtype-98, KDD-09, MiniBooNe-10 and UCSD-10, the learning curves of the ES algorithm are not very stable.

Figure 4.4 (upper panel) shows the histogram presentation of the performance in terms of the number of wins for each algorithm over the ten data sets. We can see that BaggingES-OOB and BaggingES-OOB-EX are the top two winners.

Next, we look at the final ensemble sizes of ES, ES++, BaggingES-OOB, BaggingES-OOB-EX and BaggingES-Simple. Figure 4.5 shows the relationship between model library size and the final ensemble size for these algorithms on the ten data sets. Please note that the final ensemble size of BaggingES-OOB-EX is always 50 because the number of bagging iterations is set to 50. Except for the BaggingES-OOB-EX algorithm, we can see that the final ensemble size of the other four ensemble algorithms increases linearly or sublinearly as the size of the model library increases (note that the y-axis is logarithmic). The final ensemble size of BaggingES-OOB, ES, and ES++ grows relatively faster than BaggingES-Simple's ensemble size. One possible reason is that

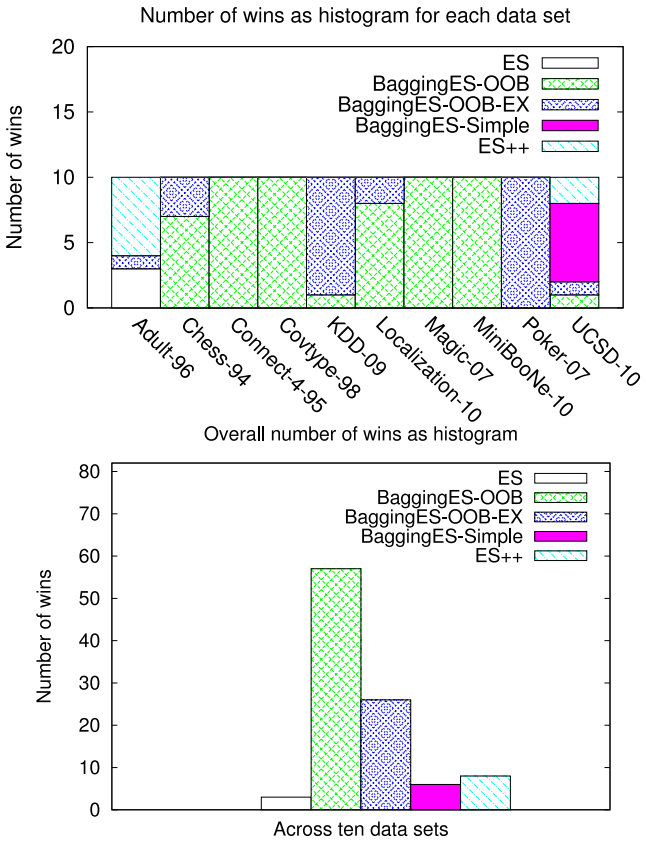


Figure 4.4: Histogram presentation for counting number of wins for each algorithm

in Bagging-OOB-Simple, the size of the build set (training set excluding the hillclimb set) is relatively small compared to BaggingES-OOB. Theoretically, for BaggingES-OOB, the hillclimb set (out-of-bag sample) has 36.8% unique instances of the training set, and the training set has 63.2% unique instances; however, BaggingES-Simple uses the bootstrap sample for both training and hillclimbing. For this experiment, the hillclimb ratio for BaggingES-Simple is set to 30%, thus its hillclimb set has fewer unique instances than BaggingES-OOB's hillclimb set. Therefore adding more base classifiers to BaggingES-Simple's model library may not necessarily improve the hillclimb performance since the hillclimb set might be too simple and the local hillclimb performance maximum could be achieved quickly.

Another interesting pattern is that ES has a much smaller ensemble size than BaggingES-OOB and BaggingES-Simple have. This could be because the

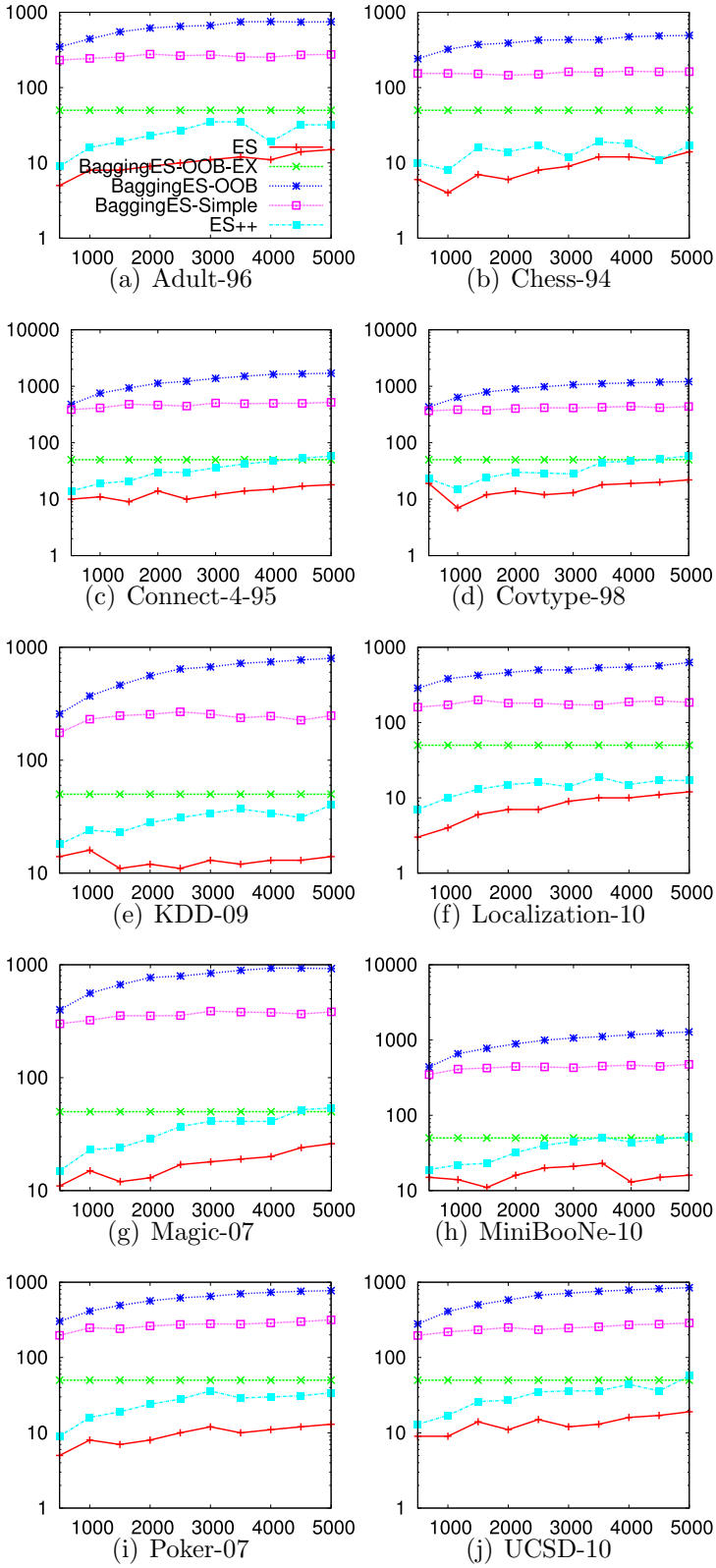


Figure 4.5: Final ensemble sizes of ES, ES++ and the three bagging ES based algorithms. X-axis is the model library size; y-axis is the final ensemble size in logarithmic scale

local performance maximum of ES on the hillclimb set can be achieved more quickly compared to bagging ensemble selection. Again, adding more base classifiers to ES's model library may not necessarily improve the hillclimb performance.

Based on those observations, it seems that one reason for the good performance of BaggingES-OOB is that it usually has a larger final ensemble compared to all other algorithms. However, this does not imply that a larger final ensemble always yields better predictive performance. Refer to the learning curves in Figure 4.3, for data sets Chess-94, KDD-09 and Poker-07: BaggingES-OOB-EX's performance is competitive with BaggingES-OOB even though its final ensemble size is only 50. Therefore, whenever final ensemble size is crucial, for example, when an application requires fast real-time prediction, then the BaggingES-OOB-EX algorithm should be considered.

To sum up, we conclude that the advantage of the BaggingES-OOB algorithm and the BaggingES-OOB-EX algorithm over ES/ES++ is that their ensembles are evaluated on diverse hillclimb sets generated by the bagging procedure, and therefore are more robust and stable.

4.3.2 Comparison to Other Ensemble Algorithms

In this experiment, we compare BaggingES-OOB (the most successful variant of the bagging ensemble selection based algorithms) to other popular ensemble learning methods. The following algorithms (WEKA [74] implementations) are evaluated: Voting with probability averaging, stacking with linear regression at the meta-level (Stacking), AdaBoostM1, and RandomForest. ES++ is also included for comparison. All ensemble algorithms use the random tree as the base classifier. The total number of base classifiers allowed to be trained for each ensemble algorithm is equal. For bagging ensemble selection the number of bags is set to 50, and the number of base classifiers of individual ensemble

selection in each bag is set to 100; thus in total 5,000 base classifiers (random trees) are trained. For other ensemble algorithms, the number of base classifiers is set to 5,000. The training complexity of random tree is $O(nlogn)$, where n is the size of the training set. In this experiment, all ensemble algorithms train on the same number of random trees, therefore the training costs for the model library of each ensemble algorithm in this comparison are roughly the same.

Table 4.2 shows the performance of each algorithm on the ten data sets. Standard deviations and significant test results were calculated from five independent runs of 66% (training) versus 34% (testing) split validation. The results for which a significant difference with BaggingES-OOB was found, are marked with a "*" or "○" next to them. An asterisk "*" next to a result indicates that BaggingES-OOB was significantly better than the respective method (column) for the respective data set (row). A circle "○" next to a result indicates that BaggingES-OOB was significantly worse than the respective method. We can see that AdaBoost.M1 significantly outperforms BaggingES-OOB on the Chess-94 and the Poker-07 data sets. On the other eight data sets, BaggingES-OOB is competitive (7 ties) to or superior (41 significant wins) to all other ensemble algorithms.

4.4 BES for Classification Summary

Ensemble selection is a popular ensemble learning method. Over the past several years, ensemble selection has been empirically examined and has proven to be a very effective and accurate ensemble learning strategy. One disadvantage of ensemble selection is that it is unstable and sometimes overfits the hillclimb set. In these experiments, to further improve ensemble selection we proposed using the bagging strategy, which utilises this instability in order to reduce the variance of a single ensemble selection. Our experiments on ten real world

Dataset \ Algorithm	BES-OOB	Voting	Stacking	AdaBst.M1	RandomFrst	ES++
Adult-96	.905±.001	.902±.002•	.892±.004•	.783±.008•	.902±.002•	.906±.002
Chess-94	.875±.004	.859±.003•	.841±.011•	.971±.002○	.862±.004•	.866±.003•
Connt-4-95	.918±.006	.911±.006•	.897±.007•	.905±.005•	.912±.006•	.916±.005
Covtype-98	.884±.002	.882±.002•	.875±.004•	.878±.003•	.882±.002•	.881±.001•
KDD-09	.678±.029	.678±.027	.656±.031•	.580±.011•	.675±.029	.669±.029
Localiz-10	.966±.002	.957±.002•	.940±.006•	.938±.004•	.960±.002•	.963±.003•
Magic-07	.920±.004	.916±.004•	.910±.004•	.868±.005•	.919±.004•	.913±.002•
MiniB-10	.964±.002	.963±.002•	.959±.002•	.928±.006•	.963±.002•	.963±.001•
Poker-07	.697±.018	.660±.022•	.620±.041•	.740±.007○	.674±.018•	.671±.020•
UCSD-10	.649±.011	.648±.008	.612±.016•	.632±.010•	.646±.008	.646±.007•
(win/tie/loss)		(0/2/8)	(0/0/10)	(2/0/8)	(0/2/8)	(0/3/7)

"•" BaggingES-OOB is significantly better, "○" BaggingES-OOB is significantly worse, level of sign. 0.05

Table 4.2: Mean and standard deviation of the AUC performance of BaggingES-OOB and five other popular ensemble learning methods

problems show that the bagging ensemble selection, especially BaggingES-OOB, which uses the out-of-bag sample as the hillclimb set, yields a robust and more accurate classifier ensemble than the original ensemble selection.

When the underlying problem requires fast prediction, we suggest using BaggingES-OOB-EX instead, because the user can easily control the size of the final ensemble. In terms of predictive performance, bagging ensemble selection is also competitive (in many cases, superior) to other state-of-art ensemble learning algorithms, such as voting, random forest, stacking and boosting. Again, bagging ensemble selection is not restricted by the type of base classifiers.

We experimented with only one type of base classifier in this section, but to get the best out of the algorithm, we suggest using a more diverse model library. The bagging ensemble selection idea can be easily generalised to regression problems, since bagging is applicable to both classification and regression. In the next section, we will compare bagging ensemble selection to other ensemble methods for regression problems.

4.5 BES for Regression

In a typical regression setting, a given training set D consists of m instances, such as $D = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_m}, y_m)\}$, where $\mathbf{x_i}$ is an instance and y_i is a target, the task is to learn an approximate function $f : X \to \mathbb{R}$ of the true function f^0 from D. Let $f_j, j = 1...k$, be a set of base regression learners that output predictions $f_j(\mathbf{x_i})$. The output of a simple regression ensemble $F(\mathbf{x_i})$ for instance $\mathbf{x_i}$ can be expressed as:

$$F(\mathbf{x_i}) = \sum_{j=1}^{k} w_j f_j(\mathbf{x_i}), \tag{4.2}$$

where w_j is the weight of base learner f_j. In this particular form, ensemble learning strategies can be seen as methods for calculating optimal weights for each base learner in terms of a regression goal. Since the mid-70s, many

ensemble strategies have been proposed. We first review a few state-of-the-art ensemble strategies for regression.

Gradient Boosting [58] is a classical ensemble learning algorithm. It produces an ensemble of base learners (e.g., decision trees) based on a stage-wise procedure to optimise an arbitrary differentiable loss function.

Stochastic Gradient Boosting [57] is an extension of the Gradient Boosting algorithm, where at each iteration, a base learner trains on a subset of the training set drawn at random without replacement.

Bagging (bootstrap aggregating) [24] is based on the instability of base learners, which can be exploited to improve the predictive performance of such unstable base learners. The basic idea is that, given a training set T of size n and a learner A, bagging generates m new training sets with replacement, T_i. Then, bagging applies A to each T_i to build m models. The final output of bagging is based on simple averaging [24]. For instance, in a regression setting using Eq. 1, the weight w_j for f_j is k^{-1}.

MultiBoosting [162] is an ensemble algorithm designed to reduce both variance and bias simultaneously, in which Boosting is used as the base learner for Bagging. For a more detailed review of recent developments on ensemble learning strategies please refer to [129, 117, 16, 171, 47]. Next, we discuss the motivations for proposing and studying the bagging ensemble selection (BES) strategy.

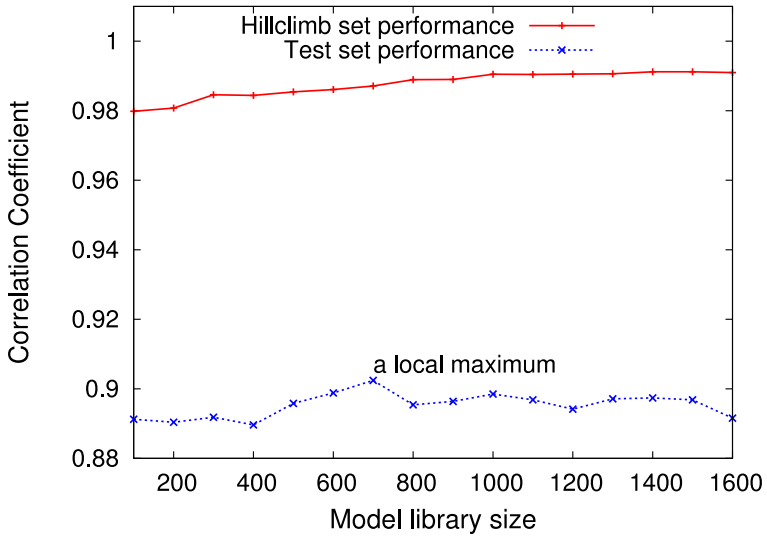


Figure 4.6: Examples of the hillclimb set overfitting problem of the Ensemble Selection strategy on the Boston housing-price data

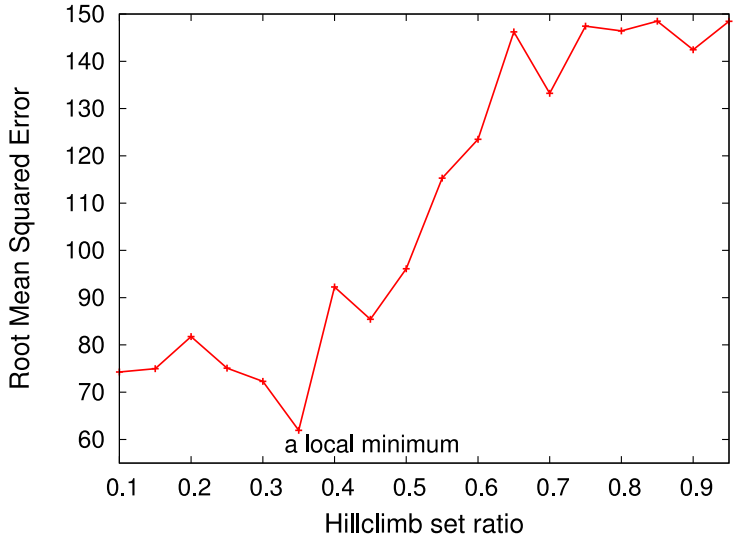


Figure 4.7: Examples of the hillclimb set ratio problem of the Ensemble Selection strategy on the CPU performance data

Experimental results in [34, 144] and the previous sections show that in the classification setting, the simple ES strategy sometimes overfits the hillclimb set, reducing its predictive performance. Our preliminary experimental results for employing ES on regression problems also identified a similar phenomenon. Figure 4.6 shows an example of the hillclimb set overfitting problem of ES on

the *Boston housing price* data. The red curve (top) is the hillclimb set performance; the blue curve (bottom) is the test set performance. We can see that as the size of the model library increases, the hillclimb set performance also improves gradually. However, the test set performance does not always improve. In this case, the local optimal performance is achieved when the model library size is about 700. Another practical issue is that users have to estimate the optimal hillclimb set ratio for a given data set. Figure 4.7 shows an example on the *CPU* data based on cross-validation based performance estimation. We can see that the local optimal performance is achieved when the hillclimb ratio is about 0.35; after that, the performance starts to drop. Although we could use cross-validation to estimate the hillclimb set ratio, this would substantially increase the practical training cost of ES. To overcome these problems and improve the predictive performance of the ES strategy, three BES strategies have been proposed in [144]. Experimental results show that, under the classification setting, the BES-OOB strategy is the most successful variant in terms of predictive performance. In this experiment, we focus on examining the predictive performance of the BES-OOB strategy for regression problems.

4.6 BES Regression Experiments

In this section, we conduct a series of experiments and statistical tests to examine the performance of the BES-OOB ensemble strategy for regression.

4.6.1 Comparison to Other Ensemble Strategies

Dataset	BES-OOB	SGB	BG	BSGB
quake	0.12	0.06 •	0.12	0.12
cholesterol	0.19	0.07 •	0.18	0.19
detroit	0.22	0.03 •	0.24	0.24
breastTumor	0.22	0.16 •	0.22	0.22
meta	0.38	0.14 •	0.36 •	0.38
veteran	0.42	0.26 •	0.42	0.42
schlvote	0.45	0.10 •	0.46	0.43
sensory	0.53	0.48 •	0.53	0.52
longley	0.54	0.43 •	0.54 •	0.50 •
strike	0.55	0.41 •	0.55	0.55
kidney	0.55	0.38 •	0.53 •	0.58 ○
baskball	0.55	0.43 •	0.55	0.56
newton-hema	0.57	0.54	0.58 ○	0.59 ○
pbc	0.57	0.52 •	0.56 •	0.58 ○
stanford	0.60	0.43 •	0.63	0.63
sleep	0.64	0.52 •	0.64	0.62
hungarian	0.65	0.63 •	0.65 •	0.66 ○
winequality-red	0.66	0.61 •	0.66 •	0.68 ○
echoMonths	0.67	0.69 ○	0.69 ○	0.68
winequality-white	0.68	0.62 •	0.67 •	0.70 ○
cleveland	0.69	0.63 •	0.69	0.70 ○
pollution	0.75	0.51 •	0.73 •	0.75
vineyard	0.76	0.67 •	0.75 •	0.76
lowbwt	0.79	0.78 •	0.79	0.79
elusage	0.82	0.81	0.82	0.84 ○
vinnie	0.86	0.85 •	0.86 ○	0.86 ○
bolts	0.86	0.83	0.83 •	0.86
gascons	0.88	0.76 •	0.84	0.83
cloud	0.91	0.84 •	0.90 •	0.91
autoMpg	0.91	0.92	0.91 •	0.93 ○
servo	0.92	0.91	0.91 •	0.93 ○
pwLinear	0.92	0.92	0.92	0.93 ○
housing	0.92	0.90 •	0.91 •	0.93 ○
boston	0.92	0.91 •	0.92 •	0.93 ○
socmob	0.92	0.92	0.91 •	0.94 ○
autoHorse	0.93	0.91 •	0.90 •	0.93
autoPrice	0.93	0.92 •	0.93 •	0.94 ○
cpu	0.97	0.93 •	0.96 •	0.98
strikes	0.98	0.97 •	0.96 •	0.98 •
fishcatch	0.98	0.96 •	0.97 •	0.97 •
visualizing-galaxy	0.99	0.98 •	0.98 •	0.99 ○
bodyfat	0.99	0.98 •	0.98 •	0.98 •

• ○, BES-OOB is significantly better or worse

	BES-OOB against		
	SGB	BG	BSGB
win/tie/loss	34/7/1	23/16/3	4/21/17

Table 4.3: Estimated correlation coefficients of BES-OOB, SGB, BG, and BSGB; and Win/tie/loss counts of *paired t-test*.

Firstly we compare BES-OOB to three state-of-the-art ensemble strategies for regression: Stochastic Gradient Boosting (SGB) [57], standard Bagging (BG) [24] and an ensemble of Bagging and Stochastic Gradient Boosting, denoted by BSGB. BSGB can be seen as a variant of the MultiBoosting algorithm [162], in which SGB is used as a base learner for Bagging. The experiments are based on 42 regression data sets from UCI repository[4] and StatLib[5]. We use 10 times 10-fold cross-validation to estimate the performance of each strategy. Then, several statistical significance tests are conducted, including the non-parametric *Friedman-test* and the *Bonferroni-Dunn test* as described in [45]. This approach utilises the ranking information of each learner in comparison, which is suitable for comparing multiple learners on multiple data sets. The total numbers of win, tie and loss for the *paired t-test* (with significance level 0.05) are also recorded. To fairly compare the four strategies, REPTree (a CART-like regression tree) [74] is used as the base learner. The ensemble size is set to 1,500 for all these strategies. For SGB, the shrinkage parameter is set to 0.5 and the subsample size parameter is set to 50%. For BES-OOB, the number of base learners per "bag" is set to 30, and the number of bagging iterations is set to 50. Also, BES-OOB is set to optimise the correlation coefficient metric. For BSGB, the number of base learners for SGB (shrinkage is set to 0.5; subsample size is 50%) is set to 30, and the number of bagging iterations is set to 50. Table 4.3 presents the *paired t-test* results. Correlation coefficient scores are reported. Figure 4.8 is the graphical representation of the *Friedman-test* for the four strategies. We can see that both BES-OOB and BSGB significantly outperform BG and SGB, and BG significantly outperforms SGB. There is no significant difference between BES-OOB and BSGB's performance over the 42 data sets.

[4]http://archive.ics.uci.edu/ml

[5]http://lib.stat.cmu.edu

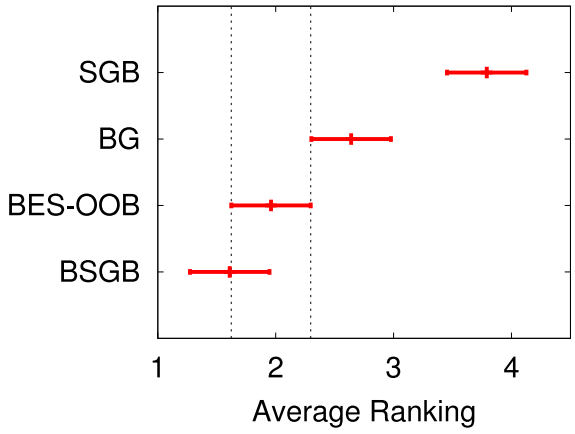


Figure 4.8: Visualization of the *Friedman-test* results for BES-OOS, SGB, BG, and BSGB with REPTree as base learners over 42 data sets. The middle point of each bar indicates the average rankings, and the bars indicate the critical values of the *Bonferroni-Dunn test* (two-tailed test at significance level 0.05). Strategies having non-overlapped bars are significantly different.

4.6.2 Diverse Model Libraries

In the previous experiments, we have been testing on a single type base learner. However, one distinguishing feature of BES-OOB is that it can use different types of base learners. In this section, BES-OOB with a diverse model library consisting of three types of base learners (REPTree, SVM regression and M5P model tree [123]), denoted by BES-diverse, is compared to BES-OOB with only one of the three base learners, denoted by BES-reptree, BES-svm, and BES-m5p, respectively. Three different model library sizes are tested: 3, 30 and 300. The experimental setup is as follows: the number of bagging iterations for all BES strategies in comparison is set to 30; for BES-diverse, when the model library size is 3, only one of each type of base learners is used; when the model library size is 30, 10 of each type of base learners are used; so 100 of each type of the base learners are used for a model library size of 300. The correlation coefficient is set as the goal metric for all strategies.

Diversity is one of the key factors for ensemble learning. To simplify the procedure for generating diverse base learners, we adopt the "random sub-

Model library size = 3			
	BES-OOB-diverse against		
	A1	A2	A3
win/tie/loss	4/20/18	32/8/2	5/13/24

Model library size = 30			
	BES-OOB-diverse against		
	A1	A2	A3
win/tie/loss	19/15/8	34/6/2	9/20/13

Model library size = 300			
	BES-OOB-diverse against		
	A1	A2	A3
win/tie/loss	5/37/0	14/28/0	4/38/0

Table 4.4: Win/tie/loss counts of *paired t-test* for BES-OOB-diverse against BES-OOB-reptree (A1), BES-OOB-svm (A2) and BES-OOB-m5p(A3).

space" idea [28] for each base learner in the library. That is, each base learner trains on a random subset (33% is used for all experiments) of the original variables. For REPTree, Weka default parameters are used, and we also randomly set its random seed; for SVM regression, we use the LibSVM default parameters for epsilon-SVM regression and RBF kernel, except the gamma value is randomly set to be between 0 and 1. We use the Weka default parameters for M5P model tree. Table 4.4 shows the *paired t-test* results of BES-OOB-diverse against BES-OOB-reptree, BES-OOB-svm and BES-OOB-m5p under three different model library sizes: 3, 30, and 300, respectively. Please note that the number of bagging iterations is set to 30. Therefore, the numbers of base learners that are allowed to be built for the three model library sizes are: 90, 900, and 9,000, respectively.

Figure 4.9 shows the average *Friedman-test* rankings of each strategy under

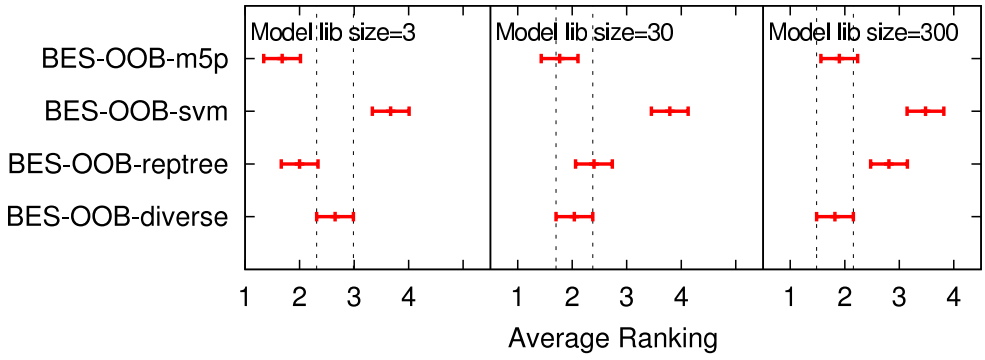


Figure 4.9: Friedman average rankings under different model library sizes.

the three model library sizes. We can see that the ranking of BES-OOB-diverse improves (from the third to the second, and finally to the first) when the model library size increases. The result implies that the advantage of using a diverse model library becomes clear when the model library size is relatively large.

4.6.3 Pruning an Ensemble of ES Ensembles

Until now, we have been using the standard output aggregation method for BES-OOB. That is, the final prediction of BES-OOB is simply the average of all individual ES learners. The final ensemble size of BES-OOB is therefore the number of bagging iterations. In this section, we consider methods for ensemble pruning. Usually, there are two main reasons for doing ensemble pruning. The first is to reduce the prediction cost (e.g., runtime or memory requirements) without sacrificing too much predictive performance. The second is to obtain a more accurate model. Based on the theoretical work of [172] in the study of neural networks, we know that theoretically "many could be better than all". This implies that the performance of an optimal subset of base learners may outperform the population average. Since the default BES-OOB strategy uses simple averaging, appropriate ensemble pruning may improve BES-OOB's performance in terms of both accuracy and prediction cost. We compare simple averaging to two pruning methods: pruning with the cocktail ensemble (CE)

algorithm, and pruning with the stacking strategy using the non-negative least-squares (NNLS) algorithm as the meta-level learner. The three methods in this comparison are denoted by BES-OOB-avg, BES-OOB-ce, and BES-OOB-nnls, respectively. Next, we briefly introduce the BES-OOB-ce and the BES-OOB-nnls methods.

Cocktail ensemble (CE) [168], is a novel method of ensemble learning. One reason for using CE as an ensemble pruning method for BES-OOB is that the authors explicitly mentioned that the method is proposed for learning ensemble of ensembles. Since combination of multiple ensembles (equivalent to finding the optimal weights for each base learner) is an NP-hard problem [75], the authors of [168] proposed using the pair-wise combination for multiple ensembles. In addition, CE has an appealing mathematical foundation, which we will briefly discuss here. For a full account of the method, we refer readers to [168]. The basic idea is that, given two ensembles f_1 and f_2, a linear ensemble of ensembles f_1 and f_2 can be expressed as:

$$f^c = pf_1 + (1-p)f_2, \;\; wrt \;\; p \in [0,1]$$

where p is the weight for f_1 and $1-p$ is the weight for f_2. Then, the optimal weight of f_1 is:

$$p^* = \frac{E_2 - E_1}{2\triangle} + 0.5, \tag{4.3}$$

where E_1 and E_2 are the generalization errors of f_1 and f_2, and $\triangle = \mathbb{E}_x[(f_1 - f_2)^2]$ is the squared output difference of the two ensembles. Here E_1, E_2 and $\triangle$ can be estimated from data (in BES-OOB, we use the out-of-bag sample). Figure 4.10 shows the pseudocode for the CE method, which has been adapted for BES-OOB pruning.

Stacking, or stacked generalisation [164], is a popular ensemble learning strategy, where the weights of the base classifiers are the regression coefficients of the meta-level regressor. Usually linear regression (LR) is used at the meta-level. Since our goal is to prune an ensemble, simply using LR would not

The BES-OOB-CE Algorithm

Inputs:

S is the training set

E is the Ensemble Selection (ES) learner

T is the number of bootstrap samples

1: $H \leftarrow$ BES-OOB(S,E,T) // an ensemble of T ES ensembles
2: $f_1^c \leftarrow$ the ensemble in H with the smallest out-of-bag estimate of error
3: $e_{min} = +\infty$
4: for $i \leftarrow 2$ to T
5: $f_i \leftarrow null$
6: for each $f \in H$
7: $e \leftarrow$ estimated error of combing f and f_{i-1}^c
8: if $e < e_{min}$ then $f_i \leftarrow f$ and $e_{min} \leftarrow e$
9: }
10: if f_i is *null* then $f_N^c \leftarrow f_{i-1}^c$ and *break*
11: $f_i^c \leftarrow p_i f_{i-1}^c + (1 - p_i) f_i$, where p_i is obtained by Eq.2 for the mean squared error
12: }
13: return f_N^c

Figure 4.10: Pseudocode of the CE method for BES-OOB ensemble pruning.

reduce the ensemble size. Here, we propose using stacking with the NNLS algorithm. To the best of our knowledge, this is the first time that stacking with NNLS is considered as an ensemble pruning approach. Eq. 4.4 shows the basic form of the NNLS optimisation problem.

$$\min_{w \geq 0} \|Xw - y\|_2^2. \tag{4.4}$$

Here, X is the data matrix, w is the regression coefficient vector, and y is the target matrix. We can see that it is the same as the linear least-squares regression form, but with extra constraints on the values of the coefficient vector. For our experiment, we use the NNLS algorithm proposed in [94]. The BES-OOB ensemble strategy constructs an ensemble of ES ensembles. Each individual ES is trained on a corresponding bootstrap sample, and its ensemble selection is guided by its performance on the out-of-bag sample. The

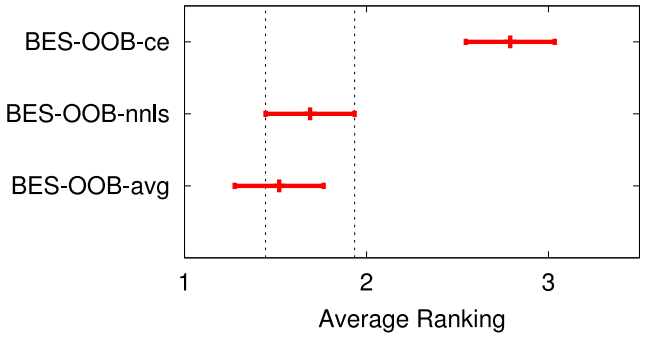


Figure 4.11: The result of the *Friedman-test* over 42 data sets with two-tailed *Bonferroni-Dunn test* at significance level 0.05. Strategies having non-overlapped bars are significantly different.

basic steps of using stacking with NNLS for BES-OOB pruning are as follows: Suppose S is the training set, and H is an ensemble of ES ensembles (same as line 1 in Figure 4.10). A meta-dataset can be constructed by using the predictions of each ES in H on S. The targets in S are used as the targets of the meta-dataset. Then, we use NNLS to build a model on the meta-dataset. The NNLS regression coefficients are used as the weights for each ES. Therefore, the final ensemble consists only of ES ensembles with greater than zero weight.

The experimental setup is as follows: the number of bagging iterations is set to 30 for all three methods (BES-OOB-avg/ce/nnls). For each bagging iteration, one REPTree-based ES learner is trained. The number of trees used for each ES is 10. As in the previous experiment, each REPTree is built using a random 33% of the original attributes. So in total 300 REPTree learners are built for each of the three methods in the comparison. At the individual bagging iteration level, all three BES-OOB methods are set to optimise the mean squared error (MSE) metric. At the pruning level, BES-OOB-ce is also set to optimise the MSE metric based on Eq. 2. Also, based on Eq. 5, we know that BES-OOB-nnls optimises square error by default. In total 42 data sets are used for this experiment.

Data set	Root Relative Squared Error			Ensemble Size			Number of Trees		
	nnls	ce	avg	nnls	ce	avg	nnls	ce	avg
autoHorse	34.60	36.87	34.78	5.7	5.6	30	22.0	22.2	118.8
autoMpg	37.53	38.48	37.58	8.7	4.6	30	38.4	21.3	138.6
autoPrice	34.99	38.52	35.91	6.2	4.3	30	23.3	17.2	114.1
baskball	84.46	85.26	83.69	4.9	4.2	30	15.0	12.5	98.2
bodyfat	14.79	○ **27.53**	17.13	5.3	3.4	30	13.3	9.7	75.2
bolts	31.39	36.12	35.50	5.4	4.5	30	14.7	14.4	91.0
boston	41.85	44.99	43.02	7.4	5.3	30	28.7	23.4	124.9
breastTumor	97.40	97.75	96.50	4.5	4.1	30	15.9	15.7	104.2
cholesterol	101.31	100.47	99.15	3.7	3.9	30	13.6	15.0	108.1
cleveland	74.15	75.01	73.78	5.9	4.0	30	22.0	17.0	119.0
cloud	44.80	47.19	44.09	5.6	5.3	30	20.3	19.5	109.5
cpu	19.46	○ **29.41**	22.10	4.8	4.1	30	21.3	16.7	115.0
detroit	151.11	283.02	151.73	3.6	19.3	30	11.5	61.9	94.2
echoMonths	72.99	71.40	70.55	3.8	3.8	30	11.7	10.9	89.3
elusage	49.91	49.52	48.76	6.0	7.2	30	16.0	21.0	83.2
fishcatch	19.84	24.40	21.24	7.5	3.7	30	26.9	13.0	106.7
gascons	25.50	30.09	25.24	5.2	9.1	30	19.5	35.8	115.2
housing	41.57	45.03	43.16	7.3	5.2	30	28.5	21.2	123.7
hungarian	74.99	75.10	74.36	4.7	5.2	30	15.3	17.8	102.2
kidney	78.38	82.08	81.29	4.6	5.2	30	12.9	16.0	91.2
longley	47.35	61.70	49.97	5.0	12.5	30	16.4	39.8	102.1
lowbwt	63.02	64.00	61.62	4.4	3.9	30	13.8	11.6	92.3
meta	149.15	147.21	112.30	1.6	3.9	30	4.8	14.3	94.2
newton-hema	85.56	85.20	82.78	3.9	5.4	30	13.2	17.4	97.1
pbc	84.45	85.07	84.24	4.9	6.1	30	18.2	24.5	116.6
pollution	71.70	77.01	72.06	5.8	4.6	30	18.8	16.2	101.5
pwLinear	• **48.92**	55.53	53.93	5.0	3.8	30	14.0	12.9	96.8
quake	100.00	99.78	99.54	3.5	6.2	30	18.0	28.3	132.6
schlvote	84.32	○ **110.16**	79.01	2.9	12.6	30	8.4	36.4	84.8
sensory	• **84.82**	86.99	86.97	5.4	5.8	30	16.7	20.7	103.7
servo	37.66	44.59	43.44	3.9	7.0	30	10.9	23.3	94.0
sleep	82.01	80.57	77.49	4.3	4.5	30	11.9	13.3	90.0
socmob	• **37.24**	41.23	39.81	6.5	4.4	30	19.9	15.7	100.0
stanford	92.16	95.82	88.37	4.1	6.9	30	12.6	21.8	91.7
strike	88.44	87.79	80.79	3.3	4.8	30	11.8	16.9	111.8
strikes	• **0.41**	8.95	4.42	1.4	10.0	30	2.9	21.9	66.0
veteran	97.46	93.98	91.51	3.3	4.3	30	10.3	13.5	94.8
vineyard	62.82	63.97	62.57	4.5	8.1	30	14.2	24.1	95.1
vinnie	• **50.89**	52.39	51.30	7.7	4.5	30	23.5	13.3	88.4
visualizing-galaxy	15.60	16.31	15.65	10.5	5.7	30	42.7	23.6	126.0
winequality-red	77.08	77.98	77.39	7.4	9.8	30	40.2	57.8	166.1
winequality-white	76.13	76.75	76.39	9.0	15.4	30	63.4	106.1	215.1
Average	63.53	69.79	62.65	5.2	6.2	30	19.0	23.2	106.7
(win/tie/loss); avg vs. nnls: 0/37/5; avg vs. ce: 3/39/0;									

• ○, BES-OOB-avg is significantly worse or better, respectively; at significance level 0.05

Table 4.5: The Root Relative Squared Error values, the ensemble sizes, and the number of trees in the final ensemble for BES-OOB-avg, BES-OOB-ce and BES-OOB-nnls, over 42 data sets.

Table 4.5 shows the *corrected paired t-test* results. The reported root relative squared errors are estimated from 10 times 10-fold cross-validation. The fi-

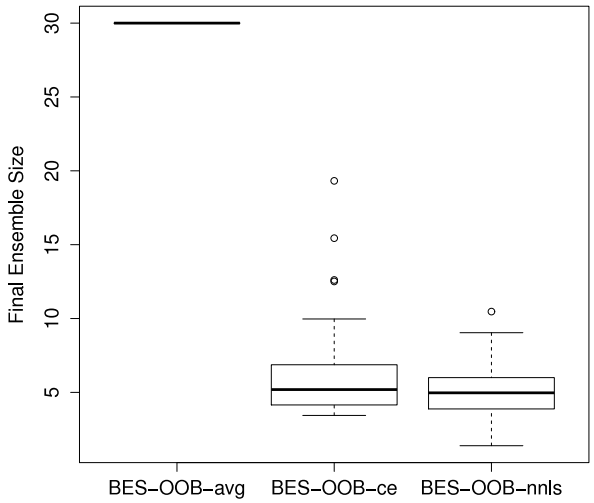


Figure 4.12: The boxplot visualization for the final average ensemble sizes.

nal ensemble sizes, and the final number of trees, are also reported. Figure 4.11 shows the *Friedman-test* results. Based on the *corrected paired t-test* results, we can see that both of the two pruning methods, BES-OOB-ce and BES-OOB-nnls, show competitive predictive performance compared to BES-OOB-avg, but with smaller final ensemble sizes and final number of trees. There are no significant *t-test*-based performance differences between BES-OOB-avg and the two pruning methods on most of the 42 data sets (39 for BES-OOB-ce; 37 for BES-OOB-nnls) in this experiment. Based on the *Friedman-test* and the *Bonferroni-test*, we can see that the performance of BES-OOB-avg and BES-OOB-nnls has no significant differences over the 42 data sets.

The final ensemble size for BES-OOB-avg is 30 (equal to the number of bagging iterations). Over the 42 data sets, the average final ensemble size of BES-OOB-ce is 6.2, corresponding to a 70% reduction in terms of ensemble size; the average final ensemble size of BES-OOB-nnls is 5.2, corresponding to a 83% reduction in terms of ensemble size. The average final number of trees of BES-OOB-avg is 106.7, which is about 36% of the total 300 trees. The average final number of trees of BES-OOB-ce is 23.2, corresponding to a 78% ((106.7 - 23.2)/106.7) reduction in terms of number of trees; the average final number of

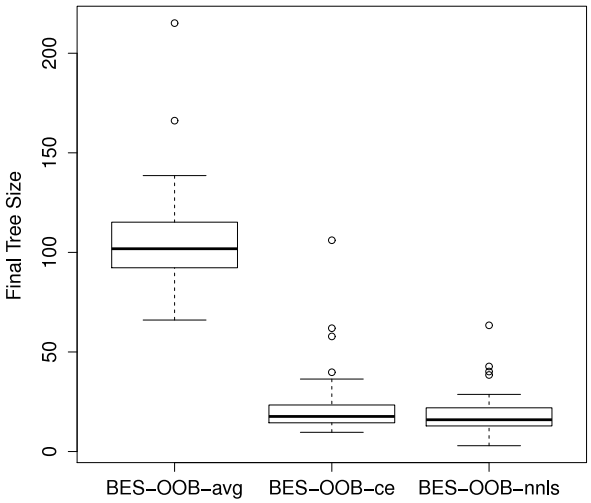
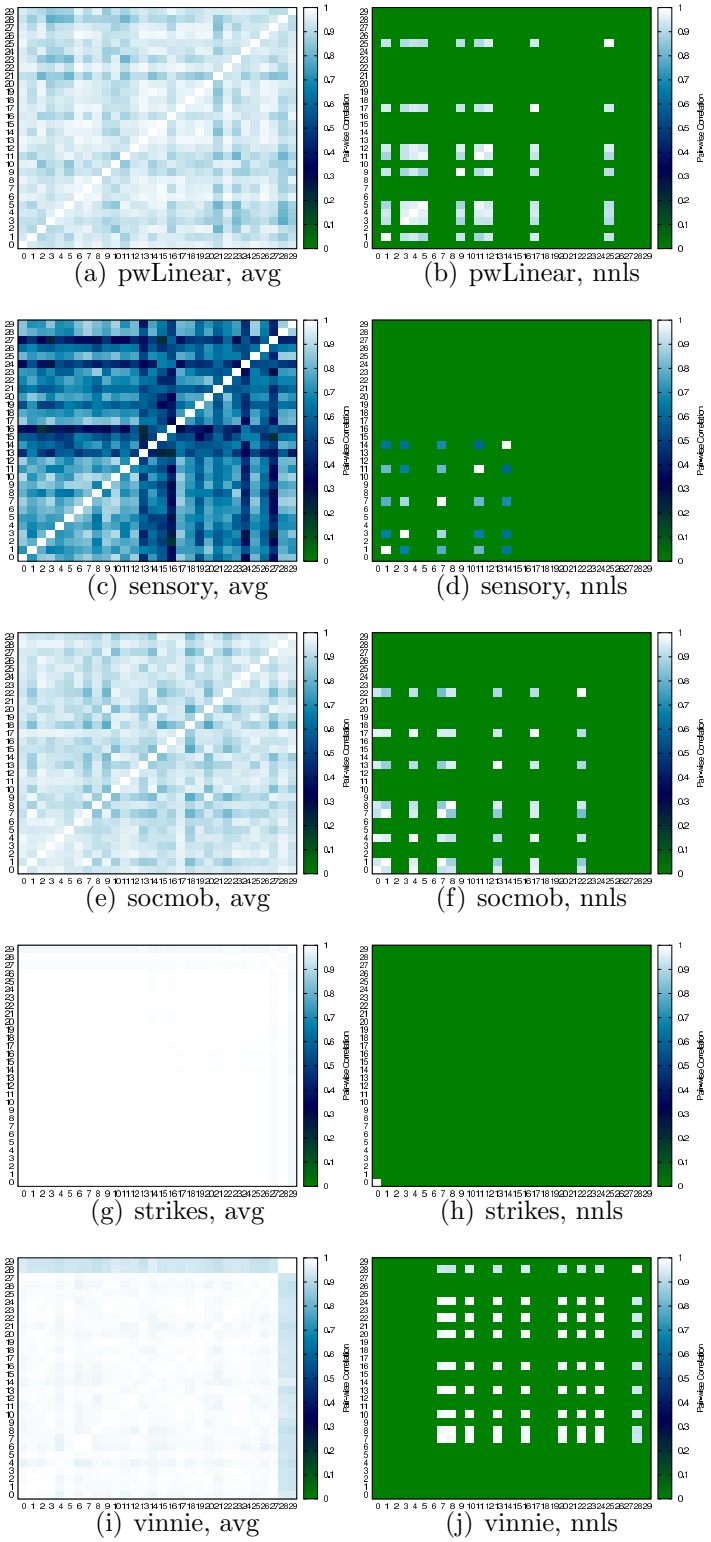


Figure 4.13: The boxplot visualization for the final average tree sizes.

trees of BES-OOB-nnls is 19.0, corresponding to a 82% reduction in terms of number of trees. Figure 4.12 and Figure 4.13 show the boxplot visualization for the ensemble sizes and the numbers of trees of BES-OOB-avg, BES-OOB-ce and BES-OOB-nnls.

Notably, the BES-OOB-nnls method significantly outperforms the BES-OOB-avg method on 5 data sets (about 12% of the 42 data sets). This is a significant empirical result indicating that BES-OOB-nnls not only works well for ensemble pruning, but also could be used for further improving the predictive performance of the BES-OOB strategy.

Figure 4.14 shows the heatmap visualizations for the final ensembles of BES-OOB-avg and BES-OOB-nnls on the 5 data sets where BES-OOB-nnls is significantly better. The x-axis and the y-axis are the indices of 30 base ES learners, sorted by out-of-bag error (lower index means lower oob error). There are in total $30 \times 30 = 900$ pixels (cells) per image. The colour of each cell is determined by the pair-wise correlation score (normalised) between the predictions of the ES at x and the predictions of the ES at y. A cell with the same index on both x and y will have a score of 1.0, and is in white. For the right hand side panels, a cell in *green* means the corresponding ES has



(a) pwLinear, avg (b) pwLinear, nnls

(c) sensory, avg (d) sensory, nnls

(e) socmob, avg (f) socmob, nnls

(g) strikes, avg (h) strikes, nnls

(i) vinnie, avg (j) vinnie, nnls

Figure 4.14: The heatmap visualizations for the final ensembles of BES-OOB-avg (left panel) and BES-OOB-nnls (right panel) on the 5 data sets where BES-OOB-nnls is significantly better.

been pruned. BES-OOB-avg and BES-OOB-nnls are trained on 90% of each data set, and the correlation scores are based on predictions of individual ES learners on the remaining 10% of each data set.

From the image processing point of view, we can see that BES-OOB-nnls works by finding a reduced image for BES-OOB-avg, which maintains the general pattern of the original image. Or it could be also thought of as BES-OOB-nnls removes noise cells from the original image. In this sense, the noise cells are the base ES learners that were determined as not contributing to improving performance. Interestingly, on the *strikes* data set, we can see that most ES learners in the BES-OOB-avg ensemble are very similar to each other (pair-wise correlation scores are high), so the BES-OOB-nnls pruning method could reduce the image to a single point. This makes sense because in an extreme case, in which all ES learners are the same, then the ensemble average should be the same as a single ES learner. Also, we can see that, on the *vinnie* data set, the BES-OOB-nnls method tends to keep the pattern for cells with relatively high diversity in the original image (please see the cells for $x = 28$ or $y = 28$). This partially explains the good performance of the BES-OOB-nnls method.

4.7 BES Regression Summary

Bagging ensemble selection using the out-of-bag sample for hillclimbing (BES-OOB), is a new ensemble learning strategy. In this section, we studied the predictive performance of BES-OOB in the regression setting. The main contributions of this section are:

- Previous studies focused on using BES-OOB for classification problems only. In the above experiment, through a series of experiments and statistical tests, we have shown that, in the regression setting, the BES-OOB strategy is competitive to MultiBoosting, and is superior to Bagging and

Stochastic Gradient Boosting when using CART-like regression trees as the base learners.

- We have shown that using a diverse model library could further boost BES-OOB's predictive performance when the model library size is relatively large.

- Our results also have shown that both the cocktail ensemble and the stacking with NNLS methods work well for BES-OOB ensemble pruning. Particularly, the latter method can be also used to improve the predictive performance of the BES-OOB strategy.

One reason for the good predictive performance of the BES-OOB strategy is that it can optimise a user-specified error metric directly in the base learner selection stage. Out-of-bag samples seem to work well for ES's ensemble selection in practice. Another notable feature of BES-OOB is its simplicity and ease of implementation.

The success of the BES-OOB ensemble strategy over a broad range of classification and regression problems examined in the previous sections strongly suggests the applicability of the method to a wide range of problems. In the next section, we apply the BES-OOB strategy to the meta-learning-based algorithm and parameter ranking problems.

4.8 Theoretical Aspect of the BES Strategy

Although in recent years, the ES algorithm has been highlighted in winning solutions of many data mining competitions [144], there is no theoretical work on explicitly examining the convergence property of the forward stepwise procedure used in the algorithm. Here we attempt to give a brief discussion on the theoretical aspect.

Here we employ the notations used in [29] in the context of high-dimensional variable selection. For simplicity, we use the regression setting as an example. Consider a general linear model with univariate response Y and m-dimensional covariates $X \in \chi \subseteq \mathbb{R}^m$: $Y_1, ...Y_n$ independent $g(\mathbb{E}[Y_i|X_i = z]) = \mu + \sum_{j=1}^{m} \beta_j z^{(j)}$, where $g(.)$ is a real-valued link function, μ denotes the intercept, $z = \{x_1, x_2, ..., x_m\}$ (where x_v are the predictions from the m base models) and the covariates X_i. For simplicity we absorb μ into the right hand side, and then an ES model can be expressed by:

$$f(x) = f_{\mu,\beta}(x) = \sum_{j=1}^{m} \beta_j z^{(j)}.$$

In the ES model, we have an (empirical) loss function $l(\beta) = \frac{1}{n} \sum_{i=1}^{n} l_{\beta(X_i,Y_i)}$ where the value of the loss is estimated from a hillclimb set. In every iteration $k = 1, 2, ...$, we have an active set of variables (base model predictions) H:

$$H^k \subseteq \{1, ..., p\}.$$

Now, the problem is to look for an additional variable reducing the empirical loss most when adding it to the previous active set (ensemble). For a subset $H^{k-1} \subseteq \{1, ..., p\}, \beta_H \in \mathbb{R}^m$ is defined as:

$$\beta_{j,H} = \begin{cases} \beta_j, & j \in H, \\ 0, & j \notin H. \end{cases}$$

Therefore the ES procedure estimates the weights corresponding to H by:

$$\hat{\beta} = \underset{\beta_H}{\operatorname{argmin}}\ l(\beta_H),$$

where the minimization is done over the components corresponding to H. The forward-stepwise procedure of ES searches in every iteration k for the best single variable (base model) with index $\hat{j_m}$, and adds it to the previous active set to reduce the empirical loss as much as possible:

$$\hat{j_k} = \underset{j \in \{1,...,p\} H^{[k-1]}}{\operatorname{argmin}}\ l(\hat{\beta}_{H^{[k-1]} \cup \{j\}}),$$

and the new ensemble (active set) is:

$$H^{[k]} = H^{[k-1]} \cup \{\hat{j}_k\}.$$

Putting this in words, we see the forward step-wise method used in ES as a "greedy" (active set) algorithm as we have described above, and the predictions of each base regression model can be seen as the "feature" values.

When certain conditions are met, such as sufficient conditions for convex optimization, use of squared error loss and in the absence of noise, the theoretical convergence rate of the forward step-wise method used in ES is sublinear in m. [29] proves it to be $m^{-\frac{1}{2}}$. The BES strategy can be seen as running multiple ES algorithms in parallel, therefore, the theoretical convergence rate of BES is the same as the ES algorithm.

4.9 ES and BES-OOB for Meta-learning

In this section, we introduce the idea of applying the ES and the BES ensemble strategies for the meta-learning-based ranking problem. The simple forward model selection based ES for ranking works as follows: (1) start with an empty ensemble; (2) add to the ensemble the ranker in the library that maximizes the ensembles performance to the ranking error metric on a hillclimb set; (3) repeat Step 2 until all ranking models have been examined; (4) return that subset of rankers that yields maximum performance on the hillclimb set. To distinguish between ES for classification, regression and ranking, we call the above strategy **ESMetal** (Algorithm 4.2), stands for ES for meta-learning.

The basic idea of applying BES-OOB to ranking problems is relatively straightforward. Algorithm 4.3 shows the pseudocode. In each bagging iteration, firstly a library of base-level meta-learners (rankers) are trained based on the corresponding bootstrap sample. Then, an ensemble of base-label rankers is constructed using ESMetal. For simplicity, we call the algorithm

Algorithm 4.2 The ESMetal Algorithm for Ranking

Input Training set S_t

Input Hillclimb set S_h

Input The ESMetal model library L

Input Ranking performance metric M

train base-level rankers in L on S_t

E = do ensemble selection based on base rankers' performance M on S_h, e.g., forward stepwise ensemble selection

return E

BESMetal, stands for BES for meta-learning.

4.9.1 Case Study 1: Algorithm Ranking

In this experiment we use the algorithm ranking dataset which has been used in Section 3. The dataset has 466 instances, 80 numeric features and 20 target labels (corresponding to 20 algorithms). Figure 3.1 in Section 3 shows the format of the dataset.

We compare the ranking performances of three algorithms under 8 ranking evaluation metrics and functions. The performance value of each algorithm is estimated using 10 runs of 90% vs. 10% training/testing split evaluation. All three algorithms in comparison use 1000 base-level rankers. The detailed parameter settings are as follows:

ART forests—We use 1000 ARTs, and the minimum number of instances γ at a leaf node is set to 1 for each ART; The "pruning" parameter θ is set to 0.95.

BESMetal-a—We set the number of bagging iterations to 10, and use 100 ARTs for the ESMetal algorithm in each bag. The minimum number of instances γ at a leaf node is set to 1 for each ART; The "pruning" parameter

Algorithm 4.3 The BESMetal Algorithm for Ranking

Input Training set S

Input The ESMetal algorithm E

Input number of bootstrap samples T

Input performance metric M

for $i = 1 \rightarrow T$ **do**

S_b = get a bootstrap sample from S

S_{oob} = out of bag sample

train base-level rankers in the model library of E on S_b

E_i = do ensemble selection based on base rankers' performance M on S_{oob}

end for

θ is set to 0.95. Here the algorithm named BESMetal-a, stands for BES for meta-learning using ART as the base learners. BESMetal-a is set to optimize the target metric directly.

BESMetal-d—We set the number of bagging iterations to 10, and use 100 different rankers for the ESMetal algorithm in each bag. The 100 rankers include:

- 1 default ranker
- 29 k-NN rankers with different k values ($k \in \{1, ..., 29\}$);
- 20 ART rankers with different parameter settings (e.g., number of instances at a leaf, and the maximum R^2 value);
- 20 binary pairwise comparison (BPC) rankers with different parameter settings (logistic regression is used as the base learner, with different ridge values);
- 20 PCTR (predictive clustering tree for ranking) ranker with different parameter settings (e.g., number of instances at a leaf $\in \{1, ..., 20\}$);

Metric	**ART forests**	**BESMetal-a**	**BESMetal-d**	**DefRanker**
SRCC	0.613 ± 0.01	0.615 ± 0.01 •	0.620 ± 0.01 •	0.472 ± 0.03 ∘
WRC	0.597 ± 0.02	0.604 ± 0.01 •	0.611 ± 0.01 •	0.460 ± 0.03 ∘
LA@1	0.229 ± 0.06	0.228 ± 0.04	0.224 ± 0.05 ∘	0.135 ± 0.05 ∘
LA@3	0.491 ± 0.08	0.501 ± 0.06 •	0.485 ± 0.06 ∘	0.394 ± 0.07 ∘
LA@5	0.629 ± 0.09	0.625 ± 0.08 ∘	0.630 ± 0.06	0.504 ± 0.06 ∘
NDCG@1	0.422 ± 0.06	0.429 ± 0.05 •	0.432 ± 0.04 •	0.330 ± 0.04 ∘
NDCG@3	0.422 ± 0.02	0.428 ± 0.02 •	0.435 ± 0.02 •	0.341 ± 0.02 ∘
NDCG@5	0.488 ± 0.02	0.491 ± 0.03 •	0.493 ± 0.02 •	0.404 ± 0.02 ∘

Table 4.6: A comparison of ranking performances of the three algorithms on the algorithm ranking problem. • ∘, ART forests is significantly worse or better, respectively, based on paired t-test at significance level 0.05.

- 10 AdaRank ranker with different parameter settings ("weak (depth=3)" ART is used as the base learner for AdaRank; number of iterations of AdaRank is set to 30).

Here the algorithm name BESMetal-d, stands for BES for meta-learning using a diverse base-level meta-learners (rankers) library. BESMetal-d is set to optimize the target metric directly.

Table 4.6 shows the comparison results of ranking performances on the algorithm ranking problem. The performances of the default ranker (please see Section 3 for details) is also included as a baseline. The BESMetal-a algorithm significantly outperforms the ART forests algorithm on 6 out of 8 metrics, indicating that the BES strategy (BESMetal-a) is more likely to find a better ensemble than the simple averaging method used in ART forests. The BESMetal-d algorithm is significantly better than the ART forests algorithm on 5 out of 8 metrics. However, on the LA@1 and LA@3 functions, the BESMetal-d algorithm is significantly worse, which seems to suggest that BES

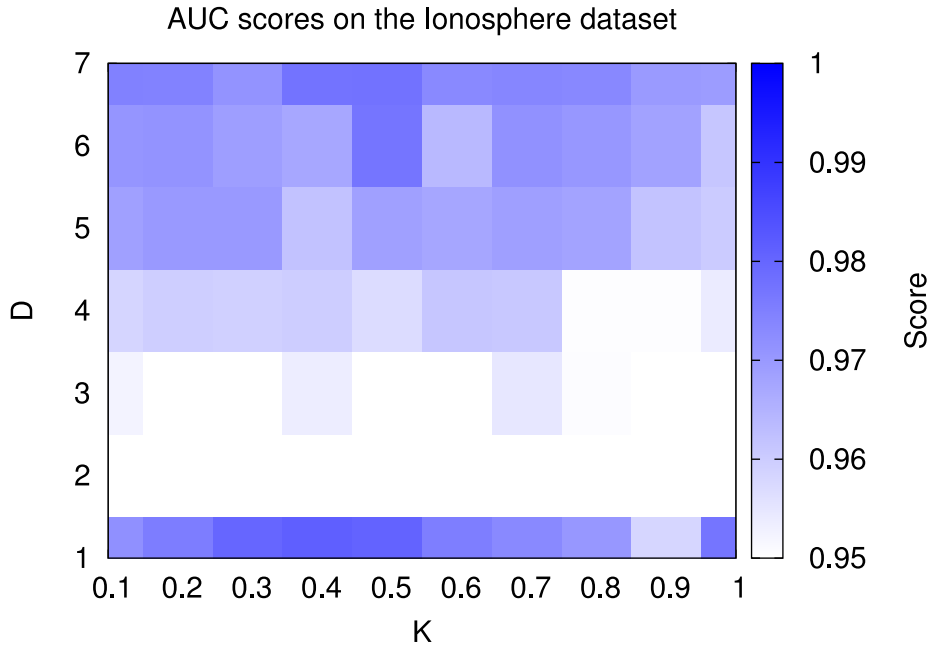


Figure 4.15: Heat map of the AUC space based on random forests models.

with a diverse model library is not stable on non-smooth metric functions. However, this does not necessarily imply BESMetal-a is a better algorithm compared with BESMetal-d since the model library used by BESMetal-d in this experiment is relatively small (only 100 base rankers). In future work, we plan to do an empirical comparison between BESMetal-a and BESMetal-d with different settings on multiple datasets.

4.9.2 Case Study 2: Parameter Ranking

In this case study, we examine the predictive performances of ART forests, BESMetal-a and BESMetal-d in the context of parameter ranking for the random forests algorithm. Similar experiments can be done for other machine learning algorithms.

Random forests [25] is a popular off-the-shelf ensemble learning algorithm. Not only does it produces accurate models for real-world problems, but it also features a relatively small number of parameters.

The random forests[6] algorithm implemented in WEKA is used in the following experiment. We use meta-learning to rank the combinations of two parameters, namely, the percentage of features to be used as random features, K; and the depth of a random tree, D. Figure 4.15 shows an example of the 3-fold CV based AUC space of random forests (with 30 trees) on a binary classification dataset. We can see that although the overall AUC scores are already very good (all greater than 0.95), there are clearly multiple "hot zones (highest AUC scores)" in the heat map, indicating that some parameter pairs can produce "better than average" models. For application domains that require "as accurate as possible" models, parameter selection is essential. Meta-learning goes one step further by providing algorithm/parameter ranking instantly.

Next, we describe how the meta-dataset for this experiment is generated. The following parameters are used:

$$K \in \{0.1, 0.2, 0.3, ..., 1.0\} \text{ and } D \in \{1, 2, 3, ..., 7\}.$$

For instance, $K = 0.1$ means random 10% of all features are used for finding the best feature to split for an inner node; $D = 5$ means a base-level tree will stop growing if its depth exceeds 5. The above parameters can be transformed to $10 \times 7 = 70$ parameter pairs:

Pair 1: $K = 0.1$ and $D = 1$

Pair 2: $K = 0.1$ and $D = 2$

...

Pair 69: $K = 1.0$ and $D = 6$

Pair 70: $K = 1.0$ and $D = 7$

So in this case, each instance of the meta-dataset consists of two parts: the meta-feature vector (80 SIL meta-features) and a rank vector corresponding to 70 parameter pairs, where the rank vector has 70 rank values, $r \in \{1, 2, 3, ..., 70\}$. The rank values are generated based on 10-fold CV based

[6]The weka.classfiers.trees.RandomForest class

Metric	ART forests	BESMetal-a	BESMetal-d	DefRanker
SRCC	0.484 ± 0.04	0.489 ± 0.03 •	0.491 ± 0.04 •	0.390 ± 0.04 ○
WRC	0.446 ± 0.04	0.447 ± 0.04	0.449 ± 0.06 •	0.310 ± 0.04 ○
LA@1	0.054 ± 0.03	0.053 ± 0.04	0.050 ± 0.04 ○	0.008 ± 0.01 ○
LA@3	0.135 ± 0.03	0.131 ± 0.03 ○	0.133 ± 0.05 ○	0.058 ± 0.03 ○
LA@5	0.167 ± 0.04	0.173 ± 0.05 •	0.165 ± 0.05 ○	0.070 ± 0.03 ○
LA@10	0.259 ± 0.07	0.260 ± 0.05 •	0.260 ± 0.06	0.208 ± 0.04 ○
NDCG@1	0.136 ± 0.05	0.137 ± 0.03 •	0.139 ± 0.04 •	0.086 ± 0.02 ○
NDCG@3	0.162 ± 0.03	0.165 ± 0.02 •	0.159 ± 0.04 ○	0.119 ± 0.01 ○
NDCG@5	0.199 ± 0.02	0.201 ± 0.03	0.192 ± 0.05 ○	0.150 ± 0.01 ○
NDCG@10	0.280 ± 0.02	0.288 ± 0.02 •	0.283 ± 0.04 •	0.231 ± 0.01 ○

Table 4.7: A comparison of ranking performances of the three algorithms on the algorithm ranking problem. • ○, ART forests is significantly worse or better, respectively, based on paired t-test at significance level 0.05.

AUC score of each parameter pair. For each individual parameter pair, we have 500 random trees as the base-level models. One reason for using AUC scores to generate rankings is that the number of ties is negligible. The final meta-dataset in this experiment consists of 466 instances (because we have 466 datasets). Each instance has 80 meta-feature values and 70 rank values.

We use the same three algorithms and experimental settings described in Case Study 1, except for two additional metrics: LA@10 and NDCG@10 are added due to the dataset having much more labels than the algorithm ranking problem. The performance of the default ranker is also included as a baseline. Table 4.7 shows the comparison results of ranking performances on the parameter ranking dataset (for random forests). The BESMetal-a algorithm significantly outperforms the ART forests algorithm on 6 out of 10 metrics. There are 3 ties between these two algorithms. The result indicates that the

BES strategy (BESMetal-a) is more likely to find a better ensemble than the simple averaging method used in ART forests. The BESMetal-d algorithm is significantly better than the ART forests algorithm on 4 out of 10 metrics, but significantly worse on 5 out of 10 metrics. The result seems to suggest again that BES with a diverse model library (BESMetal-d) is unstable, and is more likely to overfit than simple averaging. In future research, we will attempt to improve this issue.

4.10 Conclusions

In this chapter, we proposed a new ensemble learning algorithm—Bagging Ensemble Selection (BES) for classification, regression and meta-learning-based ranking with extensive empirical evaluation and preliminary theoretical analysis.

For the meta-learning-based algorithm/parameter ranking problems, our experimental results indicate that the BES strategy based BESMetal-a algorithm is more likely to produce a better ensemble than the ART forests algorithm. One reason could be that BESMetal-a can be set to optimize the target metric directly in the base-level ensemble construction stage.

Yet our studies of the BESMetal-d algorithm (with a diverse ranking model library) have shown mixed results. One reason could be due to that, compared to our experiments in the regression setting, the number of instances in the meta-datasets for algorithm/parameter ranking is too small (only 466 examples), which led to a diverse ranking model library becoming to be more likely to overfit.

Chapter 5

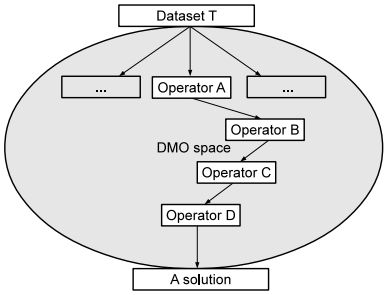
Meta-Heuristic Search Based Full Model Selection

In this chapter[1], we propose a framework, in the DMO space setting, which can be used for designing new Meta-heuristic search based full model selection, or FMS algorithms, and a novel FMS algorithm which can be seen as a realization and an application of this framework.

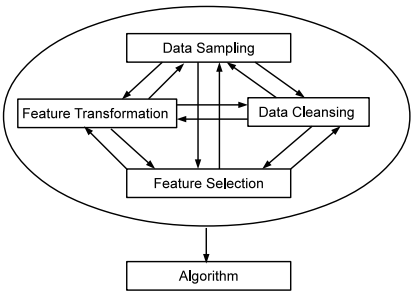
5.1 Motivations

Machine learning users now have to face the new problem of how to choose a combination of data processing tools and algorithms. The goal is usually defined as maximizing or minimizing a quantitative measure. In classification problems, the goal could be optimising the classification accuracy, the Lift score or the ROC area (AUC); in regression problems the goal could be optimising RMSE (root mean squared error), MAE (mean absolute error), or any other proper loss function.

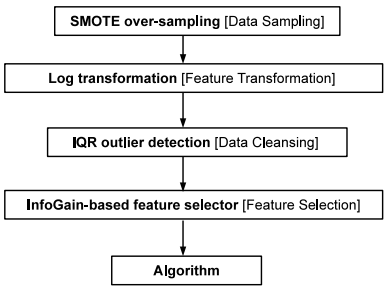
[1]Part of the research presented in this chapter has appeared in [147, 148]



(a) An illustration of the DMO space



(b) A graphical representation of the DMO template used by GPS



(c) A graphical representation of a DMO solution template instance

Figure 5.1: A full model defined by the GPS algorithm

Sometimes the final goal might be a combination of multiple goals. Traditionally, these problems are addressed separately in the feature selection, model or parameter selection and the meta-learning fields. A practical data mining problem consists of many sub-problems which presents an extremely large search space that could be a very time-consuming task for humans to explore manually. Therefore, strategies and methods that can help people to

choose, or that could automatically suggest, an optimised data mining solution are useful. In this chapter, we propose a framework which can be used for designing new FMS algorithms, and we also present a novel FMS algorithm which is a realization and an application of the proposed framework.

5.2 The DMO Space

We first define the DMO space and discuss potential approaches for searching the space. We here attempt to define a search space that consists of all data mining actions (operators) that are available to a given dataset for a user-specified goal, such as a set of outlier filters, a set of feature selection methods, a set of data transformation techniques and a set of base learning algorithms. In this sense, we call the subject of interest "the space of data mining operators (DMO)", or simply "the DMO space" [147].

In this search space, a data mining solution is abstracted as a directed acyclic graph (DAG) consisting of DMOs that are connected based on some relations: see Figure 5.1 (a) for an illustration. For simplicity, in Figure 5.1 (a) we consider that an optimal data mining solution is given by a DAG defined by four DMOs (A, B, C and D) for dataset T. The DMO space is represented by the largest oval, which consists of all DMOs applicable to T. The directed arrows represent the relationships (action rules) in the DAG. If Operator A is an outlier filter, Operator B is a feature reduction method, Operator C is a decision tree algorithm, and Operator D is a post-processing method, the DAG can be interpreted as follows: given a dataset T, in an optimal solution we first use the outlier detection method (DMO A) to remove outliers, and then we employ the feature selection method (DMO B) to remove useless features, and then build a decision tree model (DMO C), and finally, we use a probability calibration method (DMO D) to calculate the model outputs. This is a very large search space because in theory there exists an arbitrary number

of DMOs (including an arbitrary number of link directions, node orders and arrangements). Therefore, the next question is how to search in this space?

5.3 DMO Objects

In practice, due to the limited resources at hand, usually we do not search in an infinite DMO space, and, moreover, we can make the DMO space a finite space by defining the DMOs that are to be included. For example, given a dataset T, and given we have one outlier detection algorithm, two feature selection methods, three classification algorithms and that the goal is to build a model that gives the lowest classification error on T, typically, we can define the following *node* type DMO objects:

$DMO_{filter}, DMO_{no-filter}, DMO_{feature-selection-1}$,

$DMO_{feature-selection-2}$, $DMO_{no-feature-selection}$,

$DMO_{algorithm-1}, DMO_{algorithm-2}, DMO_{algorithm-3}$.

For simplicity, we rewrite the above DMOs to:

filter, no-filter, feature-selection-1,

feature-selection-2,no-feature-selection,

algorithm-1, algorithm-2, algorithm-3.

Given these DMOs, if we want to preprocess the data, we can define some *function* type DMOs that output a new data object. For example:

$data \Longleftarrow$ preprocessing-1(filter, feature-selection-1)

$data \Longleftarrow$ preprocessing-2(no-filter, feature-selection-1)

$data \Longleftarrow$ preprocessing-3(filter, feature-selection-2)

...

where preprocessing-1,2,3 are *function* type DMOs. We can also define more

complex *function* type DMOs which take *function* and *node* type DMOs as inputs and output a solution. For example:

solution $\Longleftarrow$ build-model(preprocessing-1, algorithm-1)

solution $\Longleftarrow$ build-model(preprocessing-2, algorithm-2)

solution $\Longleftarrow$ build-model(preprocessing-3, algorithm-1)

...

where build-model and preprocessing-1,2,3 are all *function* type DMOs. In this way, we are free to define which, and what kind of, DMOs are to be added to the DMO space.

To meet the data mining goal, we could simply search all the DMO function-object relations (paths) in the space. Therefore, the solution which has the lowest classification or regression error could be the output of a grid-search-like exhaustive search. One advantage of an exhaustive search in a finite DMO space is that the user controls the search complexity. Another advantage is that the DMO relations can be designed by a data mining expert and then shared and reused. For example, if an expert designed a good DMO search space for an unbalanced binary classification problem, she can probably share it with her colleagues or reuse it for a new project.

However, the disadvantage is also obvious because the search complexity grows dramatically as the number of DMOs increases, with the result that if the search space is too large, due to computational and time costs the user may have to terminate the search before all DMOs are explored. To overcome this problem, we may need to think about questions such as how promising DMOs can be automatically defined/generated for a given dataset.

In the previous examples, we have defined some DMOs by hand. One could generate DMOs simply by generating all possible DMO combinations of different types, but doing so would create an extremely large (even infinite)

search space, and the problem becomes intractable. We here propose a semi-automatic method to solve this problem.

Firstly, we define some DMO functions, and add these functions to the DMO search space as we did on previous page. Secondly, we define some templates (rules) for searching. Here are two examples:

solution $\Longleftarrow$

chain-search([filter], [feature-selection], [tree-model]) (1)

solution $\Longleftarrow$

chain-search(random-topology-search([filter], [feature-selection]), [tree-model]) (2)

Template (1) is a chain solution. Here a chain solution means the order (such as from left to right) of each DMO does matter. A "[...]" is a placeholder for a certain type of DMO object: in this example, the [filter] placeholder can be substituted by any filter-type DMO. The [feature-selection] placeholder follows the same rule, and the [tree-model] placeholder can be substituted only by a "tree" type model. In template (2), we can see a new DMO function called "random-topology-search", which means that the order of the DMOs will be changed automatically during the search. So we can see that template (1) is actually a specialisation of template (2). Once we have a set of DMO objects added, and a DMO template defined, then we have a finite DMO space.

Please note that the DMO definition supports recursive application of operations. For example:

chain-search(A, B, C) $\Longleftarrow$

RETURN chain-search(A, B, C) **IF** runtime(solution) $<$ 1 hour **ELSE RETURN** chain-search(REMOVE-BASED-ON-RUNTIME(A, B, C))

So far, we have defined a DMO space that consists of *node* type DMOs, *function* type DMOs and DMO templates. In the template part of the search space, we will have to make a decision on what kind of search strategy to use when searching for substitution DMOs for placeholders. We here consider only cases where an exhaustive search (in the case of too many DMOs permutations) is not feasible, and we are particularly interested in a search method that optimises a problem by iteratively trying to improve a candidate DMO with regard to a given measure of quality (the goal metric). These methods are usually referred to as a "heuristic search", such as the best-first search, the local search (using neighborhood relation) and the population-based evolutionary algorithms.

5.4 Related Work

The PSMS system proposed in [51], is an application of Particle Swarm Optimization (PSO) to the problem of full model selection for binary classification problems. In total, 3 feature transformation objects, 13 feature selection objects and 10 classifier objects are used in the PSMS system. A full PSMS model is defined as a 16-dimensional particle position. For the details of the PSMS system, we refer the reader to [51]. Based on the experimental results in [51], the PSMS system shows promising results when it is compared with the Pattern Search (PS) strategy [112] for the FMS problem. The system also showed competitive performance compared with other search strategies in a model selection competition.

From the system architecture point of view, PSMS assumes that a full model has three components: feature transformation, feature selection, and learning algorithm. In the DMO framework, we can define the following DMO template for the search space covered by the PSMS system:

solution $\Longleftarrow$ chain-search(

random-topology-search([feature-transformation], [feature-selection]), [algorithm])

We can see that the search space covered by the above DMO template is a simplified presentation of a full model, because a full model may have other components, such as data cleansing and data sampling. Extended from our previous work [147], in the next section, we introduce a novel search strategy for the FMS problem, which covers five data mining components, namely, data cleansing, data sampling, feature transformation, feature selection and algorithm DMOs.

5.5 The GPS Search Strategy

In this section, we propose a novel algorithm for searching for a FMS solution in the DMO space. The algorithm combines both genetic algorithm (GA) [77] and particle swarm optimization (PSO) [87]. GA is used for searching the optimal template instance of a DMO template, and PSO is used for searching the optimal parameter set for a particular template instance. The motivation is that GA is usually considered a good strategy for combinational optimization problems, whereas PSO is usually considered good at numerical optimization.

The proposed algorithm is named as GPS (**G**A-**P**SO FM**S**). It can be seen as a realization and an application of the DMO framework. Before introducing the GPS algorithm, we first define a DMO template. Here, we assume a FMS solution consists of five DMOs:

[data-cleansing],

[data-sampling],

[feature-transformation],

[feature-selection], and

[algorithm] .

Then, a DMO template for the FMS problem covered by GPS is defined as:

solution $\Longleftarrow$

chain-search(

random-topology-search(

[data-cleansing], [data-sampling], [feature-transformation], [feature-selection]),

[algorithm]) (3)

Graphically, this template can be represented as Figure 5.1 (b). The four DMOs at the top can be performed in any order, then followed by an *Algorithm* DMO. Figure 5.1 (c) shows a solution instance of the DMO template, which can be interpreted as: given a dataset, we firstly apply the data sampling technique, SMOTE [39], followed by applying log-transformation, then, we do IQR outlier detection, and then use information gain based feature selection; finally, an AdaBoost.M1 [55] model is built based on the transformed data. We call such a solution a "DMO solution template instance", shortened to "template instance".

For each of the five DMOs we have defined in template (3), we have a pool of data mining tools available. For this research, the filters and algorithms in the WEKA [74] machine learning package are used. Table 5.1 shows the tools that are included in the GPS system.

Algorithm 5.1 shows the pseudocode of the GPS algorithm. The basic steps of the system are: firstly a initial population of DMO template instances is randomly generated based on a predefined template (e.g., template (3) and Figure 5.1 (b)), and the placeholders of each template instance are randomly populated with the objects in the pools of DMOs (e.g., Figure 5.1 (c)).

Then for each GA iteration (generation), PSO is used for searching for optimal parameters for each template instance (similar to the PSMS system). The population of template instances is then sorted by their PSO-based evaluation scores. After the PSO optimization procedures are done, typical GA operators, such as crossover and mutation, can be applied for generating new

Algorithm 5.1 Pseudocode of the GPS strategy for searching a FMS solution

Inputs T (number of generations for GA), P (population size for GA), M (number of evolutions for PSO), W (swarm size for PSO), G (goal metric)

Get P random template instances based on template (3)

Populate template instances with objects in the DMO pools (Table 5.1)

for $i = 1 \rightarrow T$ **do**

Use a standard PSO procedure **PSO**(M,W,G,I) to search for the optimal parameters for each template instance I (optimising the goal metric G), and assign an evaluation score to each template instance I. This procedure is similar to the PSMS system [51].

Do *crossover* // single point crossover among the top 20% template instances

Do *mutation* // randomly choose 30% template instances from the population, and randomly change one DMO in each template instance

Replace the worst N template instances with the N new template instances generated in the above two steps, here we use $N = (20\%+30\%) \times P$

$solution_{best} \leftarrow population_{best}$

end for

return $solution_{best}$

template instances which are used for replacing the template instances with relatively low evaluation scores. The above procedure is repeated T times, where T is the number of GA generations. Finally, the template instance with the best evaluation score is returned as the GPS solution.

5.6 Comparing GPS to PSMS and Other Learning Systems

We experiment with ten classification problems. All of them are real-world datasets which can be downloaded from the UCI repository, the UCSD data mining contest repository and the KDD Cup repository. These data sets were selected because they are large and come from different research and industrial areas.

To speed up the experiments, all five multi-class datasets were converted to binary problems by retaining only the two largest classes from each. After this conversion to binary problems, for datasets that are larger than 10,000 instances, a subset of 10,000 instances is randomly selected for experiments. Table 4.1 shows the basic properties of the original and the final datasets.

To test the performance of the GPS algorithm, we implemented a variant[2] of the PSMS system proposed in [51] with the DMO pools defined in Table 5.1.

The two systems are set to optimise the AUC performance[3] and are tested under 30 configurations (3 experiments per dataset): for GPS, the population size for GA and the swarm size for PSO are both set to 10, and the number of PSO evolutions is set to 10; for PSMS, the swarm size is set to 10.

For each dataset, three experiments were conducted. Let g be the number of GA generations for GPS; when g=10, the number of PSO evolutions for PSMS is set to 1000; when g=20, the number of PSO evolutions for PSMS is set to 2000; when g=30, the number of PSO evolutions for PSMS is set to 3000. So, for each experiment, the training cost for both systems is roughly the same.

[2]In our implementation, the dimensionality of each particle is adapted automatically based on the number of parameters of a particular DMO

[3]The balanced error rate (BER) was used in the original PSMS system

The objective functions of both GPS and PSMS are based on the respective training set AUC performance obtained from 3-fold cross validation of a particular template instance. The AUC performance of two popular ensemble learning algorithms, AdaBoost.M1 [55] with 1,000 decision stumps, and Random Forest [25] with 1,000 unpruned random trees are also reported as baseline performance.

Figure 5.2 (a) to Figure 5.2 (j) show the comparison results based on the AUC performance obtained from 5 times 3-fold cross validation.

Figure 5.2 (k) gives a summary in terms of number of wins. Overall, on the 10 datasets, the GPS algorithm wins 83% (25 wins) of the 30 experiments.

The results demonstrate the benefit of combining GA and PSO for the FMS problem. Also, we can see that the best performance of both GPS and PSMS outperform AdaBoost.M1 and Random Forest on 9 out of the 10 datasets, which indicates the advantage of a full model over the single algorithm model. Another interesting pattern is that the GPS algorithm outperforms the baseline algorithms with big margin on datasets with a relatively imbalanced class distribution.

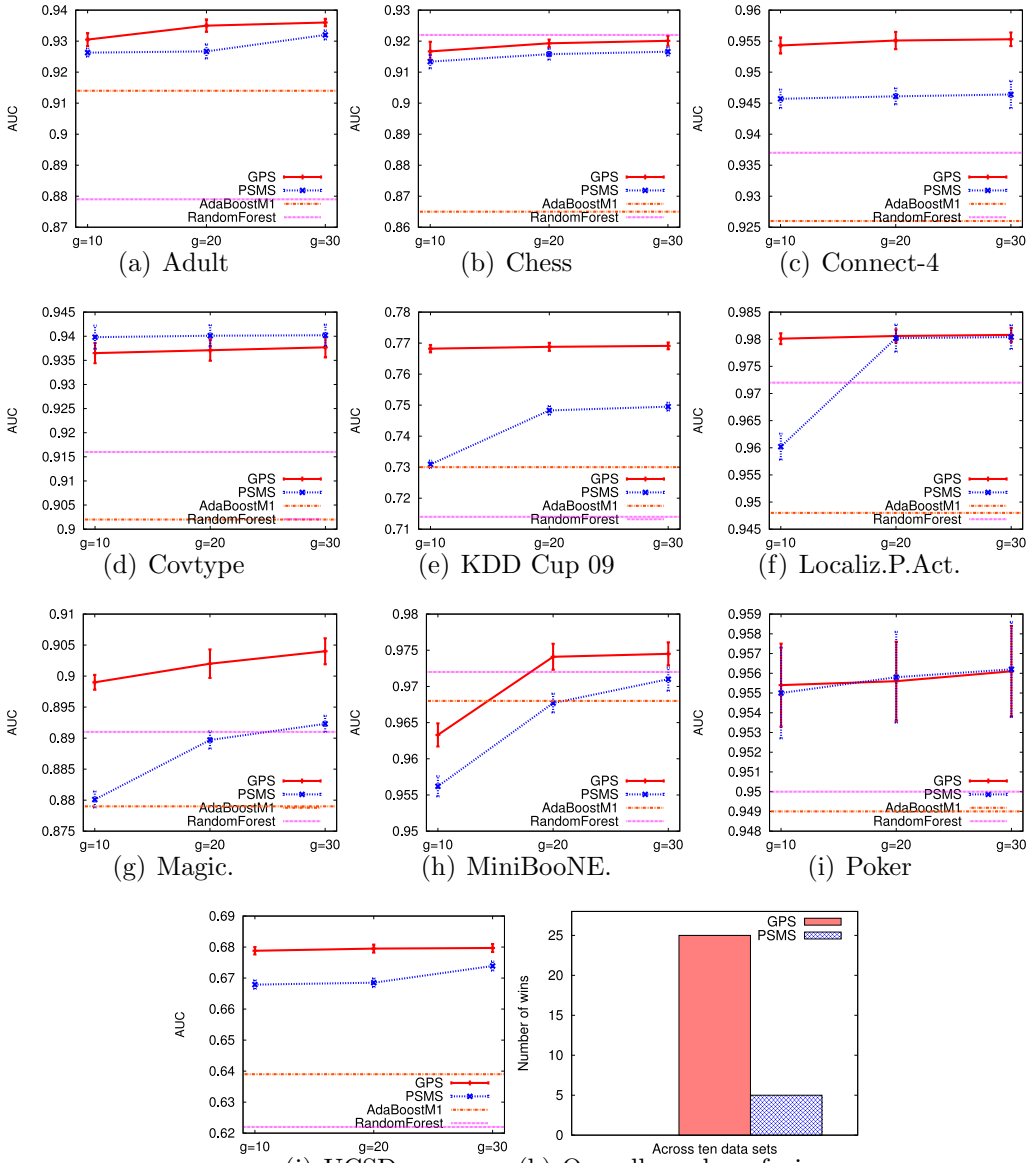


Figure 5.2: A comparison of AUC performance between GPS and PSMS under 30 different configurations; the number of PSO evolutions for GPS is set to 10; x-axis g is the number of GA generations for GPS; when g=10, the number of PSO evolutions for PSMS is set to 1000; when g=20, the number of PSO evolutions for PSMS is set to 2000; when g=30, the number of PSO evolutions for PSMS is set to 3000

Data Sampling	Data Cleansing	Feature Trans.	Feature Sel.
SMOTE oversampling	NumericCleaner	Normalize	CfsSubsetEval
Resample with replacement	RemoveUseless	Standardize	InfoGainAttributeEval
Resample no replacement	ReplaceMissingValues	Center	GainRatioAttributeEval
Do nothing	Do nothing	AddNoise	OneRAttributeEval
		Discretize	PrincipalComponents
		NominalToBinary	ChiSquaredAtt.Eval
		NumericTransform	Do nothing
		Do nothing	

Algorithm	HyperParameters
Bagging with Random Tree	num.Bagging.Iterations, num.Atts., depth.Tree
Bagging with REPTree	num.Bagging.Iterations, num.Folds., depth.Tree
AdaBoost.M1 with DecisionStump	num.Boosting.Iterations , useResample
LogitBoost with DecisionStump	num.Boosting.Iterations , useResample
Bagging with J48 Decision Tree	num.Bagging.Iterations , prune , conf.
RotationForest with REPTree	num.Iterations, Percentage.removed, projection

Table 5.1: WEKA algorithms and filters that are used as the DMO objects

5.7 Speeding Up the GPS System

The training complexity of the GPS algorithm depends on the base learners found and evaluated during the search. The main cost for GPS is the cost for estimating a base learner's performance (e.g., cross validation). The algorithm searches for a full model consisting of many data mining operators. Therefore, although GPS is powerful in modeling, the user may have to wait for several hours, or even days on relatively large data. For example, on the reduced version of the KDD Cup 2009 data (with 50,000 data points and 190 numeric attributes), the GPS system took about six hours to complete on an AMD 2.8G

PC with 16G RAM (number of GA generations, number of PSO evolutions, GA's population size and PSO's swarm size were all set to 10, and 3-fold cross validation was used in the objective function). Therefore, in this section, we present a strategy for speeding up the GPS algorithm. Before introducing the new algorithm, we first review the model tree idea.

5.7.1 Model Trees

A model tree [160] is a decision tree system that uses linear models at the leaves instead of using discrete class labels for a classification tree or mean as the prediction for a regression tree. Model trees, like decision trees scale well since the training data is stored in a tree structure. Some variants that have been designed based on the model tree idea show promising results, such as the logistic model tree [92].

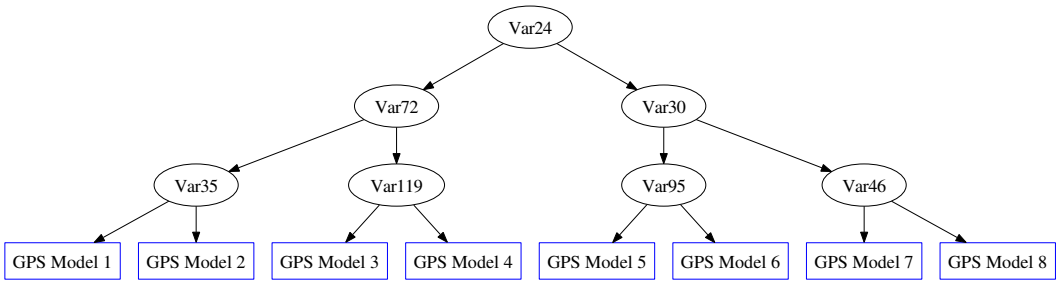
5.7.2 Full Model Tree

We here propose a novel GPS-based model tree algorithm named the Full Model Tree, because GPS builds a full model on a given dataset. The idea is that instead of training the GPS algorithm on the full training data, we build GPS models at the leaves of a tree structure. In the second set of experiments in this chapter, we compare GPS to Full Model Tree with two different tree structures, namely, the perfect binary tree and the random binary tree based on the following definitions. Figure 5.3 shows two examples of the tree structures for the KDD Cup 2009 data.

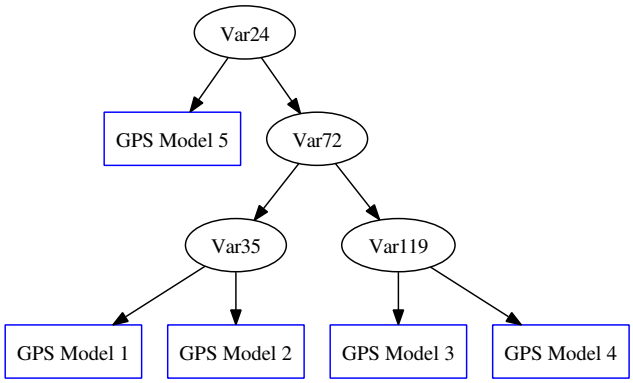
Definition 1: A perfect binary tree is a binary tree with all leaf nodes at the same depth.

Definition 2: A random binary tree is a binary tree formed by inserting nodes one at a time according to a random mechanism.

Next, we show that theoretically when the above two binary tree structures



(a) An illustration of a perfect-binary-tree-based Full Model Tree (FMT-perfect) with height = 3



(b) An illustration of a random-binary-tree-based Full Model Tree (FMT-random) with height = 3

Figure 5.3: GPS-based Full Model Trees with two different tree structures on the KDD 09 customer churn data, tree heights are set to 3

are used, and if the tree height is greater than zero and the training complexity of GPS is worse than linear, then GPS-based Full Model Tree is faster than GPS when training on the same data.

Assume the running time of the normal GPS algorithm (GPS-0) for training its model on a dataset of n data points is $O(f(n))$, and that for the GPS-based Full Model Tree is $O(g(n))$. Based on our preliminary experiments, we found that the *empirical* training complexity of GPS is worse than linear on most of the datasets we have tested, so here we consider the case for $f(n) > n^1, n > 1$.

Theorem 1. *For a perfect-binary-tree-based GPS Full Model Tree T with height $h \geq 1$; If GPS-0's empirical training complexity is worse than linear,*

such as $f(n) > n^1, n > 1$*, then we have g(n) < f(n).*

Proof.

Let l be the number of leaf nodes of T, we have $l = 2^h$, $(l \geq 2)$.

Let k be the number of data points at each leaf of T, we have $k = n/l$. Then, we have $g(n) = l \times f(k)$ and $f(n) = f(k \times l)$.

Let $f(n) = n^x$, so we have $x > 1$.

$$\begin{aligned} f(n) - g(n) &= f(k \times l) - l \times f(k) \\ &= (k \times l)^x - l \times k^x \\ &= k^x \times l^x - k^x \times l \\ &= k^x \times (l^x - l) \\ &= k^x \times l^{x-1} \\ &> 0. \end{aligned}$$

Theorem 2. *For a random-binary-tree-based GPS Full Model Tree T with height* $h \geq 1$*; If GPS-0's empirical training complexity is worse than linear, such as* $f(n) > n^1, n > 1$*, then we have g(n) < f(n).*

Proof.

Let l be the number of leaf nodes of T, $h \geq 1$ so we have $l \geq 2$.

Let k_i be the number of data points at leaf i of T, we have $\sum_{i=1}^{l} k_i = n$.

Let $f(n) = n^x$, we have $x > 1$, then we have $g(n) = \sum_{i=1}^{l} g(k_i) = \sum_{i=1}^{l} {k_i}^x$ and $f(n) = n^x = (\sum_{i=1}^{l} k_i)^x$. Therefore, $f(n) - g(n) = \{(\sum_{i=1}^{l} k_i)^x - \sum_{i=1}^{l} {k_i}^x\} > 0$.

The two theorems state that theoretically the two Full Model Tree variants are faster than GPS in the case that the training complexity of GPS is worse than linear. Results above are also applicable to memory consumption stating that the two Full Model Tree variants are supposed to be more memory efficient than GPS if the training complexity in terms of memory of GPS is worse than

Algorithm 5.2 Full Model Tree (FMT)

Input:

D (training data)

h (the tree height)

C (splitting and stopping criterions, details are given in Sec 5.7.3)

bestSplit $\leftarrow$ Test features based on the splitting criterion C.

if *stopping criterion* (e.g., tree height = h or the current data contains only one class label) is met **then**

Build a GPS model and return a leaf node.

else

// D^+ and D^- are two sub-partitions based on the best split.

leftSubtree $\leftarrow$ **FMT**($D^+_{bestSplit}$, h, C)

rightSubtree $\leftarrow$ **FMT**($D^-_{bestSplit}$, h, C)

Return (*bestSplit*, *leftSubtree*, *rightSubtree*)

end if

linear. The results also imply that if GPS's training complexity is linear or better, then theoretically the Full Model Tree variants will not speed up the original GPS algorithm. Please note that in this complexity analysis we did not consider the cost for training the tree structure itself. When full model trees are used in an ensemble strategy, one should also consider the cost for building a particular full model tree variant. Next, we describe how the GPS-based Full Model Trees are built.

5.7.3 Growing GPS-based Full Model Trees

Algorithm 5.2 shows the pseudocode of FMT. Next we explain how a FMT is built. When growing a perfect binary tree, firstly the algorithm checks if the tree height is equal to a user-specified value h. If tree height = h or the current

data contains only one class, then make a leaf node and build a GPS model, else the best variable is selected for splitting. Here, *best* is based on the information gain measure of a variable. For numeric variables, we examine information gain using the median of a variable as the splitting point; for nominal variables, we balance the number of data points from distinct categorical values. For instance, imagine a nominal variable having two distinct categorical values A and B; if the data has 100 data points, where 80 of them belong to A, and 20 of them belong to B, then we randomly select 30 data points from A, and put them into B. If the nominal variable has three distinct categorical values, say A with 60 data points, B with 30 data points, and C with 10 data points, then we merge B and C first, and then balance A and BC by randomly moving 10 data points from A to BC. The same balancing strategy is also applicable to a nominal variable having more than three distinct categorical values. In this way the amount of data from the current node is roughly equally split for its two child nodes.

When growing a random binary tree, firstly the algorithm checks if the tree height is equal to a user-specified value h. If tree height $= h$ or the current data contains only one class, then make a leaf node and build a GPS model, else the algorithm randomly chooses one of the best K variables for splitting. Here, *best* is based on the information gain measure of a variable, where K is a user-specified value. For numeric variables, the best splitting point is found by trying all possible splitting points between two neighbored numbers (the splitting point with the highest gain will be selected); for nominal variables, the data is split between the majority categorical value and the other categorical values.

Predictive Performance and Runtime Next, we examine both the predictive performance and the runtime of the two Full Model Tree variants (one uses the perfect binary tree structure, the other uses the random binary

Dataset	GPS	FMT-perfect	FMT-random	GPS	FMT-perf.	FMT-rand.
	AUC			Runtime (mins)		
Adult	.94 ± .002	.93 ± .003	.93 ± .002 ⊖	45 ± 6	37 ± 4 ◊	48 ± 11
Connect-4.	.95 ± .002	.95 ± .002	.95 ± .003 ⊖	91 ± 5	77 ± 9 ◊	74 ± 14 ◊
KDD Cup.	.77 ± .002	.77 ± .002	.76 ± .003 ⊖	178 ± 9	157 ± 11 ◊	189 ± 8
Mini.B.E.	.98 ± .002	.98 ± .002 ⊖	.97 ± .003 ⊖	124 ± 7	123 ± 9	135 ± 12
UCSD.	.68 ± .003	.68 ± .002 ⊖	.67 ± .002 ⊖	487 ± 16	417 ± 19 ◊	476 ± 17

Table 5.2: Performance and runtime of the GPS and the Full Model Tree algorithms; A "⊖" indicates that in terms of AUC, the GPS algorithm is significantly better than the respective algorithm; A "◊" indicates that in terms of runtime, the GPS algorithm is significantly slower than the respective algorithm; level of significance 0.05

tree structure, namely, FMT-perfect and FMT-random, respectively) to the original GPS algorithm.

We use five medium size datasets for this experiment. Table 4.1 shows the properties of these datasets. The original *KDDCup09* dataset has 50,000 data points, 190 numeric variables and 40 categorical variables. To speed up the experiment, the 40 categorical variables were removed from the data because some variables have thousands of distinct values. We set the height of both FMT-perfect and FMT-random to 3. So, for FMT-perfect, there will be $2^3 = 8$ leaf GPS models to be built, and each leaf will have $n/8 \pm 1$ data points where n is the total number of training data points. The K value for FMT-random is set to $log(M)+1$, where M is the number of variables. For the GPS algorithm, the number of generations for GA, the population size for GA, the number of evolutions for PSO, and the swarm size for PSO are all set to 10. The objective function of GPS is based on 2-fold cross validation.

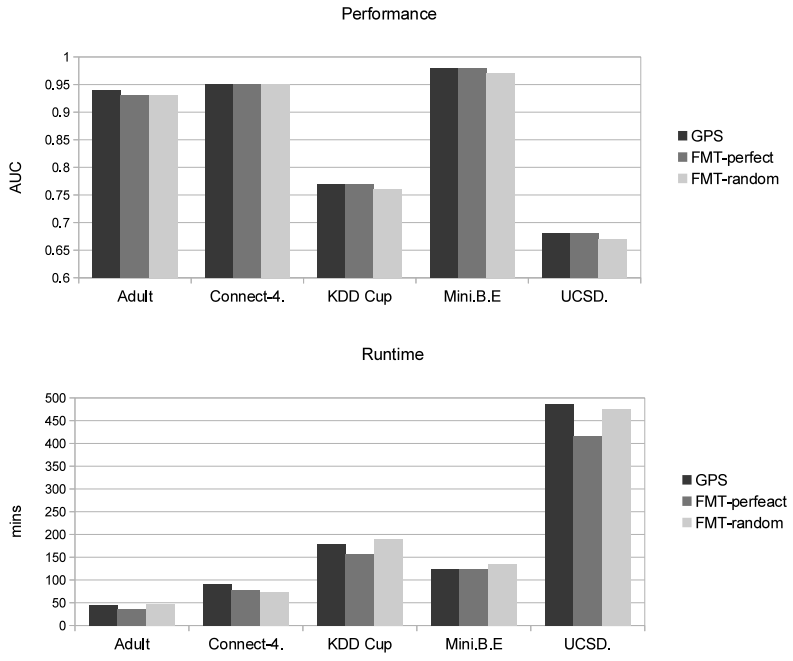


Figure 5.4: Performance and runtime of the GPS and the Full Model Tree algorithms

Table 5.2 shows the comparison results based on 5 times 3-fold cross validation. The AUC performance and the runtime are reported. For the AUC performance, we can see that GPS significantly outperforms FMT-random on all datasets, indicating that FMT-random is not good enough to be used as a GPS alternative. The GPS algorithm significantly outperforms FMT-perfect on two datasets; for the other three datasets, the performance of GPS and FMT-perfect has no significant difference. This indicates that for these three datasets, FMT-perfect can be used as a GPS alternative. In terms of runtime, the FMT-random algorithm is significantly faster than GPS only on the *Connect-4.* dataset. One reason could be that the number of data points at the leaf nodes of FMT-random are not the same, so the empirical training complexity of FMT-random varies at each leaf. We can see that FMT-perfect is faster than FMT-random on all datasets because usually the number of leaf nodes of FMT-random is less than that for FMT-perfect. The results show that

FMT-perfect is significantly faster then GPS on 4 out of 5 datasets, indicating that FMT-perfect is a viable approach for speeding up GPS. Overall, on 3 out of 5 datasets, namely, *Adult*, *Connect-4.*, and *KDDCup09*, the perfect-binary-tree-based Full Model Tree could significantly speed up the GPS algorithm without sacrificing GPS's predictive power. Figure 5.4 shows the respective bar chart based on the data in Table 5.2.

The FMT algorithms also provide a way of learning full models on larger datasets. For example, if the memory consumption complexity of a data mining solution returned by GPS is $O(n^3)$, where n is the number of training examples. Then, training this solution on a relatively large dataset would be infeasible. However, using FMTs we could reduce the actual memory consumption of that solution. In this case, in order to get competitive accuracy, FMT algorithms, such as the Random FMT could be used as base-learners of an ensemble learning strategy, e.g., random forests.

5.8 The GUI system

We have developed a proof-of-concept system based on the DMO framework using the GPS algorithm as the optimisation engine. Given a dataset, the system searches for optimal solutions, including data preprocessing, outlier detection, feature selection and algorithm DMOs. The final solution can be saved as a WEKA object and loaded into the WEKA GUI or used with the WEKA APIs. Figure 5.5 shows a screenshot of the system.

5.8.1 Workflow-Based Systems

A related and promising line of research focusses on developing workflow-based graphical user interface for KDD (Knowledge Discovery in Databases) tasks. These systems allow the end-user to design and optimise a KDD workflow for a given dataset. Usually a KDD workflow consists of some GUI nodes

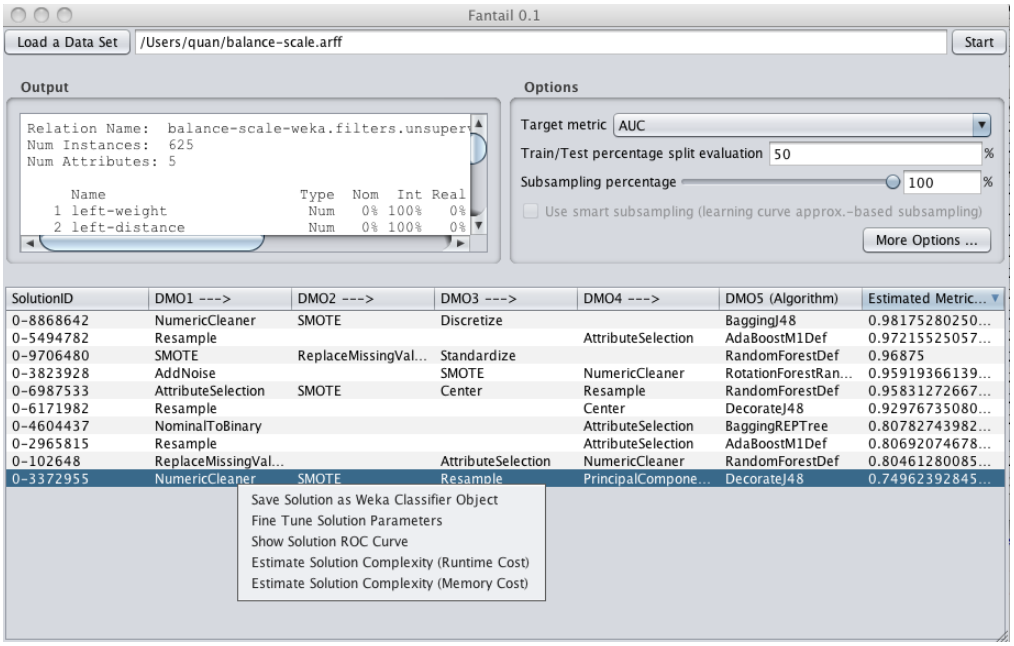


Figure 5.5: A proof-of-concept system based on the DMO framework using the GPS algorithm as the optimisation engine.

connected by arrows (a sequence of connected steps), and is displayed on a computer screen. A particular KDD workflow may be seen as an abstraction and virtual representation of a real KDD task. Several workflow systems have been developed over the past decade, including the workflow GUI of WEKA [74], ADAMS [128], RapidMiner[4], eIDA and ePro-Plan [88], and the eLico project[5].

5.9 Conclusions

In this chapter, we proposed a framework, in the DMO space setting, which can be used for designing new full model selection, or FMS, algorithms, and a novel FMS algorithm which can be seen as a realization and an application of the framework. Our experiments on ten real-world problems show that the GPS algorithm performs very competitively with PSMS, the state-of-the-art

[4]http://www.rapidminer.com

[5]http://www.e-lico.org

PSO-based FMS algorithm. We also examined the feasibility of using the model tree idea for speeding up the GPS algorithm. Our experimental results suggest that using the perfect binary tree as the internal tree structure for the GPS-based Full Model Tree algorithm is a viable approach when the empirical training complexity of GPS is worse than linear. The techniques described in this chapter could probably also be applied to regression and label ranking problems, but this needs to be verified in a future study. Another future work direction is to compare the performance of the GPS systems to fine-tuned base-level ensemble algorithms. The 5-DMO template (3) defined in Section 2 is only one of many possible templates for practical data mining solutions, in future research we will also investigate methods for optimizing alternative templates simultaneously in a cloud environment.

Chapter 6

Future Work and Conclusion

Over the past few decades, a rich set of data analysis techniques, including algorithms and methods, have been developed in the machine learning and data mining community. Nowadays users of machine learning are facing a new problem:

"*How to choose an appropriate set of tools for a given data problem in order to achieve better data mining than simply using an arbitrary selection of tools*?"

This thesis has dealt with the challenging problem from two approaches, both of which try to propose systematic and efficient solutions to the above question, namely the meta-learning-based algorithm/parameter recommendation approach (the meta-learning approach) and the meta-heuristic search-based full model selection approach (the FMS approach). The effectiveness of the proposed algorithms and systems have been examined in the context of algorithm, parameter and data mining solution recommendation.

6.1 Future Work

In this section, we discuss several directions for future research.

6.1.1 Alternative Splitting Heuristics for ART

The splitting criteria of the ART algorithm is based on the R^2 statistics. The advantage of using R^2 is that it can be computed using a close form formula. Therefore ART is suitable to be used as the base-level learner for ensemble learning.

The average rank vector used in ART is an approximation to the true modal ranking. For some datasets, this approximation might not be good enough. In this case, alternative statistics could be used as the splitting heuristics, such as the F-statistics:

$$F = \frac{n-L}{L} \frac{\sum_{l=1}^{L} n^{(l)} (\bar{d}_l - \bar{d})^2}{\sum_{l=1}^{L} \sum_{i=1}^{n^{(l)}} (d_{li} - \bar{d}_l)^2},$$

where L is the number of partitions, $\bar{d}_l$ is the sample average of the distances for partition l, and $\bar{d}$ is the average of the entire sample, n is the sample size and $d_{li} \equiv d(y^{<li>}, \hat{w}^{<T>})$. Here, $\bar{d}$ needs to be calculated by first calculating all the pairwise distance between each ranking and $\hat{w}^{<T>}$ (the modal ranking) needs to be estimated from the data.

6.1.2 Alternative Aggregation Methods for ART Ensembles

In the ART forests algorithm, the final ensemble is aggregated based on simple averaging, in which the weights of individual base-level models are all equal to $1/N$, where N is the number of ART models. In the ESMetal algorithm, the weight of each base-level model is calculated from a forward stepwise procedure based on hillclimb set performances. By its nature, a ranking aggregation method (or combiner) acts as a "smoother" in that it minimizes the effect of outliers, such as rankings containing anomalies that seem inconsistent with other rankings [93].

In future work, we plan to investigate alternative aggregation methods,

such as aggregating based on counting inconsistent pairs and removing rankings with too many inconsistent counts; Borda counting techniques; Graph-theory-based aggregation and optimization-based aggregation, e.g., binary integer linear program (BILP).

6.1.3 Multi-Purpose Optimization

When searching through the DMO space with a single goal, such as minimizing the classification error, each DMO solution is evaluated based simply on its classification error. However, we may have a complex goal (or multiple goals); for instance, we might not only want the DMO solution to be optimized to the classification error, but we might also want it to be memory efficient (e.g., use a fixed amount of memory). In such a case, we could employ multi-purpose optimization (MPO) techniques [109, 173]. A simple approach using MPO is to give each goal a weight (specified by the user), and then the final evaluation score can be expressed as:

$$score_{final} = w_1 score_{goal_1} + w_2 score_{goal_2} +, ..., + w_K score_{goal_K} = \sum_{k=1}^{K} w_k score_{goal_k},$$

where K is the number of goals. In this way, the final solution can be seen as an optimal solution for all of the K goals. In future research, we will also investigate MPO and related techniques for the problem of searching the DMO space with multiple goals.

6.1.4 Multi-Criteria Decision Making

The meta-learning for algorithm/parameter recommendation problem could possibly be studied in the Multi-Criteria Decision Making (MCDM) framework [151, 152]. In the typical MCDM problem the decision maker is given m alternatives (such as algorithms) to evaluate in terms of n decision criteria (such as accuracy, runtime, RAM consumption, interpretability etc). Denoted the m

alternatives by $(A_1, A_2, A_3, ..., A_m)$ and the n criteria by $(C_1, C_2, C_3, ..., C_m)$, the task is to determine the relative performance of the m alternatives. A decision maker needs to form a decision matrix D and also determine the vector W with the weights of the n criteria. An entry of M, denoted by m_{ij} represents the performance of A_i in terms of criteria C_j.

A central problem in MCDM is how to make decisions when the criteria are expressed in different units of measure [151]. There are several proposed methodologies on how the MCDM problem can be solved. Among these methods, the Analytic Hierarchy Process (AHP) method is widely accepted by researchers. The AHP decomposes the decision problem into a hierarchy of sub-problems. Then the decision-maker evaluates the relative importance of its various elements by pairwise comparisons. The AHP converts these evaluations to numerical values (weights or priorities), which are used to calculate a score for each alternative [130, 131].

In future research, together with the MPO approach described in the previous section, we will extend our algorithms and systems to deal with multiple criteria instead of just accuracy.

6.1.5 Interactive Learning

Adding the ability to supply user knowledge is another direction for speeding up the Full Model Selection search described in Chapter 5. For example, the GPS algorithm could present to the end-user a list of solutions during the search, so the end-user decides which solutions to include for further search in order to reduce the search space.

6.1.6 Ensembling FMTs

In Section 5, we studied using the Full Model Trees (FMT) algorithm to speedup the GPS algorithm. Another direction is to combine multiple FMTs

to form an FMT ensemble to achieve a better performance.

6.1.7 Meta-learning and FMS

A meta-learning system is trained on a meta-dataset (meta-features are based on a dataset collection). So if the values of meta-features of a new dataset are outside the ranges of the training meta-dataset, then in this case, meta-learning may fail because the target distribution is changed (no longer i.i.d).

For example, using a meta-learning system trained on balanced datasets to predict on imbalanced datasets will reduce the accuracy of the meta-learning system. Another example is using UCI dataset to predict text dataset.

Therefore, another direction for future work is to combine the two approaches proposed in this thesis. For instance, meta-learning-based algorithm and parameter ranking can be used for generating the initial population of the meta-heuristic search-based FMS approach.

Together with the models described in [143], the techniques proposed in this thesis could also be used to model and predict/rank data mining solutions by their runtime of training or prediction stage. The final Meta-learning plus FMS hybrid system would be able to suggest an optimal data mining solution with detailed algorithm and parameter and runtime recommendation.

6.2 Contributions

This research has led to general contributions to the field of data mining and machine learning. Next, we summarize the main contributions of this thesis.

- We proposed a novel meta-feature generation method for meta-learning-based algorithm ranking, which simultaneously improves the predictive performances of different meta-learners (Chapter 3).

- We proposed a new meta-learner called Approximate Ranking Tree Forests

(ART forests) that performs very competitively when compared with several state-of-the-art meta-learners. We present both theoretical and empirical contributions (Chapter 3).

- We also proposed a new experimental configuration that is more realistic for applications of meta-learning. We also present a default algorithm ranking list of 20 machine learning algorithms based on their optimized performances over 466 datasets (Chapter 3).

- We proposed a new ensemble learning algorithm for classification, regression and ranking with extensive empirical evaluation and theoretical analysis. We also provided two meta-learning case studies for testing the predictive performance of the new ensemble strategy (Chapter 4).

- We proposed a framework which can be used for designing new FMS (full model selection) algorithms in terms of the DMO space (Chapter 5).

- We proposed a novel FMS algorithm (called GPS) which can be seen as a realization and an application of the proposed DMO framework. Our experiments on real-world problems show that the proposed algorithm performs very competitively with PSMS, the state-of-the-art PSO-based FMS algorithm (Chapter 5).

- We examined the feasibility of using the model tree idea for speeding up the GPS algorithm. Our experimental results suggest that using the perfect binary tree as the internal tree structure for GPS-based Full Model Tree is a viable approach when the empirical training complexity of GPS is worse than linear (Chapter 5).

This thesis presents research results from two approaches, namely the meta-learning-based algorithm/parameter recommendation approach and the meta-heuristic search-based Full Model Selection (FMS) approach, both of which

propose systematic and efficient solutions to the problem of data mining tool recommendation.

The main difference between the two approaches is due to the time required for making a useful recommendation. By contrast, the FMS approach does not suffer the "unknown" data characteristics issue since the candidate models and solutions are evaluated directly on a target dataset. However, it can be much slower than meta-learning in terms of making a recommendation. Therefore, when fast recommendation is essential, the meta-learning approach is recommended; whereas when meta-learning's recommendation is not good enough, the FMS is a reasonable alternative.

Potentially, the new techniques, algorithms and methods proposed in this thesis can be used as building blocks in a data mining recommendation engine for a given data problem.

References

[1] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] Shawkat Ali and Kate A. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(13):173 – 186, 2006.

[3] Mayer Alvo, Paul Cabilio, and Paul D. Feigin. Asymptotic theory for measures of concordance with special reference to average kendall tau. *The Annals of Statistics*, 10(4):pp. 1269–1276, 1982.

[4] Per Andersen and Niels Christian Petersen. A procedure for ranking efficient units in data envelopment analysis. *Manage. Sci.*, 39(10):1261–1264, October 1993.

[5] J. Attenberg, K.Q. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich. Collaborative Email-Spam Filtering with the Hashing Trick. In *Proceedings of the Sixth Conference on Email and Anti-Spam (CEAS 2009)*, 2009.

[6] Timothy L. Bailey and Charles Elkan. Estimating the accuracy of learned concepts. In *In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 895–900. Morgan Kaufmann, 1993.

[7] Jacky Baltes and Bruce Macdonald. Case-based meta-learning: Sustained learning supported by a dynamically biased version space. In

Proceedings of the Machine Learning Workshop on Biases in Inductive Learning, 1992.

[8] Rajiv D Banker, Abraham Charnes, and William Wager Cooper. Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management science*, 30(9):1078–1092, 1984.

[9] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, CA, 1998.

[10] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, July 1999.

[11] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[12] H. Bensusan, C. Giraud-Carrier, and C. Kennedy. A higher-order approach to meta-learning. Technical report, University of Bristol, 2000.

[13] Abraham Bernstein, Shawndra Hill, and Foster Provost. Intelligent assistance for the data mining process: An ontology-based approach. *Information Systems Working Papers Series, Vol*, 2002.

[14] Jinbo Bi, Kristin P. Bennett, Mark Embrechts, Curt M. Breneman, Minghu Song, Isabelle Guyon, and Andre Elisseeff. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, 2003.

[15] G. Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, 13:1063–1095, April 2012.

[16] Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. En-

sembles of restricted hoeffding trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(2):30, 2012.

[17] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[18] Ulisses Braga-Neto and Edward Dougherty. Bolstered error estimation. *Pattern Recognition*, 37(6):1267–1281, 2004.

[19] Ulisses M Braga-Neto and Edward R Dougherty. Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380, 2004.

[20] P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the European Conference on Machine Learning*, 1994.

[21] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer, 2009.

[22] P. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In Ramon Lpez de Mntaras and Enric Plaza, editors, *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Computer Science*, pages 63–75. Springer Berlin Heidelberg, 2000.

[23] P. Brazdil, C. Soares, and J. P. Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Mach. Learn.*, 50(3):251–277, March 2003.

[24] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[25] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[26] Carla E. Brodley. Recursive automatic bias selection for classifier construction. *Machine Learning*, 20:63–94, 1995.

[27] Carla E. Brodley and Padhraic Smyth. Applying classification algorithms in practice. *Statistics and Computing*, 7:45–56, 1997.

[28] R. Bryll, R. Gutierrez-Osuna, and F. Quek. Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302, 2003.

[29] Peter Buhlmann and Sara van de Geer. *Statistics for High-Demensional Data: Methods, Theory and Applications*. Springer, 2011.

[30] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[31] Rui Camacho and Pavel Brazdil. Improving the robustness and encoding complexity of behavioural clones. In *Machine Learning: European Conference on Machine Learning 2001*, pages 37–48. Springer, 2001.

[32] E. Cantu-Paz. Feature subset selection, class separability, and genetic algorithms. In 959-970, editor, *Genetic and Evolutionary Computation*, volume 3102 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[33] R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *ICDM '06 Proceedings of the Sixth International Conference on Data Mining*, 2006.

[34] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *ICML '04 Proceedings of the twenty-first international conference on machine learning*, 2004.

[35] R. Caruana and Alexandru Niculescu-mizil. An empirical comparison of supervised learning algorithms. In *In Proc. 23rd Intl. Conf. Machine learning*, pages 161–168, 2006.

[36] Rich Caruana. Multitask learning: a knowledge-based source of inductive bias. *Machine learning*, 28(1):41–75, 1997.

[37] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[38] Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD'10, pages 1189–1198, New York, NY, USA, 2010. ACM.

[39] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16:321–357, June 2002.

[40] Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier. Decision tree and instance-based learning for label ranking. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 161–168, Montreal, Canada, June 2009.

[41] Weiwei Cheng and Eyke Hüllermeier. Instance-based label ranking using the mallows model. In *Workshop Proceedings of Preference Learning*, Antwerp, Belgium, 2008.

[42] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.

[43] P.B.C. de Miranda, R.B.C. Prudencio, A.C.P.L.F. Carvalho, and C. Soares. Combining a multi-objective optimization approach with meta-learning for svm parameter selection. In *2012 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2909–2914, 2012.

[44] Pierre Del Moral and Pascal Lezaud. Branching and interacting particle interpretations of rare event probabilities. In *Stochastic hybrid systems*, pages 277–323. Springer, 2006.

[45] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, December 2006.

[46] P. A. Devijver and J. Kittler. *Pattern recognition: A statistical approach.* Prentice Hall, 1982.

[47] Luca Didaci, Giorgio Fumera, and Fabio Roli. Diversity in classifier ensembles: Fertile concept or dead end? In Zhi-Hua Zhou, Fabio Roli, and Josef Kittler, editors, *Multiple Classifier Systems*, volume 7872 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg, 2013.

[48] Thomas Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, pages 1–15, 2000.

[49] Saso Deroski and Bernard enko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[50] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, 1993.

[51] H. J. Escalante, M. Montes, and L. E. Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10:405–440, 2009.

[52] M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(3):pp. 359–369, 1986.

[53] Michael A. Fligner and Joseph S. Verducci. Multistage ranking models. *Journal of the American Statistical Association*, 83(403):pp. 892–901, 1988.

[54] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[55] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[56] Jerome H Friedman. Regularized discriminant analysis. *Journal of the American statistical association*, 84(405):165–175, 1989.

[57] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

[58] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[59] Holger Frohlich and Andreas Zell. Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, volume 3, pages 1431–1436. IEEE, 2005.

[60] J. Gama and P. Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.

[61] S. Geisser. *Predictive Inference: An Introduction*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1993.

[62] C. Giraud-Carrier. Metalearning - a tutorial. In *Proceedings of the 7th International Conference on Machine Learning and Applications*. Morgan Kaufmann, 2008.

[63] C. Giraud-Carrier and F. Provost. Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper? In *Proceedings of the ICML-2005 Workshop on Meta-learning*, pages 12–19, 2005.

[64] Christophe Giraud-Carrier. The data mining advisor: Meta-learning at the service of practitioners. In *Proceedings of the Fourth International Conference on Machine Learning and Applications*, ICMLA '05, pages 113–119, Washington, DC, USA, 2005. IEEE Computer Society.

[65] Fred Glover. Tabu search part i. *ORSA Journal on computing 1*, 2:190–206, 1989.

[66] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[67] Fred Glover. Tabu search part ii. *ORSA Journal on computing 2*, 1:4–32, 1990.

[68] Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *Computers & operations research*, 13(5):563–573, 1986.

[69] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA, 1989.

[70] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2-3):95–99, 1988.

[71] Taciana A.F. Gomes, Ricardo B.C. Prudncio, Carlos Soares, Andr L.D. Rossi, and Andr Carvalho. Combining meta-learning and search tech-

niques to select parameters for support vector machines. *Neurocomputing*, 75(1):3 – 13, 2012.

[72] C Gondro and BP Kinghorn. A simple genetic algorithm for multiple sequence alignment. *Genet. Mol. Res*, 6(4):964–982, 2007.

[73] Diana Gordon and Donald Perlis. Explicitly biased generalization. *Computational Intelligence*, 5(2):67–81, 1989.

[74] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[75] D. Hernandez-Lobato, G. Martinez-Munoz, and A. Suarez. Pruning in ordered regression bagging ensembles. In *IJCNN '06 International Joint Conference on Neural Networks*, pages 1266 –1273, 2006.

[76] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[77] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, USA, 1975.

[78] Jianping Hua, Waibhav D Tembe, and Edward R Dougherty. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition*, 42(3):409–424, 2009.

[79] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artif. Intell.*, 172(16-17):1897–1916, November 2008.

[80] F. Hussein. Genetic algorithms for feature selection and weighting, a review and study. In *ICDAR '01 Proceedings of the Sixth International Conference on Document Analysis and Recognition*, 2001.

[81] Wayne Iba and Pat Langley. Induction of one-level decision trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, 1992.

[82] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[83] Ocenasek Jiri. Parallel estimation of distribution algorithms. *PhD. Thesis, Faculty of Information Technology, Brno University of Technology*, pages 1–154, 2002.

[84] A. Kalousis. *Algorithm Selection via Meta-Learning*. PhD thesis, Department of Computer Science, University of Geneva, 2002.

[85] A. Kalousis and M. Hilario. Model selection via meta-learning: a comparative study. *International Journal on Artificial Intelligence Tools*, 10(04):525–554, 2001.

[86] Maurice G. Kendall. *Rank correlation methods*. Griffin, 1970.

[87] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[88] Jörg-Uwe Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. Designing kdd-workflows via htn-planning for intelligent discovery assistance. In *5 th PLANNING TO LEARN WORKSHOP WS28 AT ECAI 2012*, page 10, 2012.

[89] Shinichi Kikuchi, Daisuke Tominaga, Masanori Arita, Katsutoshi Takahashi, and Masaru Tomita. Dynamic modeling of genetic networks using genetic algorithm and s-system. *Bioinformatics*, 19(5):643–650, 2003.

[90] Scott Kirkpatrick et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[91] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[92] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Mach. Learn.*, 59:161–205, May 2005.

[93] Amy N. Langville and Carl D. Meyer. *Who is Number One? The Science of Rating and Ranking.* Princeton University Press, 2012.

[94] C.L. Lawson and R.J. Hanson. *Solving Least-Squares Problems.* Prentice-Hall, 1974.

[95] I. Lee and B. Moon. Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11), 2004.

[96] Rui Leite and Pavel Brazdil. Predicting relative performance of classifiers from samples. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

[97] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithm with active testing on similar datasets. In *Proceedings of the 5th International Workshop on Planning to Learn*, 2012.

[98] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In Petra Perner, editor, *Machine*

Learning and Data Mining in Pattern Recognition, volume 7376 of *Lecture Notes in Computer Science*, pages 117–131. Springer Berlin Heidelberg, 2012.

[99] S. Lessmann, R. Stahlbock, and S.F. Crone. Genetic algorithms for support vector machine model selection. *Neural Networks*, pages 3063–3069, 2006.

[100] Yun Li, Kiam Heong Ang, Gregory CY Chong, Wenyuan Feng, Kay Chen Tan, and Hiroshi Kashiwagi. Cautocsd-evolutionary search and optimisation enabled computer automated control system design. *International Journal of Automation and Computing*, 1(1):76–88, 2004.

[101] A. Lorena and A. Carvalho. Evolutionary tuning of svm parameter values in multiclass problems. *Neurocomputing*, 71(16-18), 2008.

[102] Sean Luke. *Essentials of Metaheuristics*. Lulu Publishing, Department of Computer Science, George Mason University, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[103] Jinjin Ma. Parameter tuning using gaussian processes. Master's thesis, The University of Waikato, 2012.

[104] John I. Marden. *Analyzing and Modeling Rank Data*. Chapman & Hall, 1995.

[105] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[106] Michael Mayo. Evolutionary data selection for enhancing models of intraday forex time series. In *Applications of Evolutionary Computation*, pages 184–193. Springer, 2012.

[107] Michael Mayo. Identifying market price levels using differential evolution. In *Applications of Evolutionary Computation*, pages 203–212. Springer, 2013.

[108] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21:1087, 1953.

[109] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[110] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[111] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[112] Michinari Momma and Kristin P. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2002.

[113] S. Olafsson. Metaheuristics. In Shane G. Henderson and Barry L. Nelson, editors, *Simulation*, Handbooks in Operations Research and Management Science, chapter 21, pages 633–654. Elsevier, 2006.

[114] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. An ensemble uncertainty aware measure for direct hill climbing ensemble pruning. *Machine Learning*, 81(3), 2010.

[115] Martin Pelikan. *Bayesian optimization algorithm: from single level to hierarchy*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2002. AAI3070411.

[116] Martin Pelikan, David E Goldberg, and Erick Cantu-Paz. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation*, 8(3):311–340, 2000.

[117] Bernhard Pfahringer. Semi-random model tree ensembles: An effective and scalable regression method. In Dianhui Wang and Mark Reynolds, editors, *AI 2011: Advances in Artificial Intelligence*, volume 7106 of *Lecture Notes in Computer Science*, pages 231–240. Springer Berlin Heidelberg, 2011.

[118] Bernhard Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.

[119] Richard R Picard and R Dennis Cook. Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.

[120] Joaquim Pinto da Costa and Carlos Soares. A weighted rank measure of correlation. *Australian & New Zealand Journal of Statistics*, 47(4):515–529, 2005.

[121] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[122] J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5:239–266, 1990.

[123] J. R. Quinlan. Learning with continuous Classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[124] M. Reif, F. Shafait, and A. Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning*, 87:357–380, 2012.

[125] Benjamin Reilly. Social choice in the south seas: Electoral innovation and the borda count in the pacific island countries. *International Political Science Review*, 23(4):355–372, 2002.

[126] Larry Rendell and Howard Cho. Empirical learning as a function of concept character. *Machine Learning*, 5:267–298, 1990.

[127] Larry Rendell, Raj Seshu, and David Tcheng. Layered concept-learning and dynamically-variable bias management. In *In Proceedings of IJCAI-87*, pages 308–314. Morgan Kaufmann, 1987.

[128] Peter Reutemann and Joaquin Vanschoren. Scientific workflow management with adams. In PeterA. Flach, Tijl Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 833–837. Springer Berlin Heidelberg, 2012.

[129] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33:1–39, 2010.

[130] Thomas L Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. New York: McGraw-Hill, 1980.

[131] Thomas L Saaty. *Fundamentals of decision making and priority theory: with the analytic hierarchy process*. Rws Publications, 1994.

[132] C. Schaffer. A conservation law for generalization performance. In *Proceedings of the 11th International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann, 1994.

[133] J. D. Schaffer and L. J. Eshelman. *Modern Heuristic Search Methods*, chapter Combinatorial Optimization by Genetic Algorithms: The Value of the Genotype/Phenotype Distinction, pages 85–96. John Wiley and Sons, Inc, 1996.

[134] Robert E Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. MIT Press (MA), 2012.

[135] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[136] A.K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In *Nineteenth International Conference on Machine Learning*, 2002.

[137] Jude W. Shavlik, Raymond J. Mooney, and Geoffrey G. Towell. Symbolic and neural learning algorithms: an experimental comparison. In *Machine Learning*, pages 111–143, 1991.

[138] K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):6:1–6:25, January 2009.

[139] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[140] Carlos Soares. *Learning Ranking of Learning Algorithms*. PhD thesis, Department of Computer Science, University of Porto, 2004.

[141] Carlos Soares and Pavel Brazdil. Zoomed ranking: Selection of classification algorithms based on relevant performance information. *Principles of Data Mining and Knowledge Discovery*, pages 160–181, 2000.

[142] Carlos Soares, Pavel B. Brazdil, and Petr Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, 54(3):195–209, 2004.

[143] Quan Sun. Sampling-based prediction of algorithm runtime. Master's thesis, Department of Computer Science, University of Waikato, 2009.

[144] Quan Sun and Bernhard Pfahringer. Bagging ensemble selection. In *Proceedings of the 24nd Australasian Conference on Artificial Intelligence*, Perth, Australia, pages 251–260. Springer, 2011.

[145] Quan Sun and Bernhard Pfahringer. Bagging ensemble selection for regression. In *AI 2012: Advances in Artificial Intelligence*, pages 695–706. Springer, 2012.

[146] Quan Sun and Bernhard Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, 93(1):141–161, 2013.

[147] Quan. Sun, Bernhard. Pfahringer, and Michael. Mayo. Full model selection in the space of data mining operators. In *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion*, 2012.

[148] Quan Sun, Bernhard Pfahringer, and Michael Mayo. Towards a framework for designing full model selection and optimization systems. In Zhi-Hua Zhou, Fabio Roli, and Josef Kittler, editors, *Multiple Classifier Systems*, volume 7872 of *Lecture Notes in Computer Science*, pages 259–270. Springer Berlin Heidelberg, 2013.

[149] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.

[150] L. Todorovski, H. Blockeel, and S. Dzeroski. Ranking with predictive clustering trees. In *Proceedings of the 13th European Conference on Machine Learning*. Springer, 2002.

[151] Evangelos Triantaphyllou. Reduction of pairwise comparisons in decision making via a duality approach. *Journal of Multi-Criteria Decision Analysis*, 8(6):299–310, 1999.

[152] Evangelos Triantaphyllou. Multi-criteria decision making methods. In *Multi-criteria Decision Making Methods: A Comparative Study*, volume 44 of *Applied Optimization*, pages 5–21. Springer US, 2000.

[153] Paul Everett Utgoff. *Shift of bias for inductive concept learning*. PhD thesis, Rutgers University, New Brunswick, NJ, USA, 1984. AAI8507161.

[154] H. Vafaie and L. F. Imam. Feature selection methods: Genetic algorithms vs. greedy-like search. In *Proceedings of the International Conference on Fuzzy and Intelligent Control Systems*, 1994.

[155] Haleh Vafaie and Kenneth De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Tools with Artificial Intelligence, 1992. TAI'92, Proceedings., Fourth International Conference on*, pages 200–203. IEEE, 1992.

[156] Joaquin Vanschoren, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. Experiment databases. *Machine Learning*, 87(2):127–158, 2012.

[157] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[158] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18:77–95, 2002.

[159] Ricardo Vilalta, Christophe Giraud-Carrier, Pavel Brazdil, and Carlos Soares. Using meta-learning to support data mining. *International Journal of Computer Science & Applications*, 1, 2004.

[160] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.

[161] S.R. Waterhouse and A.J. Robinson. Classification using hierarchical mixtures of experts. In *Neural Networks for Signal Processing [1994] IV. Proceedings of the 1994 IEEE Workshop*, pages 177 –186, sep 1994.

[162] G. I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2), 2000.

[163] Sholom M. Weiss and Ioannis Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 781–787. Morgan Kaufmann, 1989.

[164] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

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

[166] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007.

[167] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th International Conference on Research and Development in Information Retrieval*. ACM, 2007.

[168] Yang Yu, Zhi-Hua Zhou, and Kai Ming Ting. Cocktail ensemble for regression. In *Seventh IEEE International Conference on Data Mining, 2007. ICDM 2007*, pages 721 –726, 2007.

[169] B. Yuan and M. Gallagher. A hybrid approach to parameter tuning in genetic algorithms. *Evolutionary Computation*, 2:1096–1103, 2005.

[170] Jun Zhang, Henry SH Chung, and WL Lo. Pseudocoevolutionary genetic algorithms for power electronic circuits optimization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(4):590–598, 2006.

[171] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC machine learning & pattern recognition series. Taylor & Francis, 2012.

[172] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial Intelligence*, 137:239–263, May 2002.

[173] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. In *Metaheuristics for Multiobjective Optimisation*, pages 3–37. Springer, 2004.