2011-01-27

Relationships Among Learning Algorithms and Tasks

Jun won Lee
Brigham Young University - Provo

Relationships Among Learning Algorithms and Tasks

Jun won Lee

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Christophe Giraud-Carrier, Chair
Dan Ventura
Yiu-Kai Dennis Ng
Eric Mercer
Sean Warnick

Department of Computer Science

Brigham Young University

April 2011

ABSTRACT

Relationships Among Learning Algorithms and Tasks

Jun won Lee

Department of Computer Science

Doctor of Philosophy

Metalearning aims to obtain knowledge of the relationship between the mechanism of learning and the concrete contexts in which that mechanisms is applicable. As new mechanisms of learning are continually added to the pool of learning algorithms, the chances of encountering behavior similarity among algorithms are increased. Understanding the relationships among algorithms and the interactions between algorithms and tasks help to narrow down the space of algorithms to search for a given learning task. In addition, this process helps to disclose factors contributing to the similar behavior of different algorithms.

We first study general characteristics of learning tasks and their correlation with the performance of algorithms, isolating two metafeatures whose values are fairly distinguishable between easy and hard tasks. We then devise a new metafeature that measures the difficulty of a learning task that is independent of the performance of learning algorithms on it. Building on these preliminary results, we then investigate more formally how we might measure the behavior of algorithms at a finer grained level than a simple dichotomy between easy and hard tasks. We prove that, among all many possible candidates, the Classifier Output Difference (COD) measure is the only one possessing the properties of a metric necessary for further use in our proposed behavior-based clustering of learning algorithms. Finally, we cluster 21 algorithms based on COD and show the value of the clustering in 1) highlighting interesting behavior similarity among algorithms, which leads us to a thorough comparison of Naive Bayes and Radial Basis Function Network learning, and 2) designing more accurate algorithm selection models, by predicting clusters rather than individual algorithms.

Keywords: MetaLearning, Classifier Output Difference, Navïe Bayes, Radial Basis Function Network, Clustering, Algorithm Selection Model

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to the No Free Lunch Theorem [85, 86], it is impossible to build an algorithm that performs optimally for all tasks. As we carry out machine learning experiments, we find, as expected, that each learning algorithm has its own strengths and weaknesses.[1] Single-layer neural networks, for example, are known to be weak on non-linearly-separable data, while Fisher's linear discriminants perform poorly on data where target values have the same mean. Similarly, decision trees require deep trees for non-linearly-separable data, and they are known to be inferior to multilayer neural networks on continuous data. Overall, however, our understanding and interpretation of behaviors on many algorithms remain limited.

This lack of knowledge of the behavior of algorithms prevents us from the optimal use among available algorithms. Even though some of the weakness are known, the interaction between the algorithms' mechanisms and the characteristics of learning tasks that are relevant to learning is not fully understood.[2] This is mainly due to the fact that our efforts tend to focus on designing new algorithms or extensions to existing algorithms that address known limitations. Yet, understanding algorithms and learning tasks, as well as similarity among algorithms in terms of behavior, has a profound impact on both experts and non-experts. Since new algorithms are continually developed and commercialized over time, the pool of suitable algorithms for any specific task keeps growing. This increases the

[1] To avoid redundancy and confusion, the term *learning algorithm*, or simply *algorithm*, will be understood to mean *classification learning algorithm* throughout this document.

[2] The term *learning task*, or simply *task* will be understood to mean the problem to be solve by a learning algorithm. Data or datasets are physical representations of an associated task.

chances of finding good algorithm for learning tasks, but it also requires practitioners to spend additional time to pinpoint such an optimal algorithm for their tasks. This is known as the algorithm selection problem.

The metalearning community has made some strides in this area, particularly in attempting to build automatic systems (i.e., algorithm selection models) that predict the best algorithm for a given task. Despite some progress, much remains to be done. This dissertation is another step in that direction, where we focus on gaining insight into the behavior of learning algorithms, discovering similarity among them, and leveraging such similarity to improve the accuracy of the resulting algorithm selection model. The dissertation is organized as follows.

The characterization of the training examples at the metalevel (i.e., base-level datasets) plays a crucial role in metalearning. In particular, the metafeatures used must have some predictive power.[3] Selecting appropriate features is by no means trivial. As pointed out by Rice, "The determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem" [63]. Logically, we began our investigation of metalearning by revisiting most of the metafeatures typically used in the context of metalearning for model selection. Using visual analysis and computational complexity considerations, we found four metafeatures whose values are directly relevant to certain ranges of predictive accuracy for seven learning algorithms on 135 UCI datasets. The results are presented in Chapter 3 (published in the *Proceedings of the International Conference on Machine Learning and Applications*, 2008).

While general conclusions could not be reached, a closer examination between easy and hard tasks allowed us to isolate two metafeatures whose values are fairly distinguishable. This, in turn, caused us to devise a new metafeature whose purpose is to measure the difficulty of a learning task that is independent of the performance of learning algorithms on it. These results and our new metafeature are presented in Chapter 4 (published in the

[3]While a feature in a base-level dataset represents some aspect of the associated learning task, a metafeature represents a general characteristic applicable across learning tasks.

Proceedings of the International Conference on Machine Learning and Applications, 2008). That study also led us to investigate more formally how we might measure the behavior of algorithms at a finer grained level than a simple dichotomy between easy and hard tasks. Rather we wanted a measure of similarity in behavior for learning algorithms. We turned our attention to the Classifier Output Difference (COD) measure.

Following the very preliminary but promising results of Chapter 4, we performed a formal review and analysis of most popular measures of diversity for learning algorithms, and proved that only COD had the properties of a metric. This provided us the necessary theoretical backing to perform clustering learning algorithm to understand the diversity of learning algorithms. We thus produced a clustering of 21 learning algorithms, showed how it differed significantly from a clustering based on accuracy, and how it can be used to highlight interesting, sometimes unexpected, similarities among learning algorithms. Details are in Chapter 5 (to appear in *Intelligent Data Analysis Journal*, 2011).

In Chapter 6 (submitted to Machine Learning Journal, 2011), we illustrate one of the side-effects of our clustering of learning algorithms by providing a thorough comparison of two "unexpectedly" close learning algorithms: Naive Bayes and Radial Basis Function Network. Using both analytical tools and empirical results, we show that there is a significant amount of similarity between their Weka implementations for a broad range of datasets for small numbers of kernels. We further show that the Weka's implementations are reasonable, and that a larger number of kernels is typically not useful. Hence, the observed similarity, when applicable, is of practical import. In particular, since radial basis function network learning is significantly more computationally expensive than Naive Bayes learning, we use metalearning to build a selection model capable of accurately discriminating between the two algorithms. By doing this, extra computation is only incurred when it is guaranteed to produce significant improvement in predictive accuracy.

In Chapter 7 (submitted to International conference on Machine Learning, 2011), we illustrate another one of the side-effects of our clustering of learning algorithms by showing

how traditional algorithm selection can be effectively replaced by cluster selection. Not only are the results as good as, or better than those obtained by algorithm selection methods, but the approach lends itself more naturally to the online addition of new algorithms, thus opening the way for incremental metalearning.

Chapter 2

Thesis Statement

In order to gain insight into the behavior of algorithms and their interaction with tasks, we employed a metalearning technique. This reveals the relationship of different but behaviorally-similar algorithms and contributes to build an alternative algorithm selection model.

Chapter 3

Predicting Algorithm Accuracy with a Small Set of Effective Meta-Features

Abstract

We revisit 26 meta-features typically used in the context of meta-learning for model selection. Using visual analysis and computational complexity considerations, we find 4 meta-features whose values are directly relevant to certain ranges of predictive accuracy for 7 learning algorithms on 135 UCI datasets. Discretization of these 4 meta-features based on thresholds derived from our analysis significantly boosts the accuracy of the meta-level classification task.

3.1 Introduction

The No Free Lunch theorem states that there is no single superior learning algorithm that performs best across all learning tasks [85, 86]. As a consequence, it becomes important for researchers and practitioners to discover and implement mechanisms that may determine which machine learning algorithms perform best on which tasks. For over a decade, meta-learning researchers have put a significant amount of effort in finding functions mapping learning tasks to their corresponding optimal machine learning algorithm. In the StatLog project and its successor, the METAL project, a relatively large number of meta-features were devised in the hope that they (or at least some of them) might reflect some hidden learning task properties related to the performance of specific machine learning algorithms. These projects were successful in that researchers discovered some approximate relationships between meta-features and which machine learning algorithms were likely to perform best on the associated learning tasks. However, most meta-features were generated in an *ad hoc* way, without any evaluation of their relevance to the performance of individual machine learning algorithms. The actual properties and usefulness of these features remain largely unknown.

In this paper, we report on our analysis of the usefulness and complexity of 26 of the most popular meta-features. The result of such an analysis is critical for at least two important reasons:

1. Generating meta-features is often non-trivial and may be as expensive as just running a machine learning algorithm, which would clearly defeat the purpose since meta-features are meant to serve as surrogates for algorithm performance results; and
2. Information on meta-features that are highly relevant to the predictive accuracy of specific learning algorithms can provide valuable insight into these learning algorithms and relationships among them.

We examine each meta-feature one by one according to its relevance to the predictive accuracy of 7 well-known learning algorithms over 135 UCI datasets. Visual analysis reveals 4

meta-features with acceptable computational complexity and strong correlation with learning algorithm accuracy. We derive simple binary meta-features from these original meta-features and show that:

1. Random but controlled artificial datasets explicitly designed to satisfy the conditions embedded in the converted meta-features do indeed give rise to models with the expected behavior, and
2. When used for algorithm selection at the meta-level, where one learns to predict algorithm accuracy from learning task meta-features, the converted meta-features give rise to meta-models with significantly higher accuracy.

The remainder of the paper is organized as follows. In Section 2, we briefly review some related work. In Section 3, we formulate our target meta-learning problem and describe our experimental procedure. Section 4 shows how we discover our small subset of useful and efficient meta-features. Section 5 contains experimental results and section 6 concludes the paper.

3.2 Related Work

Much work has been done in designing and evaluating meta-features in meta-learning research, generally focusing on the algorithm selection task. The first such results were generated as part of the European StatLog project [51], which identified 16 meta-features and used them in an attempt "to relate performance of algorithms to characteristics or measures of classification dataset." [17]. The European METAL project [50] extended StatLog to cover more learning algorithms and more datasets, and investigated a number of other meta-features (e.g., [9, 56, 59]). Both projects sought to map meta-features to either a best performing algorithm or to a ranking of algorithms [18]. Neither StatLog nor METAL spent much time analyzing the individual relevance of the meta-features they used. A very recent survey offers both a nice review as well as an unifying framework for meta-learning, as well

as an explicit recognition that one must be weary of generating meta-features that are more costly to compute than it would be to follow a brute force approach wherein one runs the target learning algorithms and selects the best one [70].

Interestingly, most of the work in meta-learning has focused on algorithm selection or ranking. Comparatively little has been done in trying to predict algorithm performance, as we do here. Notable exceptions include the early (somewhat unsuccessful, at least in terms of performance) attempt of [29], as well as more recent and more successful results in [38, 73]. While their focus was on regression models using raw meta-features, we discretize accuracy leading to a classification model and spend time analyzing each meta-feature in detail.

One attempt at gaining some insight into the values of meta-features relative to algorithm performance is in [69]. This paper shows the result of clustering 57 problems based on 21 meta-features (statistical and information theoretic measures over the datasets) using self-organizing maps. Overlaid on each cluster is the relative performance of 6 learning algorithms. The authors then draw simple conclusions such as, cluster C_x contains tasks that seem to have these characteristics and are best solved by these learning algorithms. Unfortunately, the performance of the algorithms is averaged over the cluster, yet there is huge variance in the values. The same is true of the input features. Hence, although this seems like a good idea, the results are not very reliable and much more data and analysis would be required to get to something more actionable.

Finally, we mention the recent proposal of the design, implementation and maintenance of experiment databases [14]. Such databases would be public and serve as repository for machine learning experiments so that a complete account of experimental procedures and parameters would be available to the community. Although not directly related to the work here, the existence of such a database would greatly enhance our ability to conduct meaningful analyses of learning at the meta-level.

3.3 Problem Formulation

Our main objective is to find important meta-features among candidate meta-features. Here, important meta-features mean features demonstrating high relevance to predictive accuracy for some (set of) machine learning algorithm(s). The problem can be formulated as follows.

Given:

a set of datasets $D = \{d_1, \ldots, d_{|D|}\}$

a set of learning algorithms $M = \{m_1, \ldots, m_{|M|}\}$

a set of meta-features $F = \{f_1, \ldots, f_{|F|}\}$

Find:

$F^{'} \subseteq F$ s.t. $\forall k \in M, m_k(G(F^{'})) > m_k(F)$

with $G(F^{'}) = g_1(f_1^{'}) \cup \ldots \cup g_{|F^{'}|}(f_{|F^{'}|}^{'}) \cup (F - F^{'})$, where each g_i effects some transformation on its input, and $m_k(F)$ is the accuracy of m_k on the meta-dataset defined by the meta-features in F.

As we started our investigation, we thought we might be able to find direct linear relationships between meta-features and accuracy, i.e., the above would hold when g_i is the identity function (i.e., no transformation). However, no such relationship was found. As we will show in the next section, it is possible to find step-like relationships so that the g_i's may be viewed as thresholding functions. Hence, for a meta-feature $f_i^{'}$, $g_i(f_i^{'})$ returns categorical values, each of which represents some interval of predictive accuracy for some learning algorithm m_k. For example, as we will see in Section 4, the g function for mean mutual information returns A when its corresponding input is greater than or equal to 0.18 and B otherwise. A and B are the so-called converted meta-features.

For our experimentation, we must decide on the sets D, M, and F. The set M consists of seven well-known learning algorithms, each representing general classes of learning paradigms (e.g., rule-based learning, neural learning, probabilistic learning, etc.): C4.5 decision tree (DT), multilayer perceptron (MLP), support vector machine (SVM), naive Bayes

(NB), K-nearest neighbor (KNN), radial basis function network (RBF), and RIPPER. We use the default Weka implementation of all of these algorithms. The set F contains 26 meta-features that are mostly derived from the StatLog project, the METAL project, and [2]. They include simple measures, statistical measures, and information theoretical measures. Statistical measures are applicable to continuous attributes while information theoretical measures are suitable for categorical attributes. We do not list all 26 features here as many will be discussed in the next section. Finally, the set D consists of 135 data sets from the UCI Machine Learning Repository [3], which cover most of the classification datasets currently available.

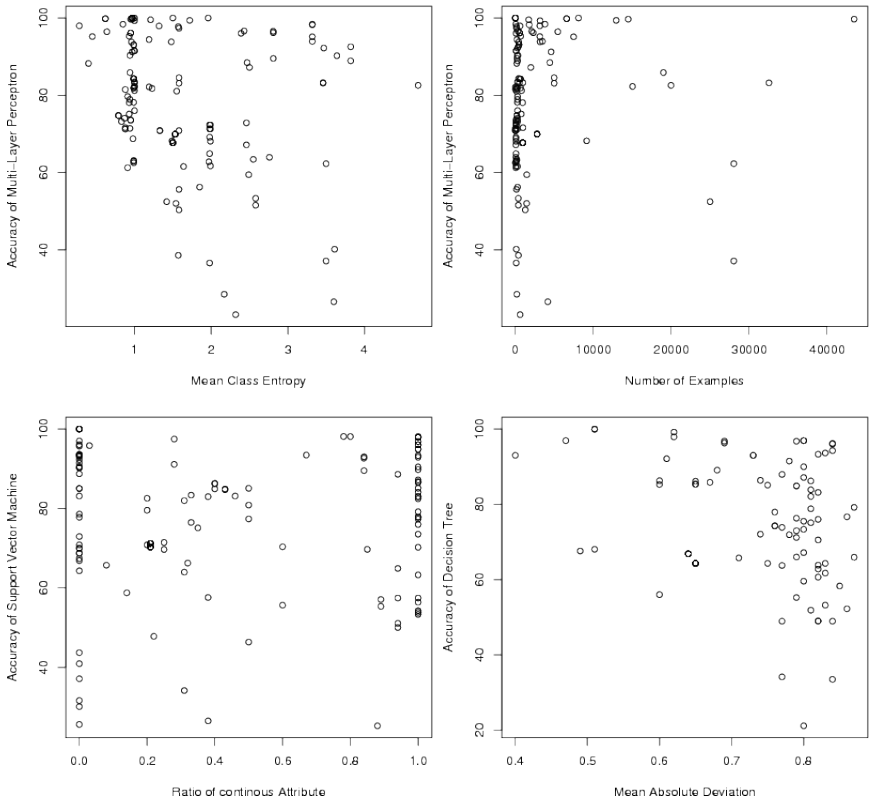
After selecting the 135 UCI datasets, we compute the values of the meta-features in F, together with the 10-fold cross-validation accuracy of the 7 learning algorithms in M, for each dataset. We then construct a meta-dataset where each row i consists of the values derived for each dataset i. Visual analysis leads to the identification of the subset F' of meta-features and the design of the associated g functions. Finally, we build and compare the performance of meta-models obtained from the meta-features in F' and F, using the algorithms in M as meta-learners.

3.4 Meta-feature Subset Selection

Upon visual analysis of the scatterplots, we find that a significant number of individual meta-features are very weakly related to the predictive accuracy of learning algorithm. We are aware that the combination of meta-features can be correlated with the predictive accuracy of some learning algorithm even when the individual meta-features do not reveal it. In this sense, our attempt at looking at individual meta-feature can be limited, but it turns out that even individual examination is still worthwhile.

Figure 3.1 shows a few representative meta-features that are seemingly unrelated to accuracy. For the mean class entropy plot on the top left, all points are spread fairly uniformly across the accuracy of neural network. Even though lots of data points whose

Figure 3.1: Random Meta-Features



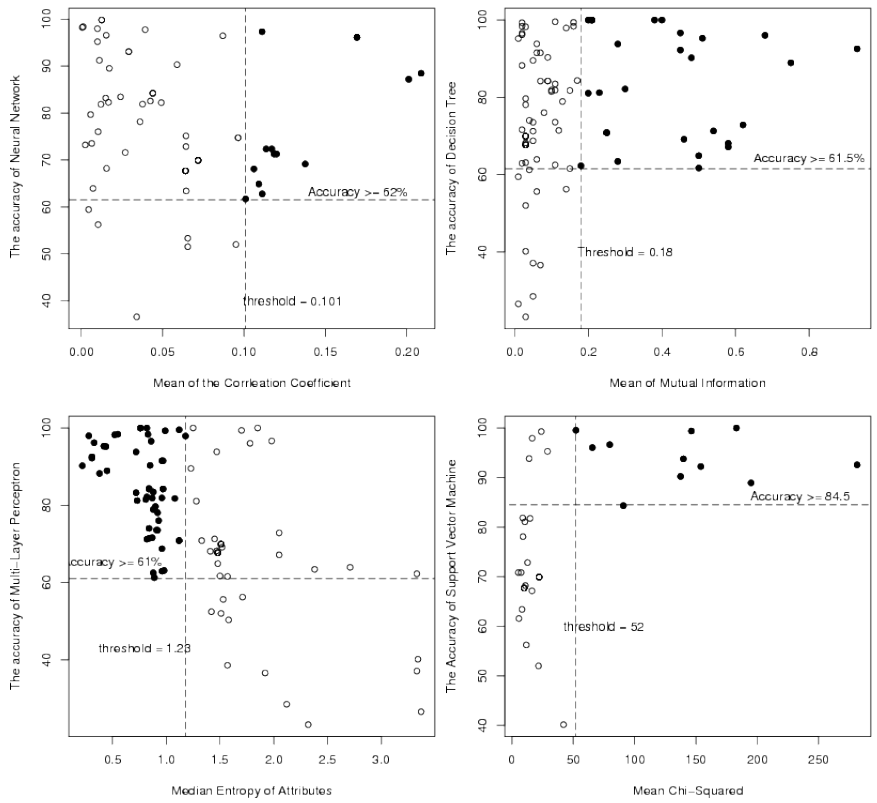
class entropy is between 1 and 3 are clustered in the 50% to 100% accuracy region, there is no clear boundary to draw to separate them. A similar situation may be observed for the other plots.

Although most meta-features exhibit patterns similar to the above across most or all learning algorithms, 4 meta-features stand out and reveal a relatively strong relation with the predictive accuracy of all learning algorithms. A few representative plots are shown in Figure 3.2.[1]

Solid dots indicate datasets whose accuracy is in the range of specific values of associated meta-feature. Regarding the correlation coefficient plot, located at the top left, it is the mean correlation coefficient for any two continuous attributes. Datasets whose correlation coefficient is greater than 0.101 tend to have accuracies over 60% with MLP. We

[1] All 26 meta-features on all algorithms are available at http://dml.cs.byu.edu//wiki//index.php//Jun_won_lee//icmla08.p

Figure 3.2: Accuracy-related Meta-Features



obtained very similar patterns across all learning algorithms. On the right hand side is the mean mutual information meta-feature with C4.5. It is calculated as the mean of the mutual information for each discrete attribute and target class. Datasets whose mutual information with target is above 0.18 tend to have accuracies over 61.5% with C4.5. For the median entropy of attribute at the bottom left, its value is acquired by taking a median of entropies of all attributes. Datasets whose median entropy is less than or equal to 1.28 tend to have accuracies above 60% with MLP. Finally, the χ^2 meta-feature is shown on the bottom right. Its value is calculated by averaging across the χ^2 of discrete attributes and target class. Data sets having a value above 52 tend to have accuracies above 84.5% with SVM. This threshold is valid across all seven learning algorithms.

From each of these four meta-features, we create converted meta-values: the one above threshold and the one below threshold. Before verifying the effect of these converted

meta-features in terms of the improvement in meta-learning, we note that the complexity of mCC is the same as ID3, while the other meta-features are cheaper. If the choice of experimental learning algorithm is ID3, then mCC may not be the best choice. However, the other selected meta-features are effective in terms of time complexity and relevance to predictive accuracy. As many learning algorithms are also far more costly than ID3, our set of meta-features, including mCC, remains generally widely applicable.

3.5 Experimental Results

We have identified four meta-features that demonstrate a strong relation with learning algorithms and converted each of them to a corresponding thresholded meta-feature based on the distribution with accuracy values. In this section, we investigate the impact of each converted meta-features and how these are related with meta-learning.

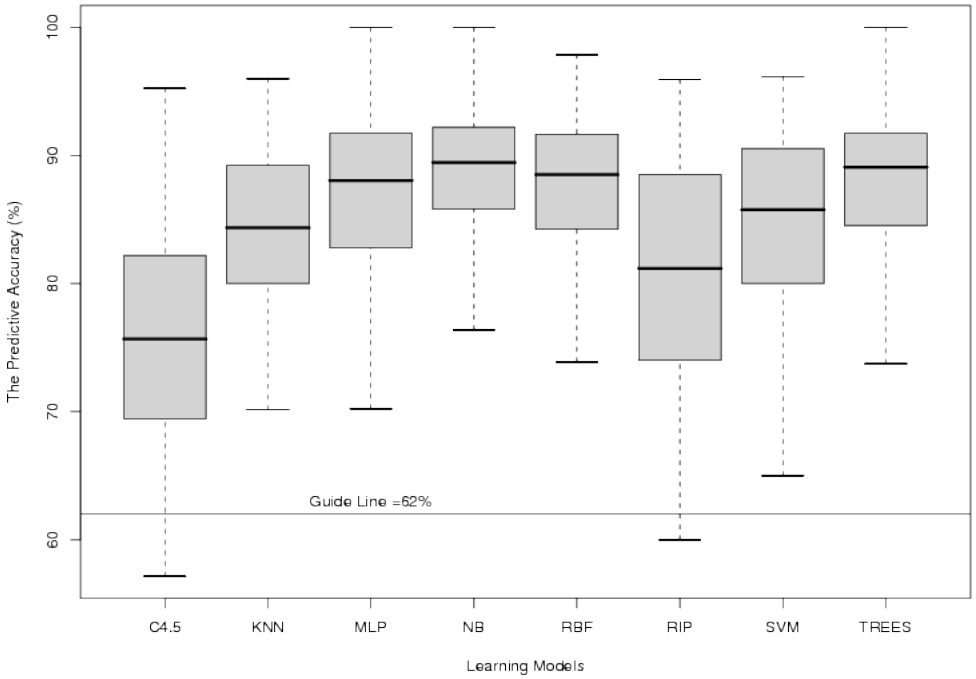
3.5.1 Influence of Converted Meta-features

In Section 4, we created fixed thresholds for specific learning algorithms to generate the converted meta-features. By doing this, we are able to distinguish what may be viewed as good datasets (i.e., those on which learning algorithms perform well) from bad datasets, based on the range of meta-feature values. However, as we mentioned earlier, there are still chances that the derived thresholds may not hold on some unseen datasets. Therefore, it is necessary to see the degree of robustness of our suggested thresholds.

We build 100 synthetic datasets with converted meta-feature constraints and test them over diverse learning algorithms. The experiment provides a direct relationship between converted meta-features and "good" datasets. Our synthetic datasets are generated by a 48-bit seed random number generator, which uses a linear congruential formula. Initially, we randomly set the number of attributes, the number of instances, and the number of target classes. Then, for each attribute, we generate its corresponding data instances randomly. These random values are repeatedly re-generated until the conditions on the selected meta-

features are satisfied. Because it is very hard to generate synthetic datasets randomly which satisfy any condition on mχ^2, it was left out of the experiment. Hence, the results are for datasets that satisfy conditions on only the other three converted meta-features. For our experiment, the average number of attributes is 7 and the average number of instances is 53.25. Figure 3.3 shows the box-whisker-plot of the average performances of 8 learning algorithms on these datasets.[2] All predictive accuracies are obtained with 10-fold cross-validation.

Figure 3.3: Performance of Synthetic Datasets



About 89% predictive accuracy on average is obtained by TREES, NB, and RBF, while about 86% predictive accuracy is obtained by MLP, SVM, and KNN. As for C4.5 and RIP(PER), their average accuracies are around 75% and their standard deviations are relatively larger than the other models. However, even the minimum accuracies for C4.5 and RIP are about 60%, and their first quartiles are 69% and 75%, respectively. This seems to suggest that datasets near a minimum data point are very rare. Hence, Figure 3.3 demon-

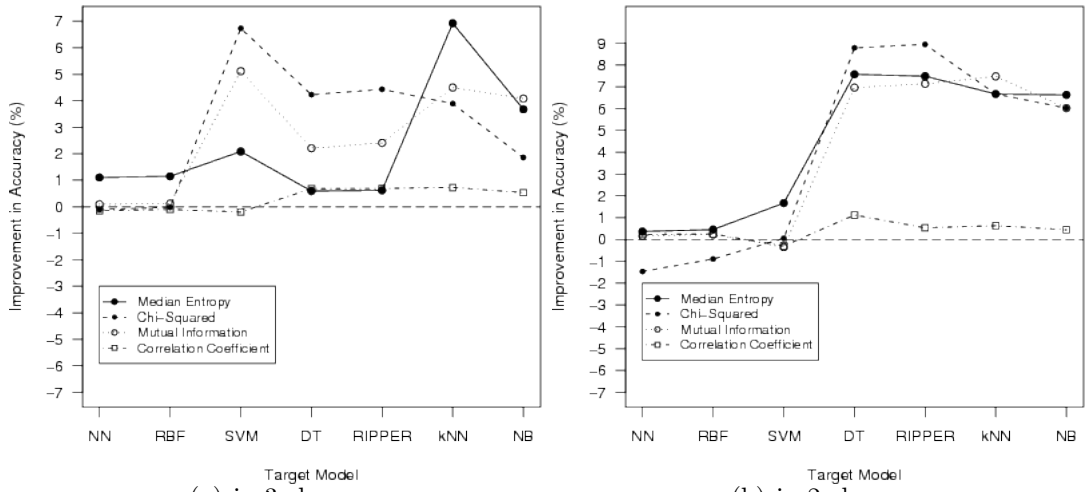
[2]The added learner TREES is a decision tree learning algorithm with NB classifiers at the leaves.

strates that our thresholds for converted meta-features are indicative of expected positive performance, i.e., accuracy above the threshold (here 62%) for the selected algorithms.

3.5.2 Meta-learning

We here turn to the performance of our converted meta-features at the meta-level. In particular, we test the effect of the converted meta-features by comparing the performance of predicting accuracy for each meta-learning algorithm from a meta-dataset represented with and without the converted meta-features (see Section 3). In addition, to create a meta-level classification problem, we discretize the target class, i.e., the predictive accuracy of base learners, into either 2 values (above 80%, below 80%) or 3 values (above 80%, between 60% and 80%, below 60%). The results are in Figure 3.4.

Figure 3.4: Meta-Model Accuracy Improvement



(a) in 3 classes (b) in 2 classes

The x-axis corresponds to each of the seven learning algorithms. The y-axis indicates the average improvement in accuracy from a meta-feature set including no converted meta-features to a meta-feature set including the corresponding converted meta-feature. It seems that all converted meta-features induce some improvement across most learning algorithms, with only small losses for MLP, SVM and RBF. The mCC meta-feature shows relatively

weak improvement (up to only about 1.1%) compared to the other meta-features. The best result are obtained with DT for two-class discretization, where the accuracy is improved by about 9%. These results strongly suggest that the use of our 4 converted meta-features generally improves the ability to predict (at the meta-level) the predictive accuracy of diverse learning algorithms.

3.6 Conclusion and Future Work

In this paper, we evaluate well known meta-features mainly derived from the METAL project and [2]. By examining a list of meta-features, we distinguished four that are most relevant in terms of the relation with the predictive accuracy of various learning algorithms. According to the experimental results, mean correlation coefficient of attributes, median entropy of attributes, mean χ^2 of attributes, and mutual information between attributes and target class turn out to be meta-features that are directly relevant with high performance of our seven test-bed learning algorithms. With those meta-features, we are able to formulate converted meta-features to boost meta-learning performance.

Clearly, the combination of several meta-features can be highly relevant to the predictive accuracy even when single meta-features do not reveal any relevancy, but identifying these meta-features (especially, when the number of meta-features is large) is costly. Finding a way to do this to see the impact of several combinations of meta-features is left as future work. Additionally, there are more meta-features than those studied here. We will further investigate the relevance of those other meta-features.

Chapter 4

New Insights Into Learning Algorithms and Datasets

Abstract

We report on three distinct experiments that provide new valuable insights into learning algorithms and datasets. We first describe two effective meta-features that significantly impact the predictive accuracy of a broad range of learning algorithms. We then introduce a new efficient meta-feature that measures the degree of hardness (or difficulty) of datasets and show that it is highly linearly correlated with predictive accuracy. Finally, we use the notion of COD[58] that measures the (dis)similarity of behaviors between algorithms to cluster learning algorithms and show that learning algorithms from the same model class do not necessarily exhibit similar behaviors.

4.1 Introduction

At the base level, (supervised) machine learning is concerned with approximating a function between input and output from a set of training examples. It is well known that each learning algorithm produces a hypothesis based on its underlying structural and algorithmic biases. For example, a decision tree learning algorithm adopts some variant of information gain and a tree structure to build a hypothesis, while a neural network learning algorithm constructs hypotheses based on non-linear relationships among inputs and a network-like structure. According to the No Free Lunch Theorems, each learning algorithm can learn effectively over only a limited number of tasks [66, 86, 85]. In other words, each learning algorithm is inherently "optimized" for a specific subset of learning tasks.

While the machine learning community has put significant efforts into building effective learning algorithms tailored to their specific problems, our understanding and interpretation of behaviors among these learning algorithms are still very limited. To address this shortcoming, several researchers have focused their attention on metalearning, where they study "methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes" [16]. In other words, they seek connections between learning tasks and learning algorithms, e.g., which learning algorithms are superior on which learning tasks (algorithm selection problem), how to learn a new problem using prior knowledge of similar tasks (transfer learning), etc. This meta-level knowledge increases our understanding of the nature of base-level learning algorithms and in turn our ability to apply machine learning more effectively and extensively.

In this paper, we add to the existing body of metaknowledge by answering the following three questions:

- What typical properties of datasets have the strongest impact on learning?
- What effective measurement on data can capture the difficulty of learning?

- Which learning algorithms *behave* the same or differently, and how similar (or dissimilar) are they?

We focus our attention on those datasets that correspond to classification tasks in the UCI repository [3]. The remainder of the paper is organized as follows. In Section 2, we briefly review some related work. In Section 3, we highlight metafeatures that are significantly correlated to the accuracy of learning algorithms. Section 4 describes one specific metafeature we call hardness, which measures the degree of difficulty of learning. In section 5, we use clustering to group a large number of learning algorithms based on COD in two ways, which capture the degree of behavioral similarity and dissimilarity, respectively. Finally, section 6 concludes the paper and provides directions for future work.

4.2 Related Work

This section briefly reviews the related work on meta-learning that focused on understanding algorithm and data, and differentiates this work from others studying related concepts.

A significant amount of work has been done in designing and evaluating meta-features in meta-learning research, generally focusing on the algorithm selection task. The first such results were generated as part of the European StatLog project [51], which identified 16 meta-features and used them in an attempt "to relate performance of algorithms to characteristics or measures of classification dataset." [17]. The European METAL project [50] extended StatLog to cover more learning algorithms and more datasets, and investigated a number of other meta-features (e.g., [9, 56, 59]). Both projects sought to map meta-features to either a best performing algorithm or to a ranking of algorithms [18]. Neither StatLog nor METAL spent much time analyzing the individual relevance of the meta-features they used.

One attempt at gaining some insight into the values of meta-features relative to algorithm performance is in [69]. This paper shows the result of clustering 57 problems based on 21 meta-features (statistical and information theoretic measures over the datasets) using self-organizing maps. Overlaid on each cluster is the relative performance of 6 learning

algorithms. The authors then draw simple conclusions such as, cluster C_x contains tasks that seem to have these characteristics and are best solved by these learning algorithms. Unfortunately, the performance of the algorithms is averaged over the cluster, yet there is huge variance in the values. The same is true of the input features. Hence, although this seems like a good idea, the results are not very reliable and much more data and analysis would be required to get to something more actionable.

More recently, results were reported on clustering algorithms based on the similarity of correlation distributions of pairs of algorithms across many datasets [39]. A characterization of each cluster is derived from meta-features obtained from the datasets. This approach is somewhat similar to ours. It does differ, however, in that the clustered algorithms are homogeneous, i.e., they come from the same model class (e.g., they generate a list of six neural-based algorithms that have different input parameters, number of hidden nodes and internal layers). The clustering presented here covers 26 heterogeneous learning algorithms, which reveals unexpected results about behavioral similarity and dissimilarity across model classes.

4.3 A Pair of Dataset Properties That Strongly Impact Learning

Among the UCI classification tasks, there are some tasks that appear relatively easy to learn, in the sense that most learning algorithms produce high predictive accuracies (above 90%) on them. These *easy* datasets include the well-known `Lenses`, `Congressional Voting Records`, `Zoo` and `Iris` classification tasks. On the other hand, there also appear to be tasks on which most learning algorithms suffer from poor predictive performance (below 50%). These *hard* datasets include the `Contraceptive Method Choice`, `Abalone` and `Teaching Assistant Evaluation` classification tasks. We suspect that this bi-modality is due to some hidden properties or internal structures of the data.

In an earlier set of experiments, we considered 26 of the most popular statistical and information-theoretic metafeatures and examined them in turn to see whether any of them

exhibited significant differences between easy and hard tasks. Although no obvious pattern could be found for most of these, we were able to isolate two metafeatures whose values are fairly distinguishable between easy and hard groups.[1] These metafeatures are `mean mutual information of attributes and target` (mMI) and `mean attribute entropy` (maEnt). Table 4.1 shows the values of mMI and maEnt on some of the easy and hard tasks listed above.

Table 4.1: Value of mMI and maEnt for Some Typical Easy and Hard Learning Tasks

Task		**mMI**	**maEnt**
Easy	Lenses	0.36	0.98
	Congress. Voting Records	0.34	0.72
	Zoo	0.45	1.14
Hard	Contra. Method Choice	0.03	1.51
	Teaching Assist. Evaluation	0.06	1.53

The values in Table 4.1 suggest that easy tasks have relatively low maEnt and relatively high mMI. Intuitively, this makes sense. Low maEnt indicates that associated values are relatively well discerned and high mMI shows that the attributes are highly dependent with on the target class. Datasets that satisfy these conditions can indeed be be expected to be easier to learn. The opposite is true of hard tasks. If maEnt is high, then attribute values are not well discerned, and low mMI indicates that attributes and target class are fairly independent, making learning relatively harder then the previous case.

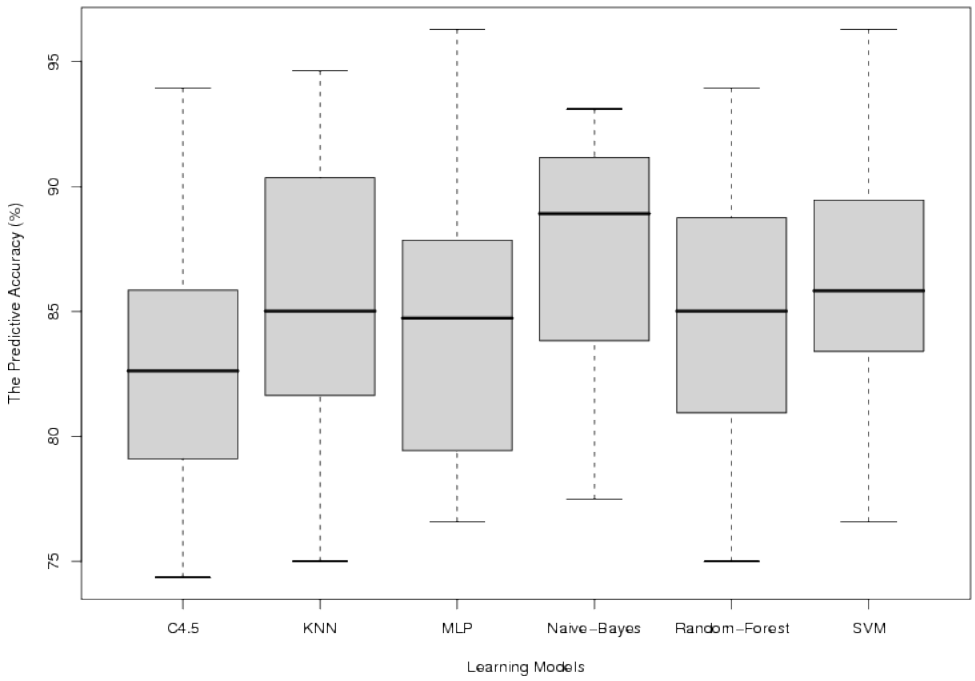
To confirm further the real effect of these two metafeatures, we built 20 random synthetic datasets satisfying low maEnt (i.e., less than 1.0) and high mMI (i.e., greater than 0.18), which we expect to be easy to learn, and 20 random synthetic datasets satisfying high maEnt (i.e., greater than 1.5) and low mMi (i.e., less than 0.06), which we expect to be

[1] We actually isolated a third discriminatory metafeature, the `mean correlation coefficient between attribute and target class`. However, the cost of computing this feature is as high as that of running many learning algorithms (e.g., decision tree), which defeats the purpose. Indeed, it has been argued, rightly so, that one should consider only efficient metafeatures, i.e., the time complexity of computing a metafeature should not exceed that of running any learning algorithm [59, 70]. Otherwise, one would be better off simply running the learning algorithms with no value added in examining metafeatures. Accordingly, we select only the two efficient metafeatures from our set of three candidates.

harder to learn. Due to the high computational cost associated with randomly generating datasets that meet such stringent requirements, we limited the size of the datasets to 70 examples.

Figure 4.1 shows the box-and-whisker plot of the performance of six well-known learning algorithms over the 20 (expected to be) easy synthetic datasets, while Figure 4.2 shows the same kind of plot over the 20 (expected to be) hard synthetic datasets. The results are as anticipated and confirm the strong impact of mMI and maEnt on learning. The average accuracy for the easy datasets is above 85%, and only between 40% and 45% for the hard datasets.

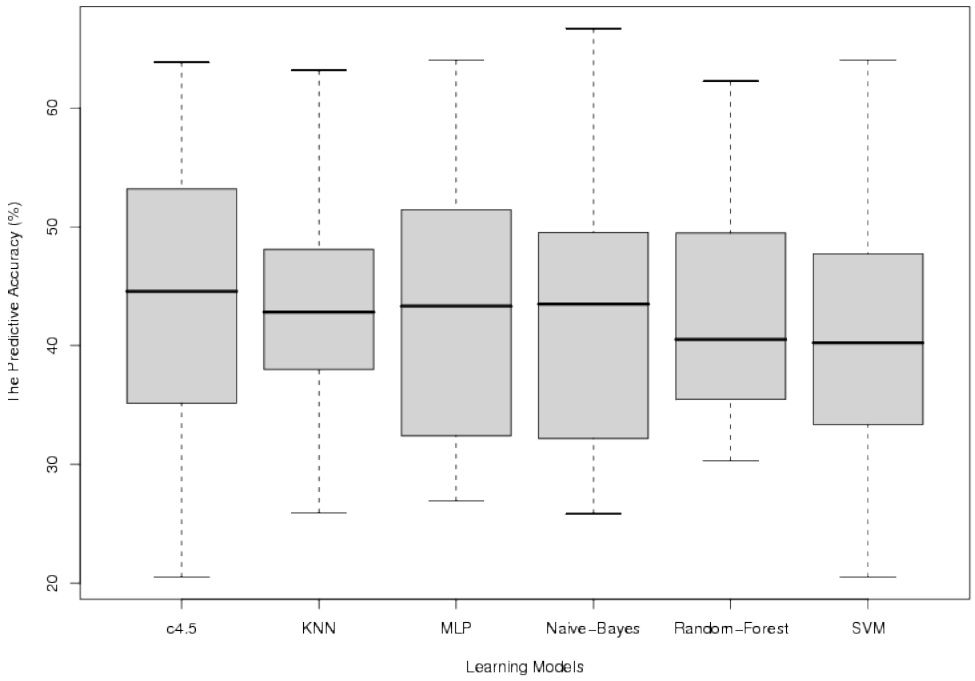
Figure 4.1: Accuracy of 6 Algorithms on 20 Datasets of mMI (≥ 0.18) and maEnt (≤ 1.0)



4.4 An Effective Measure of Hardness for Learning

Viewing each attribute as a dimension, the set of attributes of a dataset defines a hyperspace and each example in the dataset corresponds to a labeled (with its target classification) point in that space. For simplicity, let us restrict our attention to two dimensions and two target

Figure 4.2: Accuracy of 6 Algorithms on 20 Datasets of mMI (≤ 0.06) and maEnt (> 1.5)

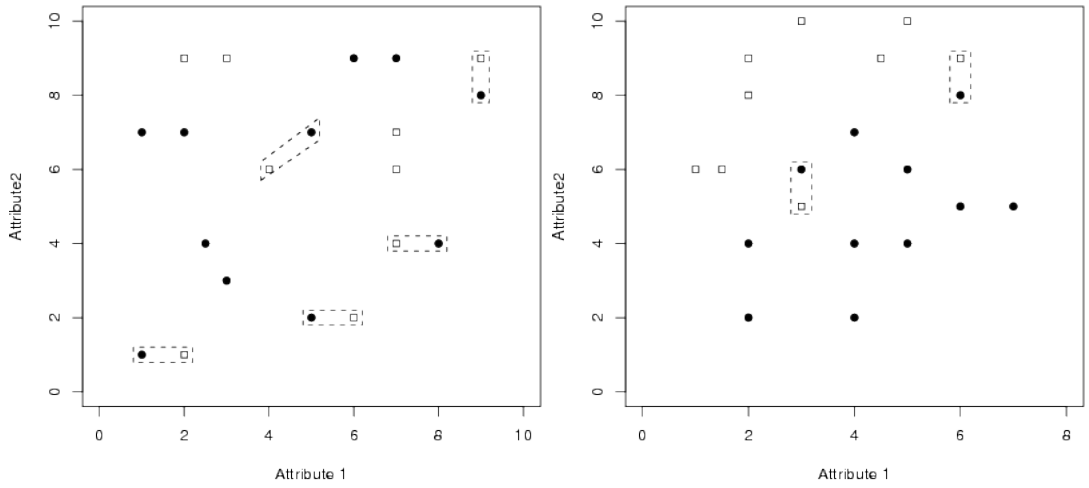


classes. Figure 4.3 shows two datasets, where the target classes are represented by a solid circle and an empty square square, respectively. For both datasets, a rectangular box has been drawn around each pair of data points that are nearest neighbors but have different target class values.

In Figure 4.3(a), the two target classes are rather mixed, which may indicate that learning this concept would be complex, since when differently labeled data points are closely mixed together, it is generally hard to draw a decision boundary. On the other hand, in Figure 4.3(b), the two classes of data points are rather well clustered and well separated, which suggests that learning the concept of this dataset would be much easier, since when data points with the same labels are relatively clustered together, a decision boundary can be drawn easily. This rather intuitive consideration provides the motivation for our *hardness* measure.

We define a dataset's hardness as the ratio of the number of examples whose nearest neighbor has a different target class value to the total number of examples. The hardness

Figure 4.3: Distribution of Examples for Two Illustrative Datasets



(a) Relatively hard to classify (b) Relatively easy to classify

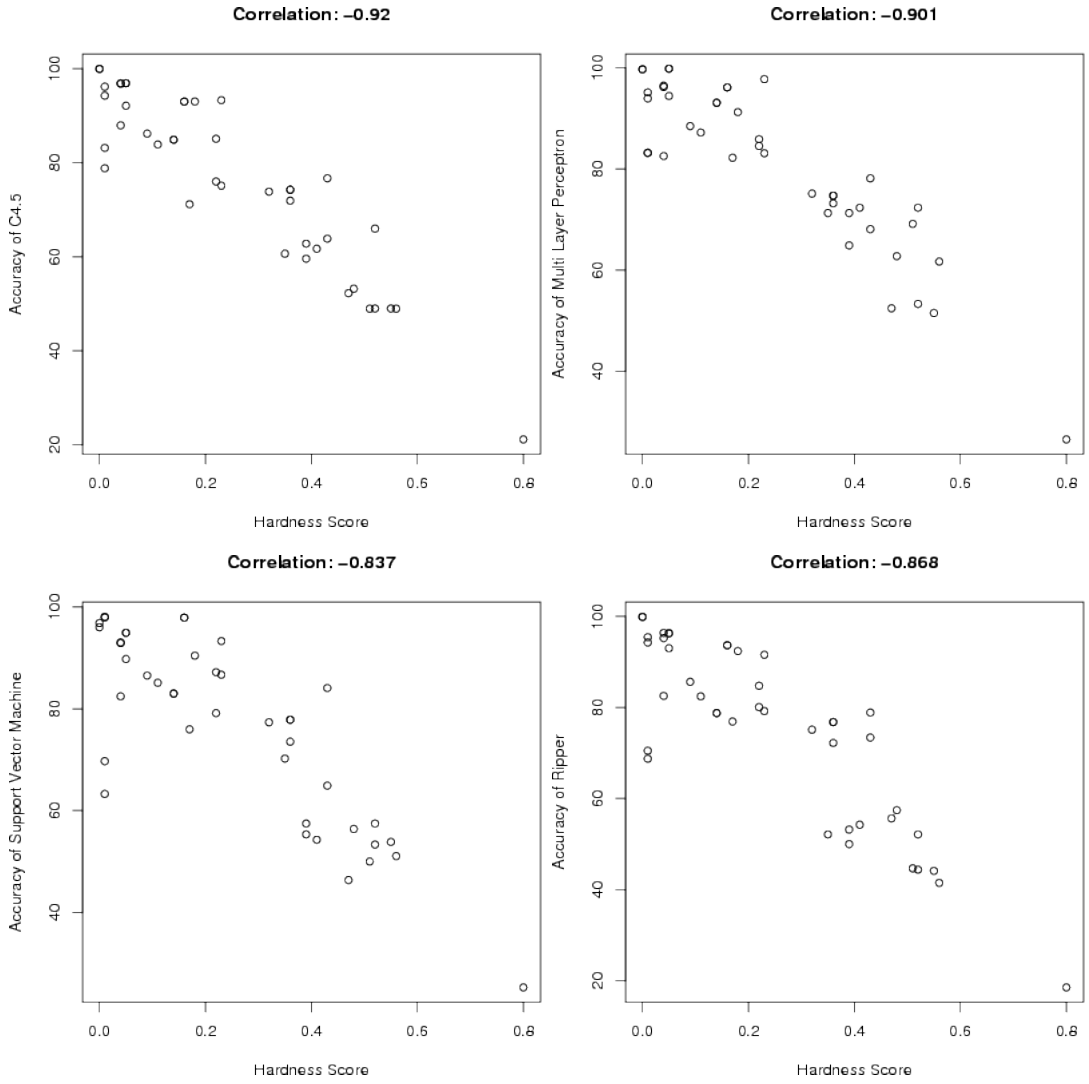
value of the dataset depicted in Figure 4.3(a) is 10/20=0.5, while the hardness value of the dataset depicted in Figure 4.3(b) is only 4/20=0.2.

Note that hardness is essentially the complement of the predictive accuracy of the 1-nearest neighbor algorithm (1-NN) on the dataset. In that sense, our use of hardness as a discriminatory metafeature shares the same motivation as the use of 1-NN as a landmark in landmarking metalearning [8]. Unlike landmarking where the performance of 1-NN is used as one of the metafeatures in building a model for algorithm selection, hardness is used here to understand the nature of datasets.

To ascertain the discriminatory power of hardness, we study the relationship between accuracy and hardness for several learning algorithms over a large number of UCI datasets. Figure 4.4 shows graphs of accuracy versus hardness for four learning algorithms. Similar graphs are obtained with other learning algorithms.

In all cases, there is a strong correlation between hardness and predictive accuracy. In addition, hardness can be implemented efficiently [78], making it an attractive measurement of the difficulty of learning prior to running typically more expensive learning algorithms.

Figure 4.4: Accuracy vs. Hardness for Several Learning Algorithms



4.5 An COD-Based Clustering of Algorithms

The COD (Classifier Output Difference) [58] is a distance tool for measuring the degree of similarity between two hypothesis generated from two algorithms. Formally, the COD between two learning algorithms A and B, denoted $COD_{A,B}$, is given by:

$$COD_{A,B} = P(\hat{f}_A(x) \neq \hat{f}_B(x))$$

where $\hat{f}_A$ (resp. $\hat{f}_B$) is the hypothesis or model induced by algorithm A (resp. B) from the training data. By definition, if the behaviors of two algorithms are opposite, then COD = 0. On the other extreme, if the behaviors of two algorithms are exactly the same, then COD = 1.

Here, we focus on clustering learning algorithms using COD. One of our contributions is in the wide range of algorithms and datasets under consideration. For our experiment, we included 26 classification learning algorithms from Weka [83] and 135 datasets from the UCI repository [3], yielding a total of 325 (= 26*25/2) error correlations of pairs of algorithms. Table 4.2 shows those algorithms that exhibited either the smallest (≤ 0.2) or largest (≥ 0.8) amount of COD among them, along with their model class, as per the Weka hierarchy.[2]

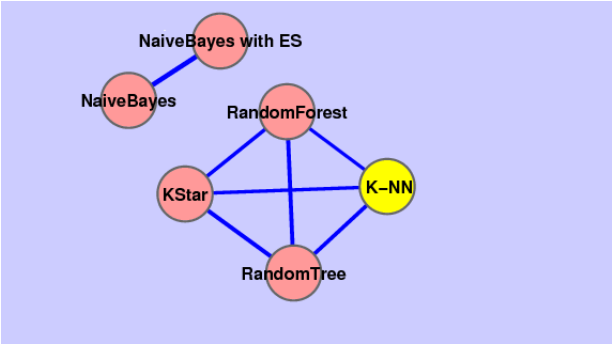
Table 4.2: Algorithms Exhibiting the Highest (or Lowest) Error Correlations

Algorithm	**Model Class**
NaiveBayes (NB)	bayes
NiaveBayes with ES (NB-ES)	bayes
RandomForest (RF)	trees
RandomTree (RT)	trees
DecisionStump (DS)	trees
KStar	lazy
LWL	lazy
IBk	lazy
OneR	rules
ConjunctiveRule (CR)	rules
ZeroR	rules
HyperPipes (HP)	misc
VFI	misc

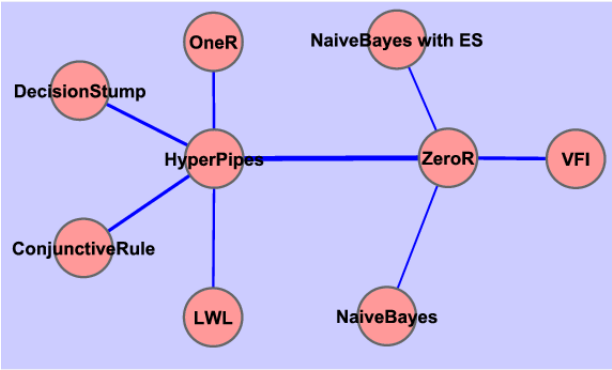
Figure 4.5 provides a graph-like representation of the algorithms in Table 4.2 for high and low COD. An edge between two algorithms indicates that COD is above (resp. below) the specified threshold, and the thickness of the edge is proportional to the strength (resp. weakness) of the COD. Actual values of the pairwise COD are found in Table 4.3.

[2]We obviously get diverse sets of algorithms depending on the threshold values we select for error correlation. The values 0.8 and 0.2 were chosen as they produce simple and distinguishable sets of algorithms.

Figure 4.5: Algorithm Clustering Based on Behavior Similarity and Dissimilarity



(a) COD ≤ 0.2 (Algorithms showing high similarity)



(b) COD ≥ 0.8 (Algorithms showing high dissimilarity)

Figure 4.5 reveals some unexpected patterns. Whereas one would probably expect that algorithms belonging to the same model class should exhibit higher COD, i.e., behave rather similarly, and that algorithms belonging to different model classes should exhibit lower COD, the data seems to suggest otherwise, at least in part.

Figure 4.5(a) shows two disconnected sets of most similar groups. The first group consists of NB and NB-ES, a small variation on NB. No surprise here. The second group, however, consists of a fully connected subgraph involving four learning algorithms, RF, RT, KStar, and IBk. Interestingly, two of these algorithms belong to the `trees` model class while the other two belong to the `lazy` model class (see Table 4.2). It is notable that RF and RT are not as similar to other well-known tree-based algorithms such as J48 and ID3 (according to our threshold) as they are to the two instance-based algorithms.

Table 4.3: Normalized Pairwise Error Correlation (EC) Scores

	Algorithm Pair		EC Score
Most Similar	NB	NB-ES	1
	RT	K-NN	0.989581
	KStar	K-NN	0.985074
	KStar	RT	0.975032
	RF	K-NN	0.974697
	RF	RT	0.954655
	KStar	RF	0.92151
Least Similar	HP	ZeroR	0
	ZeroR	VFI	0.101897
	HP	CR	0.125333
	HP	DS	0.129127
	HP	LWL	0.184132
	HP	OneR	0.173588
	ZeroR	NB	0.194024
	ZeroR	NB-ES	0.194024

Figure 4.5(b) shows a single, loosely connected graph of 9 learning algorithms. The graph does not show any strong dissimilarities within model classes, as expected. It turns out that HP and ZeroR are the most dissimilar algorithms. Apart from ZeroR, HP is also fairly different from two other rule-based learning algorithms (OneR and CR), one tree-based algorithm (DS), and one instance-based algorithm (LWL). On the other hand, ZeroR is different from NB, NB-ES and VFI.

We note in passing that, although Figure 4.5(a) is transitive, this is mostly an artifact of the high threshold value we selected. As the threshold is lowered, it becomes increasingly likely to find situations where algorithms A and B are similar, algorithms B and C are similar, but algorithms A and C are not (at that threshold level).

4.6 Conclusion

In this paper, we report on three experiments that help our understanding of the nature of datasets and learning algorithms. First, we highlighted a pair of meta features, `mean`

`mutual information of attributes and target class` and `mean attribute entropy`, that have a strong impact on learning performance. When the former is high and the latter is low, then one can expect that most learning algorithms (that we have considered here) would perform rather well; and when the conditions are reversed, most learning algorithms would perform poorly. We also introduced a novel measure, called hardness, that is basically the complement of the accuracy of 1-NN, to capture the difficulty of learning. Experiments do indeed suggest that hardness is highly linearly correlated with the predictive accuracy of many learning algorithms. Finally, we clustered diverse learning algorithms based on COD. Our experiments reveal a few interesting patterns. In particular, there are algorithms whose model classes are different yet they are more similar to each other than to their own variants.

As future work, we would like to analyze more theoretically the reasons why some heterogeneous learning algorithms behave similarly. In addition, we would like to probe the degree of robustness of COD-based clustering as we collect more datasets from diverse channels. As far as our hardness measure, we are conscious that it is subject to the curse of dimensionality since potentially non-relevant attributes may make two truly similar data points appear distant. A method to eliminate the least relevant attributes is discussed in [53], but it does run some risk of overfitting. Since this measure is not designed for improving classification accuracy, we search for an alternative way to improve current correlation with learning algorithms.

Chapter 5

A Metric for Unsupervised Metalearning

Abstract

We argue the value of unsupervised metalearning and discuss the attendant necessity of suitable similarity, or distance, functions. We leverage the notion of diversity among learners used in ensemble learning to design a distance function for the clustering of learning algorithms. We revisit the most popular measures of diversity and show that only one of them, Classifier Output Difference (COD) is a metric. We then use COD to produce a clustering of 21 learning algorithms, and show how this clustering differs from a clustering based on accuracy, and how it can be used to highlight interesting, sometimes unexpected, similarities among learning algorithms.

5.1 Introduction

Two main forms of learning are generally considered: supervised and unsupervised. In supervised learning, the task consists in discovering a mapping from a set of features, or independent variables, representing data items, to some target outcome, or dependent variable. The task is deemed *supervised* because the training data available to the learner includes both the data items and their associated, teacher-given target values. Classification and regression are typical examples of supervised learning. In unsupervised learning, the tasks consists in discovering a grouping of data items based on the set of features used to represent them. The task is deemed *unsupervised* because the training data available to the learner is restricted to the data items themselves. No outcome is specified; the learner must rely on some notion of similarity among data items to induce a meaningful grouping thereof. Clustering and segmentation are typical examples of unsupervised learning.

A mapping, in the supervised sense, can easily be viewed as a grouping of data items based on the values of the target outcome, i.e., one group per target value. Because a teacher provides the value of the outcome for each training data item, supervised learning may thus theoretically be used to induce arbitrary groupings. On the other hand, given a similarity measure, unsupervised learning can only induce a single grouping. In practice, however, supervised learning also induces a single grouping, namely that specified by the teacher-labeled training data items. The main difference is that, whereas supervised learning requires, and is constrained by, something external to the training data items, unsupervised learning relies solely on the information available in the description of the training data items. Unsupervised learning has proven useful in a wide variety of applications, either as the technique of choice or as a complement to supervised learning.

Our goal here is to raise unsupervised learning to the metalevel, i.e., unsupervised metalearning. Metalearning has been defined as the study of "methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes" [16]. Metalearning differs from base-level learning in the scope of adap-

tation. Whereas learning at the base level focuses on accumulating experience on a specific learning task (e.g., credit rating, medical diagnosis, mine-rock discrimination, customer segmentation, etc.), learning at the metalevel seeks connections among learning algorithms and/or tasks. In other words, at the metalevel, the data items of interests, i.e., those to learn from, are learning algorithms and tasks. Metalearning increases our understanding of the nature of base-level learning algorithms and in turn our ability to apply machine learning more effectively and extensively.

While a significant amount of work has been done in supervised metalearning, especially mapping tasks to learning algorithms (e.g., see [2, 18, 59, 60, 71]), little, if any, has been done in unsupervised metalearning. Yet, we argue that, as is the case at the base level, unsupervised learning may prove useful as a complement to supervised learning at the metalevel. For example, clustering tasks could facilitate transfer learning, and clustering learning algorithms based on their behavior may reveal interesting similarities, which could be further exploited to improve algorithm selection. One of the prerequisites for unsupervised learning, however, is the existence of suitable similarity measures, or reciprocally, distance functions.

In this paper, we focus our attention on measures of similarity for algorithm behavior, as a precursor to clustering learning algorithms. Over the years, through theoretical advances and experience with increasingly many applications across a wide range of domains, we have gained some valuable insights about the relative performance of a number of algorithms over certain types of tasks. For example, Naive Bayes is known to be optimal if the features are conditionally independent given the class, and backpropagation often outperforms decision tree learning if the features are continuous. Yet, our understanding and interpretation of behaviors among these learning algorithms remain rather limited. This is mainly due to the fact that our efforts tend to focus on designing new algorithms or extensions to existing algorithms that address known limitations. Relatively few researchers have attempted to generalize across algorithms and tasks to capture knowledge about the interaction between the mechanism of learning and the concrete contexts in which that mechanism is applicable.

And when they have, the focus has been on comparing behavior based on global measures of performance such as predictive accuracy or area under the ROC curve.

Here, we propose a finer-grained approach where behavior is analyzed at the data item-level using diversity measures. Unlike global measures, which provide only an idea of average performance over all data items, diversity measures capture local variations among data items. Historically, these measures have been almost exclusively studied in the context of ensemble learning, where one seeks to maximize diversity. Interestingly, though, a small change of perspective, specifically equating diversity to a distance measure, would seem sufficient to allow behavior-based clustering of learning algorithms. However, in order for clustering to be well-defined, such distance measures should actually be distance functions, or metrics. We revisit the most popular diversity measures and show that only one of them gives rise to a distance function. We subsequently contrast it to a global measure based on accuracy by clustering 21 learning algorithms and highlighting significant differences.

The paper is organized as follows. In section 5.2, we review previous work relevant to our analysis. In section 5.3, we present an overview of the most popular pairwise measures of diversity, analyze them in terms of their suitability as distance functions, and show that only one of them satisfies all of the requirements for a metric. We use the selected metric in section 5.4 to cluster a number of well-established learning algorithms, and contrast it to the clustering obtained by a global distance measure. Section 5.5 concludes the paper and points to further possibilities for unsupervised metalearning.

5.2 Related Work

One attempt at unsupervised metalearning, focused on tasks rather than algorithms, is described in [69]. There, the authors show the result of clustering 57 learning tasks based on 21 (meta)features, i.e., statistical and information theoretic measures over the corresponding datasets, using Kohonen self-organizing maps. Overlaid on each cluster is the relative performance of 6 learning algorithms (IBk, C4.5, PART, NB, OneR and KD). Simple con-

clusions such as, cluster C contains tasks that seem to have these characteristics and are best solved by these learning algorithms, can then be drawn. Similarly, the work in [81] describes a method for clustering time series based on extracted characteristics from these series. Essentially, instead of clustering the raw time series, it replaces them by a set of 13 characteristics, or metafeatures, and clusters this transformed data. The results show that higher quality clusters may be obtained this way than with the raw data.

Perhaps closest to ours is the work of [38], where results on clustering 45 pairs of algorithms based on their error correlation distributions are reported. As in [69], and although it applies to pairs of algorithms, the characterization of each cluster is derived from metafeatures obtained from the tasks to which the learning algorithms were applied. In particular, the authors identify metafeatures predictive of whether learning algorithms are likely to exhibit very high or very low error correlation. However, the metafeature values are not clearly distinguishable across clusters, and 2 of the 4 clusters look rather mixed according to their error correlation distributions. Furthermore, the 10 algorithms under study represent only 3 model classes. The analysis presented here complements and significantly extends these, as well as others (e.g., see [44, 58]).

There has been long-standing interest in the notion of diversity in machine learning, mostly due to work in ensemble learning and multiple classifier systems, where it has long been known that diversity is essential to improving accuracy (e.g., see [19, 20, 21, 24, 31, 42, 43, 54]). Over the years, a number of measures of algorithm diversity have thus been proposed, for both pairwise comparisons and non-pairwise comparisons. Interestingly, several of the same measures of diversity used in machine learning, or their reciprocal known as measures of association, have also been used extensively in the social sciences, where one often seeks to characterize the degree of agreement among, for example, expert labelers or survey respondents (e.g., see [5, 67]). Since we will use diversity for clustering, and thus as a measure of distance, we focus exclusively on pairwise measures here. Whereas others have often focused on comparing and analyzing relationships among various measures (e.g.,

see [22, 76]), we analyze them individually with respect to their suitability for clustering, i.e., whether they induce metrics.

5.3 A Metric for the Space of Learning Algorithms

The most obvious, and often used, distance measure between two learning algorithms consists of the absolute value of their difference in (predictive) accuracy (e.g., see [84]). Let A_1 and A_2 be two learning algorithms and acc_{A_1}, respectively acc_{A_2}, be the accuracy of A_1, respectively A_2, averaged over some number of learning tasks. We would define their accuracy distance AD by:

$$AD(A_1, A_2) = |acc_{A_1} - acc_{A_2}|$$

There are some issues with AD that suggest that an alternative distance measure may be preferable. Consider, for example, a situation in which both A_1 and A_2 come out with an accuracy of 50%, i.e., $acc_{A_1} = acc_{A_2} = 0.5$. In this case, $AD(A_1, A_2) = 0$ and one would thus be tempted to conclude that A_1 and A_2 are similar. Yet, they may actually have drastically different behaviors, as A_1 could be correct on all of the instances that A_2 misclassifies, and vice-versa. This mismatch between AD and its intended interpretation is due, of course, to the fact that accuracy is a global measure of performance. Similar discrepancies would arise with any other global measure, such as area under the ROC curve.

Hence, we seek to find a distance function based on more local measures of performance, which take into account differences at the instance level rather than across whole datasets. While these measures have often been used in work on ensemble learning, where diversity is intended to be maximized, they have not, to the best of our knowledge, been used in clustering, where diversity is intended to be minimized, or conversely, similarity maximized. Table 5.1 summarizes the most popular diversity measures as reported in recent studies. We will show that all but one of these measures, although useful for diversity, fail to qualify as distance functions.

Table 5.1: Measures of Diversity

	Q	ρ	**DF**	κ	**DM**	**H**	**EC**	**COD**
Dietterich (2000)				X				
Kuncheva and Whitaker (2001, 2003)	X	X	X	X[a]	X			
Ali and Pazzani (1996) Kalousis et al (2004)							X	
Narasimhamurthy (2005)	X	X	X		X			
Brodley (1996) Peterson and Martinez (2005)								X
Gatnar (2005)	X	X	X	X	X	X		
Tang et al (2006)			X	X[a]	X			
Chung et al (2008)		X	X	X[a]	X			

[a]Non-pairwise version.

We dismiss the κ statistic, or measure of interrater agreement, at the onset. Indeed, while it has been used in a few instances as a pairwise measure, it is more often considered a non-pairwise similarity measure in machine learning. Furthermore, there is not a single definition of κ, even though the probably most reported one originated in [27], which was designed specifically as a non-pairwise measure. Some versions also lead to unexpected answers (e.g., smaller values for seemingly stronger agreement) while others produce indeterminate forms (i.e., $\frac{0}{0}$) when the learners have diametrically opposed behaviors (e.g., one is always correct and the other is always wrong). We likewise dismiss Pearson's correlation coefficient, ρ, since one can show that for any two learners, $|\rho| \leq |Q|$ [31, 43], and hence ρ is in some sense subsumed by Q. Finally, the Q statistic, originally defined in [87] as a means to capture the degree of association between two induced hypotheses with respect to the target hypothesis, cannot distinguish among different output distributions. For this reason, H was suggested in [31] as an alternative to Q. Hence, we also ignore Q in what follows.

We now turn to a brief description and discussion of the remaining measures. Since we do not typically have access to actual probabilities, we use frequency-estimates to compute diversity values. As above, let A_1 and A_2 be the two learning algorithms under study. Let h be the target classification hypothesis. Let h_1 and h_2 be the hypotheses induced by A_1,

respectively A_2, on some training set derived from h. We adopt the notation of Table 5.2, adapted from [42].

Table 5.2: Notation for Similarity Measures

Variable	**Description**
N^{11}	number of instances on which both h_1 and h_2 are correct $N^{11} = \|\{x : h_1(x) = h_2(x) = h(x)\}\|$
N^{10}	number of instances on which h_1 is correct, but h_2 is incorrect $N^{10} = \|\{x : h_1(x) = h(x) \wedge h_2(x) \neq h(x)\}\|$
N^{01}	number of instances on which h_2 is correct, but h_1 is incorrect $N^{01} = \|\{x : h_1(x) \neq h(x) \wedge h_2(x) = h(x)\}\|$
N^{00}	number of instances on which both h_1 and h_2 are incorrect $N^{00} = \|\{x : h_1(x) \neq h(x) \wedge h_2(x) \neq h(x)\}\|$
N_S^{00}	number of instances on which both h_1 and h_2 are incorrect, but they make the same prediction $N_S^{00} = \|\{x : h_1(x) = h_2(x) \wedge h_1(x) \neq h(x)\}\|$
N_D^{00}	number of instances on which both h_1 and h_2 are incorrect, and they make different predictions $N_D^{00} = \|\{x : h_1(x) \neq h(x) \wedge h_2(x) \neq h(x) \wedge h_1(x) \neq h_2(x)\}\|$
N	total number of instances $N = N^{11} + N^{10} + N^{01} + N^{00}$

Note that $N^{00} = N_S^{00} + N_D^{00}$. The measures of Table 5.1 are defined as follows.

- **Double Fault (DF).** DF, also known as compound diversity [32], is the probability that both h_1 and h_2 are incorrect.

$$DF = \frac{N^{00}}{N}$$

- **Disagreement Measure (DM).** DM, introduced in [68], is the probability that either h_1 or h_2 is correct but not both, i.e., h_1 and h_2 are complementary.

$$DM = \frac{N^{01} + N^{10}}{N}$$

- **Hamann's coefficient (H).** H, suggested in [31] as an alternative to Q, captures the degree of association between h_1 and h_2 with respect to h. H ranges over $[-1, +1]$.

$$H = \frac{(N^{11} + N^{00}) - (N^{10} + N^{01})}{N}$$

- **Error Correlation (EC).** To the best of our knowledge, EC has been proposed and used only twice as a measure of diversity, once in [1], and later in [38]. Unfortunately, while the second paper cites the first as its source, the definitions of EC in those papers are slightly different. In the former, EC is defined as $P(h_1 = h_2, h_1 \neq h)$. We refer to this version of EC as EC_a. In the latter, EC is defined as $P(h_1 = h_2 | h_1 \neq h \vee h_2 \neq h)$. We refer to this version of EC as EC_k. It is clear that unless $N^{11} = 0$, $EC_a(X, Y) \neq EC_k(X, Y)$.

$$EC_a = \frac{N_S^{00}}{N}$$

$$EC_k = \frac{N_S^{00}}{N^{01} + N^{10} + N^{00}}$$

- **Classifier Output Difference (COD).** COD, introduced in [58], is the probability that h_1 and h_2 make different predictions, i.e., $P(h_1 \neq h_2)$.[1]

$$COD = \frac{N^{10} + N^{01} + N_D^{00}}{N}$$

In practice, of course, the above measures are computed by running each algorithm on a number of datasets and averaging the results.

Now, recall that a distance measure, d, is a metric (or distance function) if and only if it satisfies the following four properties (here X and Y are algorithms):

1. Non-negativity: $d(X, Y) \geq 0$

[1]Note that while we use COD here, we must point out that earlier work on diversity in [19] had defined the *classification overlap* among a set of learning algorithms as the number of instances that are classified the same by multiple classifiers, which for two classifiers can easily be shown to be equivalent to $1 - COD$.

2. Identity of indiscernibles: $d(X,Y) = 0 \iff X = Y$[2]

3. Symmetry: $d(X,Y) = d(Y,X)$

4. Triangle inequality: $d(X,Y) \leq d(X,Z) + d(Z,Y)$

These properties, especially the second one, bring out a critical distinction between measuring distance as is typically done in ensemble learning and measuring distance as we intend to do in clustering algorithms. In ensemble learning, the focus is on diversity, where one tries to *maximize* the distance between algorithms, thus focusing on the high-end of the distance spectrum (i.e., away from 0). In clustering, the focus is on similarity, where one tries to *minimize* the distance between algorithms, thus focusing on the low-end of the distance spectrum (i.e., close to 0). As a result, the identity of indiscernibles, which has to do with close points, may easily be relaxed —and often is— in the context of diversity, but is critical in the context of clustering.

It is easy to show that DF does not satisfy the identity of indiscernibles, and thus is not a metric. The same is true of H as shown in Theorem 5.3.1.

Theorem 5.3.1. *H does not give rise to a distance function.*

Proof. We use $\frac{1-H}{2}$ rather than H because H ranges over [-1,1], with -1 meaning complete disagreement, 0 meaning an equal number of agreements and disagreements, and +1 meaning complete agreement. The transformed quantity ranges over [0,1] and behaves more like a

[2]Note that here $X = Y$ means that X and Y have indistinguishable behaviors, not necessarily that X and Y are the same algorithm, which is consistent with the idea of identify of *indiscernibles*.

distance. We show that $\frac{1-H}{2}$ does not satisfy the identity of indiscernibles in general.

$$\begin{aligned}
\frac{1-H(X,Y)}{2}=0 \quad &\iff \quad 1-H(X,Y)=0 \\
&\iff \quad H(X,Y)=1 \\
&\iff \quad \frac{(N^{11}+N^{00})-(N^{10}+N^{01})}{N}=1 \\
&\iff \quad (N^{11}+N^{00})-(N^{10}+N^{01})=N \\
&\iff \quad N^{11}+N^{00}-N^{10}-N^{01}=N^{11}+N^{00}+N^{10}+N^{01} \\
&\iff \quad 2(N^{10}+N^{01})=0 \\
&\iff \quad N^{10}+N^{01}=0 \\
&\iff \quad N^{10}=0 \text{ and } N^{01}=0 \\
&\iff \quad N=N^{11}+N^{00}
\end{aligned}$$

which, for binary classification is equivalent to $X=Y$ since $N^{00}=N_S^{00}$. However, this is not true in the general case as demonstrated in the following table, where both the predictions of X and Y, and the target value are shown.

	X	Y	Target
1	0	0	0
2	1	0	2
3	1	1	1
4	0	1	2
5	0	2	1

Then $\frac{1-H(X,Y)}{2}=\frac{1-\frac{(2+3)-(0+0)}{5}}{2}=\frac{1-1}{2}=0$. Yet X and Y act rather differently on each of the instances, i.e., they do not have the same behavior. □

It is easy to show that EC_k may give rise to an indeterminate form, for example when X and Y are 100% accurate, which makes it undesirable. On the other hand, EC_a does not

give rise to a distance function since it does not satisfy the identity of indiscernibles as shown in Theorem 5.3.2.

Theorem 5.3.2. *EC_a does not give rise to a distance function.*

Proof. We use $1 - EC_a$ rather than EC_a since EC_a is essentially a measure of similarity rather than distance. We show that $1 - EC_a$ does not satisfy the identity of indiscernibles. We need only find a counterexample. Consider a simple dataset consisting of 5 instances, each labeled as demonstrated in the following table, where both the predictions of X and Y, and the target value are shown.

	X	Y	Target
1	0	0	0
2	1	1	1
3	1	1	1
4	0	0	1
5	1	1	0

It is clear that X and Y act exactly identically on each of the instances, i.e., they have the same behavior ($X = Y$). Yet, $1 - EC_a(X, Y) = 1 - 2/5 = 3/5 \neq 0$. □

For binary classification, $COD(X, Y) = DM(X, Y)$, since in that case $N_D^{00} = 0$, i.e., it is impossible for both algorithms to be wrong and have different predicted values. In the general case, however, when $N_D^{00} \neq 0$, $DM(X, Y) < COD(X, Y)$ and one can easily show that DM does not satisfy the identity of indiscernibles.

This leaves us with only COD. We now show that COD is a distance function.[3]

Theorem 5.3.3. *COD is a distance function.*

We need to show that COD satisfies the four properties of metrics.

[3]Note that COD was originally claimed to be a metric in [58]; however, it was never proven.

1. $COD(X, Y) \geq 0$

   Follows directly from the definition of COD as the sum and ratio of only positive quantities.

2. $COD(X, Y) = 0 \iff X = Y$

$$
\begin{aligned}
COD(X,Y) = 0 \quad &\iff \quad \frac{N^{10} + N^{01} + N_D^{00}}{N} = 0 \\
&\iff \quad N^{10} + N^{01} + N_D^{00} = 0 \\
&\iff \quad N - (N^{11} + N_S^{00}) = 0 \\
&\iff \quad N = N^{11} + N_S^{00} \\
&\iff \quad X = Y
\end{aligned}
$$

3. $COD(X, Y) = COD(Y, X)$

   Follows directly from the definition of COD.

4. $COD(X, Y) \leq COD(X, Z) + COD(Z, Y)$

   Given any instance, there are 5 possible outcomes for the predictions of X, Y and Z, namely

   (a) X, Y and Z all predict the same target value

   (b) X predicts the same target value as Y, which is different from the value predicted by Z

   (c) X predicts the same target value as Z, which is different from the value predicted by Y

   (d) Y predicts the same target value as Z, which is different from the value predicted by X

   (e) X, Y and Z all predict different target values

Let $n_a, \ldots, n_e$ denote the number of instances corresponding to each of the above cases ($n_a + \ldots + n_e = N$). It follows that $COD(X,Y) = n_c + n_d + n_e$, $COD(X,Z) = n_b + n_d + n_e$ and $COD(Z,Y) = n_b + n_c + n_e$. Hence,

$$\begin{aligned} COD(X,Z) + COD(Z,Y) &= 2n_b + n_c + n_d + 2n_e \\ &= COD(X,Y) + 2n_b + n_e \\ &\geq COD(X,Y) \end{aligned}$$ □

We thus propose to use COD as an effective distance function for algorithm behavior. In addition to being a metric, COD has the following intuitively appealing characteristics.

- It generalizes DM.
- It is independent of the target, inasmuch as its computation relies only on the predictions of the two algorithms under consideration.[4]
- It is rather intuitive as a notion of distance, with some analogy to the classical Hamming distance.
- Whereas EC, and other measures, are restricted to errors only, COD captures a more complete picture of behavior differences.

Finally, Theorem 5.3.4 establishes that COD is strictly stronger than AD in the sense that if two algorithms are similar according to COD, then they also have similar predictive performance. The converse is clearly not true as shown by the counterexample discussed earlier in this section to highlight the problem with AD.

Theorem 5.3.4. $COD(X,Y) \leq \epsilon \Rightarrow AD(X,Y) \leq \epsilon$

[4]There is some dependency on the target, of course, in the sense that both predictive models are learned from data that include the target values. However, unlike measures like EC, this dependency is indirect.

Using our notation, we can write $AD = \frac{|N^{10} - N^{01}|}{N}$. Now,

$$\begin{aligned}
COD(X,Y) \leq \epsilon \quad &\Longleftrightarrow \quad \frac{N^{10} + N^{01} + N_D^{00}}{N} \leq \epsilon \\
&\Rightarrow \quad \frac{N^{10}}{N} \leq \epsilon \text{ and } \frac{N^{01}}{N} \leq \epsilon \\
&\Rightarrow \quad -\epsilon \leq \frac{N^{10} - N^{01}}{N} \leq \epsilon \\
&\Rightarrow \quad \frac{|N^{10} - N^{01}|}{N} \leq \epsilon \\
&\Rightarrow \quad AD(X,Y) \leq \epsilon
\end{aligned}$$

□

5.4 Clustering Learning Algorithms

Equipped with a distance metric, we can now apply clustering techniques over a set of learning algorithms. Since the data being clustered is the result of learning at the base level, our clustering is a form of unsupervised metalearning. Given a resulting clustering, we will be particularly interested in algorithms that are grouped together into tight clusters, as these will highlight algorithms whose behaviors are very close to each other.

We consider 21 learning algorithms from Weka [83], selected to be representative of various model classes. In all cases, the algorithms are considered with their default settings. The algorithms and their grouping into model classes based on Weka's internal taxonomy are shown in Table 5.3.

For our distance-based clustering method, we use hierarchical agglomerative clustering (HAC), as it produces a complete sequence of nested clusterings, as follows. HAC starts by assigning each learning algorithm to its own cluster. Then, the two closest clusters are merged into a single new cluster. This pairwise merging process is repeated until a single cluster containing all of the learning algorithms is obtained.

Although we have a distance defined over algorithms, HAC also needs a distance over clusters. Several distance measures may be considered. The most popular ones are complete linkage, which uses the maximum distance between all pairs of objects across clusters, single

Table 5.3: Selected Learning Algorithms

Weka's Class	**Algorithm**
Bayes (2)	BayesNet
	NaiveBayes (NB)
Functions (4)	Logistic Regression (Logistic)
	RBFNetwork (RBFN)
	MultilayerPerceptron (MLP)
	SMO
Lazy (3)	1 Nearest Neighbor (IB1)
	3 Nearest Neighbor (IB3)
	LWL
Trees (6)	SimpleCart
	LADTree
	FTree (FT)
	J48
	NBTree
	RandomForest (RandForest)
Rules (6)	DecisionTable (DecTable)
	JRip
	NNge
	PART
	Ridor
	ZeroR

linkage, which takes the minimum distance, and average linkage, which computes the average of all inter-cluster distances. We choose complete linkage here as it tends to create more compact, clique-like clusters.[5]

To compute the distance between pairs of algorithms, we use 129 datasets from several popular sources:

- 72 data sets from the UCI Machine Learning Repository [4]
- 45 data sets from the Gene Expression Machine Learning Repository [75]
- 12 data sets from ASU's Multi-class Protein Fold Recognition data[6]

[5]Complete linkage is also known to be more sensitive to outliers. But there are no outliers in our set of 21 learning algorithms.

[6]See http://www.public.asu.edu/∼sji03/resources/data/protein-data.zip

For each dataset, the predictions of all instances are obtained by 10-fold cross-validation. The COD value for every pair of algorithms is then obtained by averaging the COD values obtained on all of the datasets.

For our implementation of HAC, we use the *agnes* function from the *cluster* package of R [61]. Figure 5.1 shows the dendrogram resulting from clustering our 21 learning algorithms.

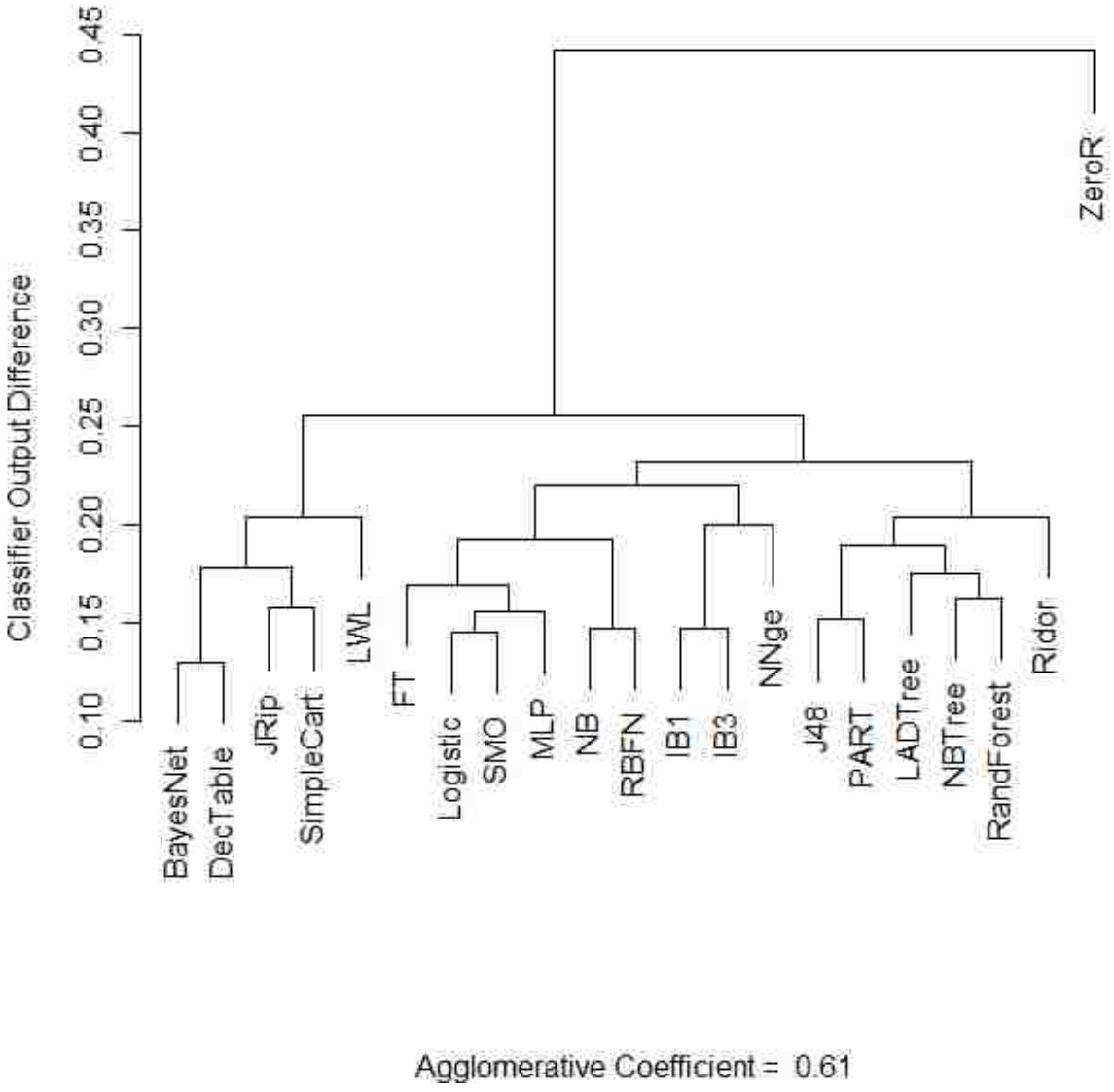


Figure 5.1: Clustering Based on COD

The agglomerative coefficient value (0.61) suggests that COD is able to extract a significant amount of structure in the data. While a detailed analysis is beyond the scope of this paper, we do make a few observations about the clustering, to illustrate how it may be used to confirm, complement and/or extend our knowledge about learning algorithms,

As one might expect, much of Weka's taxonomy (see Table 6.1) is found in the natural clustering. For example, three of the four function-based learning algorithms (i.e., Logistic, MLP and SMO) land in the same cluster relatively low in the dendrogram, with the fourth one, RBFN, joining a little higher. Similarly, most decision tree learning algorithms tend to cluster together first, as do both nearest-neighbor algorithms (i.e., IB1 and IB3). On the other hand, other clusters do not match the taxonomy so well. For example,

- The rule-based learning algorithm PART clusters first with the decision tree learner J48,
- The decision tree learning algorithm FT clusters first with the group of three function-based learners,
- The rule-based learning algorithm ZeroR is clustered last high in the dendrogram, and
- The function-based learner RBFN clusters first with the Bayes learner NB.

Despite being in different classes, PART and J48 do share significant similarity since PART also uses a divide-and-conquer approach in which, at each iteration, it builds a partial tree and extracts a rule from it. Similarly, FT bears resemblance to Logistic as it induces classification trees with logistic regression functions at the inner nodes and leaves. Also, ZeroR, despite being labeled as rule-based, simply extracts the majority class and uses it as its prediction on all new instances. Such simplistic behavior is unlikely to match any of the more sophisticated learning algorithms considered. The last grouping seems less obvious at first sight. Upon further examination of their inner workings and specific Weka default

implementations, one may better appreciate the similarities and differences between these two learning algorithms. This is the subject of a separate analysis [45].[7]

To make the value of our COD metric for unsupervised metalearning clearer, we contrast the COD-based clustering with the clustering obtained by AD, the difference in predictive accuracy. The setting is the same as above. The resulting clustering is in Figure 5.2.

Again, the agglomerative coefficient value (0.79) suggests that AD is able to extract a significant amount of structure in the data. Note, though, that the dendrogram tends to be rather flat (except for ZeroR, which, as expected, clusters last and higher up in the dendrogram). This confirms the findings of others that most learning algorithms, including simple ones, perform rather similarly in terms of accuracy across a wide variety of datasets (e.g., see [37]).

Beyond this general observation, it is clear that the picture provided by AD is different from that provided by COD, and that variations in algorithm behavior are not captured in the aggregate by AD. For example, the following pairs of algorithms, RandForest and MLP, Ridor and SimpleCart, and JRip and Ridor, are close together in the AD clustering, while the COD clustering makes clear that their behavior is not that similar. Clearly, according to Theorem 5.3.4, algorithms close to each other according to COD are also close together according to AD (e.g., see NB and RBFN).

5.5 Conclusion

In this paper, we have argued the value of unsupervised metalearning and discussed the attendant necessity of suitable similarity, or distance, functions. We have capitalized on the well-known notion of diversity among learners used in ensemble learning, and proposed to use them as distance measures to cluster learning algorithms. We have revisited the

[7]Note that such an analysis is prompted by the results of our clustering. In addition to NB being probability-based and RBFN being function-based, NB is a generative model while RBFN is a discriminative one. Yet, overall, RBFN behaves more like NB than any other algorithms. It is rather unlikely that traditional perceptions about NB and RBFN would have led to such a discovery and subsequent analysis.

most popular measures of diversity and showed that only one of them, Classifier Output Difference (COD), qualifies as a metric. We have then used COD to produce a clustering of 21 learning algorithms, based on results from 129 datasets. We have shown how this clustering differs from a clustering based on accuracy, and how it can be used to highlight interesting, sometimes unexpected, similarities among algorithms.

Unsupervised metalearning, as described here, contributes to increasing our understanding of how learning algorithms behave. By focusing its attention on similarity, rather than diversity, it might also prove useful in reducing the complexity of the corresponding supervised metalearning task of algorithm selection. Indeed, rather than attempting to build a system that maps a dataset to one of N algorithms, where N is relatively large, the metalearner could induce a mapping from a dataset to one of C clusters of similar algorithms, where C is much smaller than N. We intend to pursue work in this area.

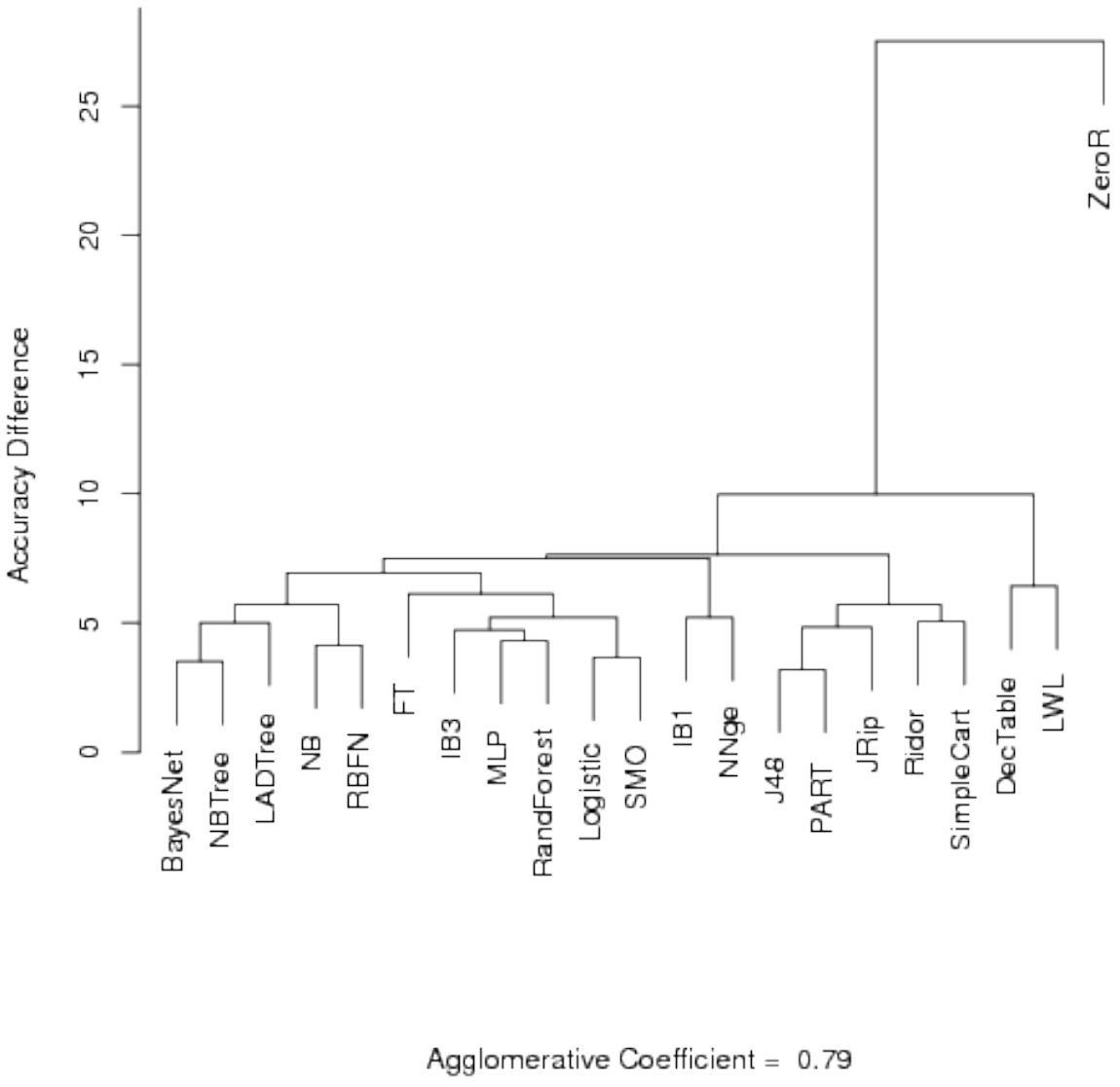


Figure 5.2: Clustering Based on AD

Chapter 6

A Comparison of Naïve Bayes and Radial Basis Function Networks in Weka

Abstract

We compare naïve Bayes and radial basis function networks. We show, using both analytical tools and empirical results, that for Gaussian kernels, there is a significant amount of similarity between them across a broad range of datasets for small numbers of kernels. We further show that larger number of kernels are typically not useful and thus the observed similarity, when applicable, is of practical import. In particular, since radial basis function network learning is significantly more computationally expensive than naïve Bayes learning, we use metalearning to build a selection model capable of accurately discriminating between the two algorithms, so that extra computation is only incurred when it is guaranteed to produce significant improvement in predictive accuracy.

6.1 Introduction

In recent work on unsupervised metalearning, we showed how Classifier Output Difference (COD) [58], one of the diversity measures typically used in ensemble learning, has the properties of a metric, and can thus be used as a distance function to cluster learning algorithms [47]. COD estimates the probability that two classification learning algorithms make different predictions. Unlike global measures, such as accuracy, which provide only an idea of average performance over all instances, COD captures local variations among instances. Furthermore, COD is strictly stronger than accuracy, in the sense that if two algorithms are close based on COD, they must also have similar accuracy.

We clustered 21 well-established learning algorithms from Weka [83] (with their default settings) using COD distance information averaged over 129 datasets from various sources including the UCI Machine Learning Repository [4] and the Gene Expression Machine Learning Repository [75]. The learning algorithms were selected to represent different model classes, including tree-based, rule-based, instance-based, probability-based and function-based approaches. The algorithms and their grouping into model classes based on Weka's internal taxonomy are listed in Table 6.1. The result of complete-linkage, hierarchical agglomerative clustering is shown in Figure 6.1.

In that study, we were, in part, interested in seeing whether our clustering would induce clusters that were consistent with the de facto taxonomy defined by Weka's model classes. We expected that to be the case, with algorithms from the same model classes finding their way to the same cluster and algorithms from different model classes being grouped separately. While this is true for many algorithms (e.g., Logistic, MLP and SMO; J48, LADTree, NBTree and RandForest), there are some notable exceptions, one of which, the focus of this paper, is highlighted in Figure 6.1.

The grouping of NB and RBFN does not appear as an obvious one at first sight. In addition to NB being probability-based and RBFN being function-based, NB is a generative

Table 6.1: Selected Learning Algorithms

Weka's Class	**Algorithm**
Bayes (2)	BayesNet
	NaiveBayes (NB)
Functions (4)	Logistic Regression (Logistic)
	RBFNetwork (RBFN)
	MultilayerPerceptron (MLP)
	SMO
Lazy (3)	1 Nearest Neighbor (IB1)
	3 Nearest Neighbor (IB3)
	LWL
Trees (6)	SimpleCart
	LADTree
	FTree (FT)
	J48
	NBTree
	RandomForest (RandForest)
Rules (6)	DecisionTable (DecTable)
	JRip
	NNge
	PART
	Ridor
	ZeroR

model while RBFN is a discriminative one.[1] Yet, overall, RBFN behaves more like NB than any other algorithms. Before proceeding any further, we wish to make a couple of things clear.

1. While based on COD values averaged across all of our 129 datasets, NB and RBFN cluster together, closer examination revealed that that relationship actually holds only for datasets whose attributes are continuous. It so happens that a majority of our datasets (85) have that property, which biases the average. We will restrict our comparison of NB and RBFN to the case of continuous-only attributes.

2. It is well-known that the result of hierarchical clustering may be data dependent. In other words, it is possible that we obtain a different hierarchical structure when

[1]Interestingly, the root of the classification method subtree in the newly developed DMOP ontology [36] splits on generative vs. discriminative methods, which would send NB down one branch and RBFN down the other.

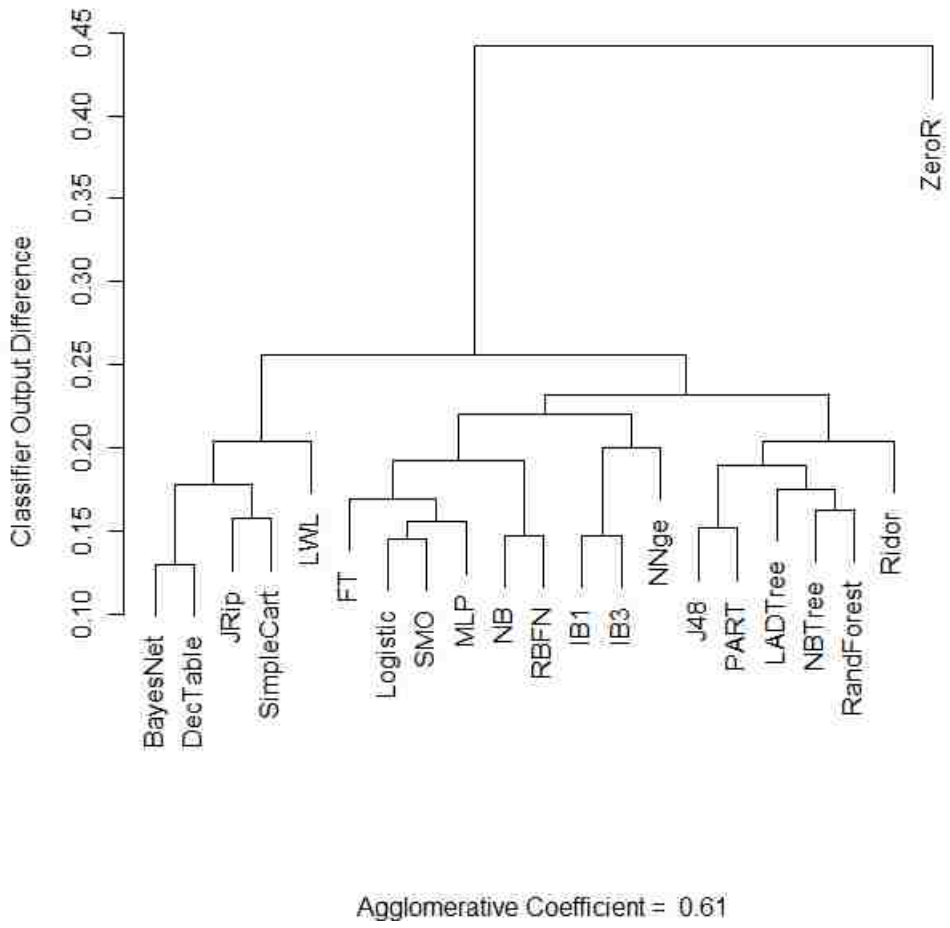


Figure 6.1: *COD*-based Clustering of Learning Algorithms

different datasets are used to construct it. We checked the robustness of our finding with respect to NB and RBFN by sampling 60 datasets (out of 85) at random, and computing the corresponding COD matrix. We then produced a ranking of the 5 algorithms most similar to NB, in decreasing value of COD. Since we use complete linkage, this ranking also represents the clustering order for NB. We repeated the sampling 10 times with different random seeds. In every case, NB did cluster first with RBFN, followed by MLP.

While the observed high degree of similarity between NB and RBFN may not be as unexpected upon further examination of Weka's implementations and default parameters, a detailed comparison is instructive, especially as it is rather unlikely that traditional perceptions about NB and RBFN would have led to such a discovery and subsequent analysis. It is also instructive in terms of the risks faced by practitioners who, lacking the necessary expertise, tend to confine themselves to the use of default implementations.

The paper is organized as follows. In section 6.2, we briefly highlight previous work relevant to our analysis. Section 6.3 reviews the way NB and RBFN learn in general, and how they are implemented in Weka. In section 6.4, we present an analysis of the similarity between NB and RBFN. We draw on both analytical arguments, where applicable, and empirical results, when analytical forms are not readily available. We show that under Weka's assumptions, and in the aggregate, RBFN and NB are indeed rather similar in terms of predictive accuracy while widely different in terms of training time. Hence, in section 6.5, we look at specific situations in which the similarity between NB and RBFN clearly does not hold, and use metalearning to build a selection metamodel capable of accuractely discriminating when RBFN should be used for prediction and when the same result may be obtained by NB at a fraction of the computational cost. Section 6.6 discusses the scope of applicability of the observed similarity, and its consequences on the practice of machine learning. Finally, section 6.7 concludes the paper.

6.2 Related Work

There are, of course, a large number of empirical studies comparing the performance of learning algorithms. It has long been a tacit requirement for publication in our community that anyone wishing to introduce a novel algorithm should compare it against at least a few others on some reasonable set of learning tasks. Less common are targeted comparisons involving a couple of algorithms and leveraging both empirical and analytical results, as

we present here.[2] Notable exceptions focus on broad classes of algorithms, such as the relationships between neural networks and statistical models (e.g., see [65]), and between generative (or informative) and discriminative models (e.g., see [55, 64]). Another interesting exception is the use of learning curves as an analytical tool in the comparison of logistic regression and decision tree learning [57].

Most closely related to the analytical part of our study is the recent work on naïve Bayes and logistic regression. Elkan [25] shows that, for discrete inputs, naïve Bayes is a generalization of logistic regression. Mitchell [52] shows that, for continuous inputs, the form of $P(Y|X)$ entailed by the assumptions of Gaussian naïve Bayes with binary classification tasks is exactly the form used by logistic regression. Ng and Jordan [55] explain that naïve Bayes and logistic regression form what they call a generative/discriminative pair. They go on to demonstrate that the generative approach has higher asymptotic error, but that two regimes of performance seem to be present. The generative approach reaches its asymptote faster than the discriminative approach, suggesting that the generative approach may be preferable for small number of examples and the discriminative approach for larger numbers. Likewise, we show that, given some restrictive assumptions on the learning tasks and parameters of RBFN, the decision boundaries used by NB and RBFN are of the same form. For more general, and realistic, settings an analytical approach proves challenging, so that we move to an empirical study.

6.3 Preliminaries

Prior to our detailed analysis, we give a brief overview of naïve Bayes and radial basis function network learning.

[2]This may be due to the fact that we are limited by our imagination or what we can think about at a particular time, while with the type of clustering presented here, new avenues open up automatically.

6.3.1 Naïve Bayes

The naïve Bayes learning algorithm (NB) is a highly practical and efficient probability-based learning algorithm built upon the assumption that the task's features are conditionally independent given the target value. That is, given the set of features $X = \{X_1, X_2, \ldots, X_n\}$ and discrete set of target, or class, values C:

$$P(X_1, X_2, \ldots, X_n|C) = \prod_i P(X_i|C)$$

For every new query instance $x = \langle x_1, x_2, \ldots, x_n \rangle$, NB returns the class value with maximum posterior probability:

$$\begin{aligned} c_q^{NB}(x) &= argmax_{c_j} P(C = c_j | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &= argmax_{c_j} \frac{P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n | C = c_j) P(C = c_j)}{P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)} \\ &= argmax_{c_j} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n | C = c_j) P(C = c_j) \\ &= argmax_{c_j} P(C = c_j) \prod_i P(X_i = x_i | C = c_j) \end{aligned}$$

The second line is the result of applying Bayes' theorem. The denominator is dropped in the third line since it does not depend on c_j. Finally, the last line follows from the conditional independence assumption.

Let D be a set of training examples for NB. Two different cases must be considered to compute $P(X_i = x_i | C = c_j)$.

1. When X_i is discrete, $P(X_i = x_i | C = c_j)$ is typically estimated with a smooth approximation:

$$\frac{\sharp D\{X_i = x_i \cap C = c_j\} + l}{\sharp D\{C = c_j\} + l \mid X_i \mid}$$

   where $\sharp D\{cond\}$ is the number of examples in D that satisfy *cond*, and l is the strength of the smoothing.

2. When X_i is continuous, $P(X_i = x_i|C = c_j)$ is assumed to follow some probability distribution.

As we restrict our attention to learning tasks with continuous attributes, the second case is the only one of interest to us here. The most common approach, and that used in Weka, is to consider that $P(X_i = x_i|C = c_j)$ is a Gaussian, with mean μ_i and standard deviation σ_i:

$$P^{NB}(X_i = x_i|C = c_j) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}}$$

Note that σ_i depends on X_i only, not on the target c_j. Since the product of Gaussian functions is also Gaussian, we can write

$$\begin{aligned}\prod_i P(X_i = x_i|C = c_j) &= \prod_i P^{NB}(X_i = x_i|C = c_j) \\ &= \prod_i G_j(\mu_i, \sigma_i)(x) \\ &= G_j^{NB}(x)\end{aligned}$$

and it follows that

$$c_q^{NB}(x) = argmax_{c_j} P(C = c_j) G_j^{NB}(x)$$

6.3.2 Radial Basis Function Network

The Radial Basis Function Network learning algorithm (RBFN) is typically described in terms of a three layer feed-forward network architecture. However, RBFN differs from classical multi-layer perceptrons in three significant ways: 1) there is only one set of trainable weights, from the hidden layer to the output layer; 2) the nodes' activation functions are non-standard (i.e., neither sign nor sigmoid); and 3) learning is effected by a combination of supervised and unsupervised techniques.[3]

[3]Note that it is possible to train RBFN in a fully supervised manner. Because it is much more computationally efficient, the hybrid learning procedure is generally preferred.

In RBFN, the nodes of the hidden layer are local attractors that encode a set of well positioned centroids together with a "sphere" of influence. Each attractor is such that its influence over points in the input space decreases as the distance from its centroid increases. Each hidden node, h, thus encodes a function $K_h(d(\mu_h, x))$, sometimes called a kernel function, where μ_h is a centroid, $d(\mu_h, x)$ is the distance from μ_h to x, and K_h is such that it reaches its maximum at μ_h and decreases smoothly as $d(\mu_h, x)$ increases. Following the transformation of the input space by the hidden layer, linear combinations of the K_h's are learned to produce the final network's outputs. Let H be the number of hidden nodes or kernel functions, and m be the number of output nodes. For regression, the value computed by output node j $(1 \leq j \leq m)$ is given by:

$$f_j(x) = w_{0,j} + \sum_{h=1}^{H} w_{h,j} K_u(d(\mu_h, x))$$

where $w_{0,j}$ is a bias weight. For classification, on the other hand, RBFN is typically set up in such a way that, if there are m target classes, the network has $m-1$ output nodes, each computing $f_j(x)$. The predicted class for query instance x is then given by $c_q^{RBFN}(x) = argmax_{c_j} P^{RBFN}(C = c_j|x)$, with the probability $P(C = c_j|x)$ of each target class defined by:

$$P^{RBFN}(C = c_j|x) = \begin{cases} \dfrac{e^{f_j(x)}}{1 + \sum_{i=1}^{m-1} e^{f_i(x)}} & \text{if } 1 \leq j \leq m-1 \\ \\ \dfrac{1}{1 + \sum_{i=1}^{m-1} e^{f_i(x)}} & \text{if } j = m \end{cases}$$

In the case of regression, the output weights $(w_{h,j})$ may be obtained analytically, but for classification an iterative method is usually necessary. The exact behavior of RBFN depends on two major design decisions: 1) the choice of the functional form of the K_h's, and 2) the value of H, i.e., the number of radial basis functions.

The type of function used for the K_h's depends on the kind of tasks RBFN is to solve. For example, in time series modeling, thin plate spline functions are often used, while for

classification and pattern recognition, Gaussian kernel functions are generally preferred. As far as the number of basis functions is concerned, there are no known theoretical results for selecting the optimal number of hidden nodes for a given task in RBFN. Hence, the value of H is typically chosen by experimentation, augmented by any prior knowledge about the task and its structure. Related to the number of radial basis functions is their locations, i.e, the positions of the μ_h's, in the input space. One solution consists in having each training instance act as a centroid. While this works well in the context of function approximation, it leads to overfitting and excessive computational time in classification. What one wishes to achieve is good coverage of the input space with relatively few basis functions. Unsupervised learning, or clustering, provides a natural solution. Herein lies the hybrid nature of RBFN learning: the training data is first clustered to obtain the μ_h's; appropriate K_h's are then chosen; finally the $w_{h,j}$'s are computed via supervised learning.

Weka's RBFN is a standard implementation based on the use of "a separate mixture model to represent each of the [class-]conditional densities," as discussed in [13]. It works as follows.

1. Construct mk clusters by applying k-means clustering to each class independently, and fit a Gaussian to each cluster.

2. For each cluster, create a hidden node and set its radial basis function to the Gaussian weighted by the corresponding class prior and normalized, i.e.,

$$K_h(d(\mu_h, x)) = \frac{G_h^{RBFN}(x)P(c_{cl(h)})}{\sum_{i=1}^{H} G_i^{RBFN}(x)P(c_{cl(i)})}$$

   where G_h is the Gaussian for node h, $cl(h)$ is the class to which cluster h belongs, and $P(c_{cl(h)})$ is the prior class probability.

3. Run logistic regression on the outputs of the hidden nodes to obtain the weights.

We note that, by default, Weka's RBFN sets $k = 2$. Also, we call the reader's attention to the fact that, while valid and natural, Weka's choice to use class-dependent basis functions

rather than to share basis functions across classes has significant implications. We return to this point later.

6.4 Analysis of Algorithms

In this section, we proceed to compare RBFN and NB, as implemented in Weka. We rely on both analytical and empirical tools. We begin with the simplest of cases, where $k = 1$.

6.4.1 RBFN with $k = 1$

Recall that $c_q^{NB}(x) = argmax_{c_j} P(C = c_j) G_j^{NB}(x)$. Suppose now that we set $k = 1$ in RBFN, i.e., we model each class with a single cluster. To fit a Gaussian to each cluster in this case, Weka uses a diagonal covariance matrix, i.e., it assumes independence of the inputs given the class. It follows that the multivariate Gaussian for the cluster is simply the product of the univariate Gaussians for each input.[4] Hence,

$$\forall 1 \leq j \leq m \quad G_j^{RBFN} = G_j^{NB} = G_j$$

and the radial basis function for each hidden node is given by:

$$K_h(d(\mu_h, x)) = \frac{G_h(x)P(c_h)}{\sum_{i=1}^{m} G_i(x)P(c_i)}$$

Let us further assume that $m = 2$, i.e., there are only two target classes, c_1 and c_2. The corresponding RBFN thus has n input nodes, 2 hidden nodes and a single output node.

[4]Note that in Weka, the inputs to RBFN are actually first standardized. If $x = (x_1, \ldots, x_n)$, the inputs to RBFN are

$$x_i^s = \frac{x_i - \mu_i}{\sigma_i}.$$

The distribution of the x_i^s is Gaussian with mean 0 and standard deviation 1, so that fitting a Gaussian to the x_i^s is identical to fitting a Gaussian to the x_i. Indeed, for each x_i, the Gaussian fit is of the form

$$e^{-\frac{1}{2}\frac{(x_i-\mu_i)^2}{\sigma_i^2}} = e^{-\frac{1}{2}(x_i^s)^2} = e^{-\frac{1}{2}\frac{(x_i^s-0)^2}{1^2}}$$

which is the Gaussian fit for x_i^s. It follows that standardization of the inputs has no effect on the Gaussian fits.

The outputs of the hidden nodes are given by:

$$K_1(d(\mu_1, x)) = \frac{G_1(x)P(c_1)}{G_1(x)P(c_1) + G_2(x)P(c_2)}$$

$$K_2(d(\mu_2, x)) = \frac{G_2(x)P(c_2)}{G_1(x)P(c_1) + G_2(x)P(c_2)}$$

which we abbreviate to k_1 and k_2 respectively, for simplicity. Note that:

$$0 \leq k_1, k_2 \leq 1$$

$$k_1 + k_2 = 1$$

These values are fed into Weka's logistic regression learner to obtain the weights. However, prior to running, Weka's logistic regression standardizes its inputs so that the actual inputs to logistic regression are:

$$k_1^s = \frac{k_1 - \mu_{k_1}}{\sigma_{k_1}} \quad \text{and} \quad k_2^s = \frac{k_2 - \mu_{k_2}}{\sigma_{k_2}}$$

Since $k_1 + k_2 = 1$, it follows that $\sigma_{k_2} = \sigma_{k_1} = \sigma$ and $\mu_{k_2} = 1 - \mu_{k_1}$, so that

$$k_1^s = \frac{k_1 - \mu_{k_1}}{\sigma}$$

and

$$k_2^s = \frac{k_2 - \mu_{k_2}}{\sigma} = \frac{k_2 - (1 - \mu_{k_1})}{\sigma} = -\frac{(1 - k_2) - \mu_{k_1}}{\sigma} = -\frac{k_1 - \mu_{k_1}}{\sigma} = -k_1^s$$

Thus, the set of points presented to logistic regression lie on the line $k_1^s + k_2^s = 0$. Each one of these points is labeled as either from class c_1 or class c_2. Logistic regression then finds a decision boundary that best separates the two classes. Such a decision boundary is orthogonal to the line on which the points lie, so that its equation is of the form $k_1^s - k_2^s = \alpha$, for some α. In practice, logistic regression returns three values: the coefficient ω_1 for k_1^s, the

coefficient ω_2 for k_2^s, and the intercept ω_0. But here, $\omega_1 = -\omega_2 = \omega$. It follows that:

$$P^{RBFN}(c_1|x) = \frac{e^{\omega(k_1^s - k_2^s)+\omega_0}}{1 + e^{\omega(k_1^s - k_2^s)+\omega_0}}$$

$$P^{RBFN}(c_2|x) = \frac{1}{1 + e^{\omega(k_1^s - k_2^s)+\omega_0}}$$

Consequently, the discriminant function for RBFN, is:

$$\lambda^{RBFN}(x) = \log \frac{P^{RBFN}(c_1|x)}{P^{RBFN}(c_2|x)} = \omega(k_1^s - k_2^s) + \omega_0$$

such that RBFN predicts c_1 when $\lambda^{RBFN}(x) > 0$, and c_2 otherwise. By substituting in the values of k_1^s and k_2^s, we rewrite $\lambda^{RBFN}(x)$ in terms of k_1 and k_2, yielding:

$$\lambda^{RBFN}(x) = \frac{\omega}{\sigma}(k_1 - k_2) + \frac{\omega}{\sigma}(1 - 2\mu_{k_1}) + \omega_0$$

which, in terms of discriminative ability, is equivalent to:

$$\lambda^{RBFN}(x) = k_1 - k_2 + \alpha$$

where

$$\alpha = \frac{\omega_0 \sigma}{\omega} + (1 - 2\mu_{k_1})$$

On the other hand, NB predicts c_1 when $c_q^{NB}(x)$ is c_1, and c_2 otherwise. Since $c_q^{NB}(x)$ is c_1 when $G_1(x)P(c_1) > G_2(x)P(c_2)$, it follows immediately that NB predicts c_1 when $k_1 > k_2$, and c_2 otherwise. Hence, the discriminant function for NB is simply:

$$\lambda^{NB}(x) = k_1 - k_2$$

The similarity between λ^{RBFN} and λ^{NB} is obvious, with the former being a small generalization of the latter. NB's resulting decision boundary, together with a possible RBFN's

decision boundary (when $\alpha = 0.45$) are depicted in Figure 6.2. The thick segments represent the projections of the input space onto the normalized (i.e., $k_1 + k_2 = 1$), respectively standardized (i.e., $k_1^s + k_2^s = 0$), Gaussian kernels. The predictions of NB and RBFN are identical for all points except those that lie between the two parallel decision boundaries. It is possible that, even when the two boundaries do not coincide exactly (i.e., $\alpha \neq 0$ for RBFN), this set may be empty, and the behaviors of NB and RBFN would still be identical. In general, we expect some small variations in behavior between the two algorithms.

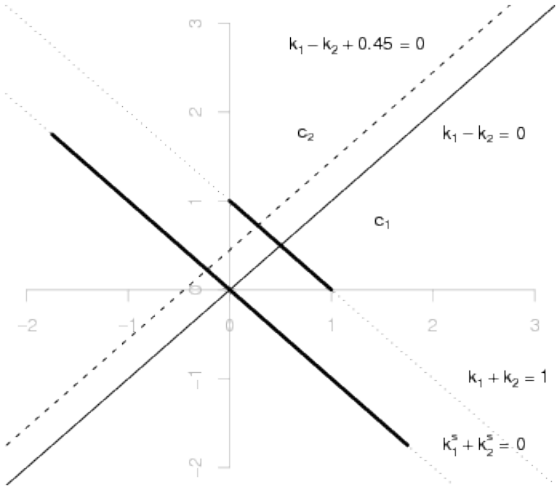


Figure 6.2: Decision Boundaries for NB and RBFN when $k = 1$ and $m = 2$. Two thick lines (long and short ones) indicate the input space after and before standardization, respectively. The thick-dotted line indicates the instance of decision line of RBFN and the solid line below it indicates the decision line of NB. It shows that the decision boundary of NB and RBFN is parallel and when $\alpha = 0$, they are identical.

As an illustration, Figure 6.3 shows the values of COD with respect to α for all 56 binary classification tasks in our selection of datasets. As expected, most points have small COD values and also congregate around small values of α. The two most significant outliers correspond to the UCI parkinsons dataset (COD=0.46) and the UCI Wisconsin prognostic breast cancer dataset (COD=0.31). A close look at these datasets reveals that in both cases they contain several attributes whose mean values are close to 0 (for both target classes). Since the multivariate Gaussians defined for each class are products of the univariate Gaussians defined over each attribute, attributes with mean around 0 tend to bring the products to 0 for both classes, so that after normalization $k_1 \approx k_2 \approx \frac{1}{2}$ and $\mu_{k_1} \approx \frac{1}{2}$. Because $k_1 \approx k_2$, $\lambda^{NB}(x)$ will be small (≈ 0) and on either side of 0, giving rise to a somewhat random classifier. On the other hand, since $\mu_{k_1} \approx \frac{1}{2}$, then $\alpha \approx \frac{\omega_0 \sigma}{\omega}$, and thus $\lambda^{RBFN}(x) \approx -\frac{\omega_0 \sigma}{\omega}$. Hence, RBFN behaves like a majority learner whose predicted class depends on the sign of ω_0 σ and ω are positive). It follows that in such cases, the behavior of NB and RBFN may become significantly different.

While for $m = 2$, the inputs to logistic regression in the RBFN learning scenario lie on the line defined by $k_1^s + k_2^s = 0$, by extension for larger values of m, the inputs to logistic regression lie on the hyperplane defined by $\sum_{i=1}^{m} k_i^s = 0$. However, the foregoing analysis of discriminant functions and geometry of decision boundaries does not generalize easily in this higher-dimensional space. Hence, we resort to an empirical analysis of the value of COD as m increases. Our selection of 85 datasets accounts for values of m between 2 and 28. However, for most of these, except for $m = 2$, there are 12 or less datasets with that value of m. Hence, we extend our selection of datasets so that each value of m ends up with 85 datasets. For each value of m, we generate a number of complementary of datasetoids [72] as follows.

1. Let d_m be the number of datasets with m target values
2. Select $85 - d_m$ datasets at random
3. For each selected dataset, create a datasetoid:

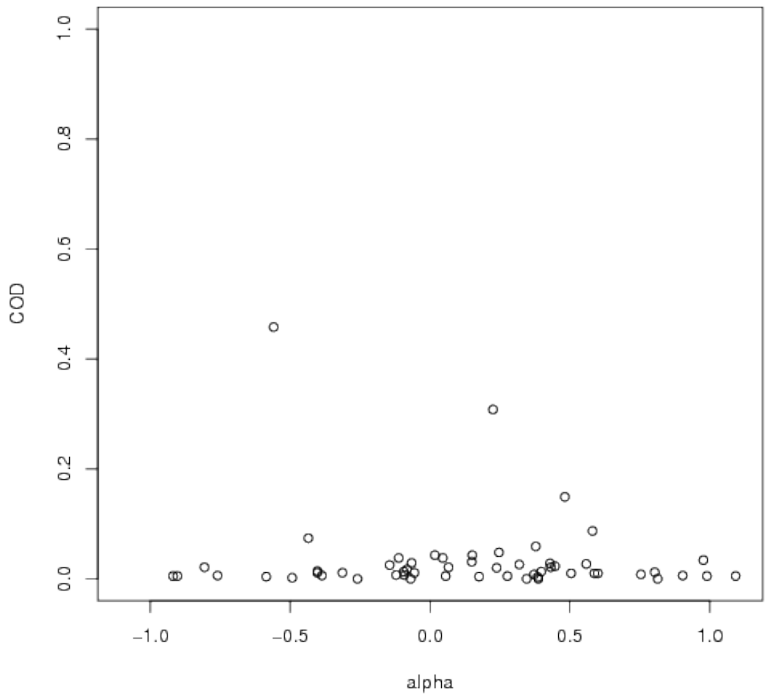


Figure 6.3: COD vs α for 56 Binary Classification Datasets ($k = 1$). It shows that the difference in behavior between RBFN and NB is small over most of datasets.

(a) Remove the target attribute

(b) Select an attribute at random

(c) Discretize the selected attribute into m bins and set it as target

4. Run each datasetoid against NB and RBFN

5. Compute average COD across all 85 datasets/datasetoids

Figure 6.4 shows the resulting average value of COD for several values of m between 2 and 28. The graph shows a generally increasing trend, such that the difference in behavior between RBFN and NB becomes larger as the number of target values increases. The value for $m = 2$ is very small, as expected, with a significant qualitative jump when $m = 3$ and beyond.

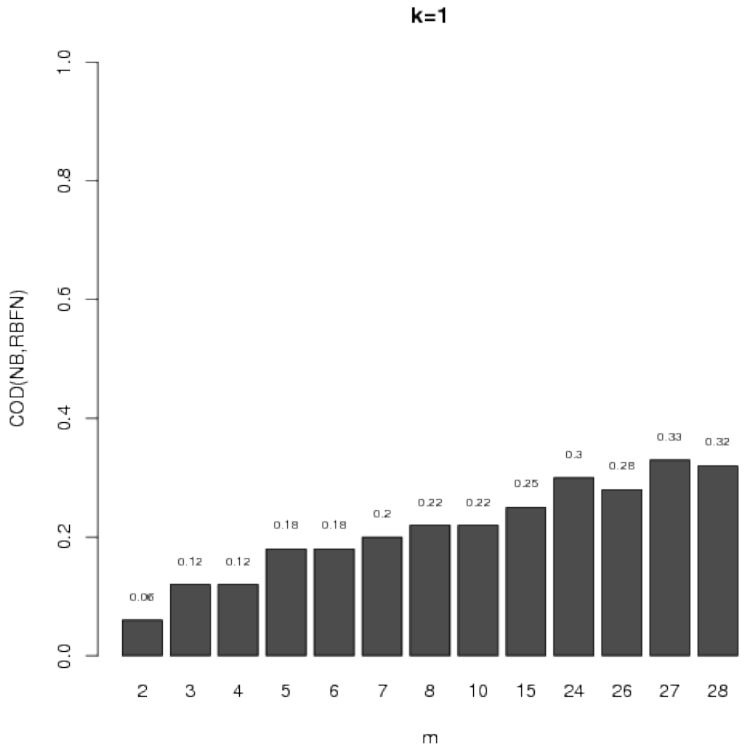


Figure 6.4: COD vs m for $k = 1$ It shows that the difference in behavior between NB and RBFN becomes large in general as m increases.

Interestingly, however, even with the increase in COD, the relative similarity remains the same with NB and RBFN clustering together first for all values of m. For the sake of space, we do not show the dendrograms here. So, while the absolute value of COD indicates that the amount of similarity between RBFN and NB decreases as m increases, the relative value of COD suggests that, of all learning algorithms considered here, RBFN with $k = 1$ is closer in behavior to NB than any other algorithm.

Of course, very few people would think of $k = 1$ as the best setting for RBFN, as it tends to nullify the advantage offered by "localization" and hybrid learning. The foregoing analysis confirms that such a choice would also not be very judicious since it would only cause RBFN to behave like NB, clearly a computational overkill in most cases (we return to

the issue of complexity shortly). As stated above, Weka uses $k = 2$ as its implementation's default value. We thus turn to the case when $k > 1$.

6.4.2 RBFN with $k > 1$

As with larger values of m for $k = 1$, a direct comparison of discriminant functions and geometry of decision boundaries does not generalize easily to values of $k > 1$. Hence, we again resort to an empirical analysis of the value of COD as k increases. We expect that for $k = 2$ the similarity between RBFN and NB may still hold, and possibly for a few larger values of k. However, as k gets larger, it would seem that the amount of similarity should eventually decrease as RBFN is likely to begin overfitting, while NB's behavior remains unchanged. Figure 6.5 shows how the value of COD evolves with increasing values of k.

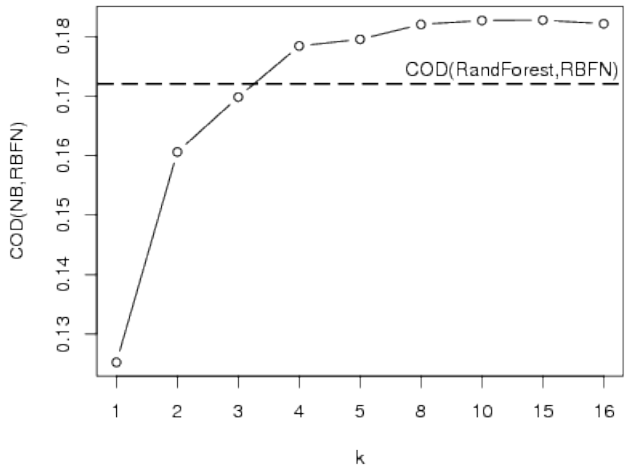


Figure 6.5: COD vs k when $m = 2$. For $k \leq 3$, RBFN clusters first with NB but clusters with RandForest when $k > 3$. This indicates that the similarity of RBFN and NB is getting weaker with increasing k.

As expected, COD increases with k. Interestingly, RBFN and NB cluster together first only up to $k = 3$. For larger values of k, the cluster is "broken" and RBFN begins to cluster first with Random Forest. However, notice that the value of COD seems to reach

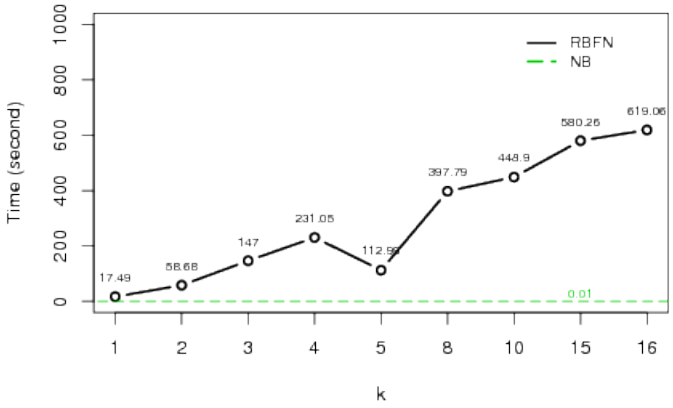
a kind of plateau around 0.18, so that, although they no longer cluster together first, the amount of difference between RBFN and NB seems to remain constant across values of $k > 3$.

At this point, we must ask an important question. We have already shown that choosing $k = 1$ is probably not judicious. Weka chooses $k = 2$, which still maintains significant behavior similarity between RBFN and NB. So the question is, should we choose larger values of k, and if we do, what kind of improvement might we obtain in terms of predictive accuracy for RBFN? Figure 6.6 shows how RBFN's training time and predictive accuracy are affected by the value of k.

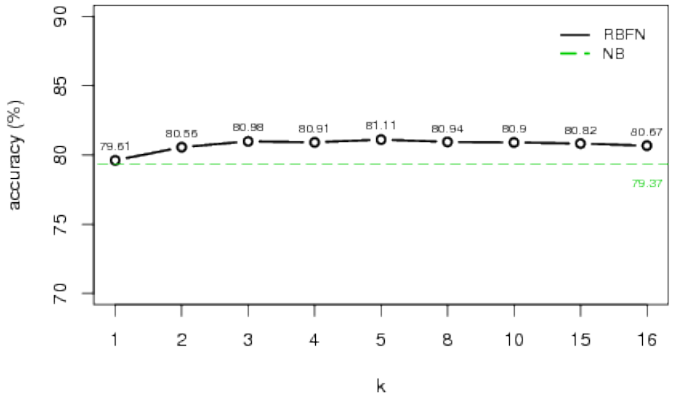
It is clear that while the training time of RBFN increases dramatically (at least linearly) with k, there is no significant change in predictive accuracy. This behavior is likely due to the overfitting alluded to above. Large values of k typically make sense in function approximation or regression tasks, less so in classification tasks. It would appear, therefore, that Weka's choice of $k = 2$, which results in km basis functions, is a good compromise between training time and predictive accuracy. Hence, we are left with a relatively high degree of similarity between RBFN and NB in realistic settings for both algorithms.

6.4.3 Consequence

There is an important practical consequence to this, since there is a significant difference in computational complexity between NB and RBFN. Let T denote the number of training instances and n denote the number of attributes. Then NB is $O(Tn)$ [25]. Since RBFN is a hybrid learner, we must consider the complexity of each of its parts. The complexity of k-means is $O(TnkI)$, where I is the number of iterations (e.g., see [82]). Since Weka's RBFN performs k-means for each class separately, its complexity is $O(TnmkI)$. In Weka's RBFN, the weights for logistic regression use the quasi-Newton method. The complexity of the quasi-Newton method is $O(W^2)$, where W is the size of the weight vector (e.g., see [35], p. 197). Here, $W = H(m-1) = km(m-1)$, so that the complexity of logistic regression



(a) Training Time



(b) Predictive Accuracy

Figure 6.6: NB and RBFN's Training Time and Accuracy with Increasing k

is $O(k^2m^4)$. It follows that RBFN is $O(Tnmk I + k^2m^4)$, which is clearly much worse than NB's complexity.

From a practical standpoint, this theoretical difference also translates into significant time differences, as shown on Figure 6.6(a). Table 6.2 shows the ratio of RBFN's to NB's training time over our selection of datasets. On several occasions, RBFN takes several hours, while NB only requires a few seconds. Overall, RBFN is more than 22 times slower than NB on half of the datasets.

Table 6.2: Ratio of RBFN's to NB's Training Time

Statistic	**Time Ratio**
Mean	46,056
Maximum	57,060
Upper Median	78
Median	22
Lower Median	10
Minimum	5

If that kind of difference does not give rise to a significant difference in predictive accuracy, one may be tempted to always use NB. However, recall that our empirical analysis relies on observations *averaged* across many datasets. So, while the observed similarity between RBFN and NB holds in the aggregate, there are noticeable local variations. As a matter of fact, the value of COD between NB and RBFN over our 85 continuous datasets ranges between 0 and 0.88 ($\mu = 0.16, \sigma = 0.22$), while their corresponding difference in predictive accuracy ranges between 0 and 20.83% ($\mu = 2.83\%$, $\sigma = 3.98\%$).

We thus turn our attention to analyzing these local variations, i.e., discovering what types of tasks RBFN is likely to be superior to NB. In other words, our similarity results suggest that unless RBFN performs significantly better than NB, we should simply use NB. What we would like to know is when it makes sense to incur the extra computational complexity of RBFN, and when it can be avoided at no significant loss to predictive accuracy.

6.5 To RBFN or Not to RBFN

In this section, we use metalearning in an attempt at characterizing the types of tasks on which the difference of behavior between RBFN and NB is significant and gives preference to RBFN. The reason we rely on metalearning is that, while it is easy to design simple cases where the performance of RBFN is much better than the performance of NB, these are generally somewhat pathological cases, and it is difficult to come up with an analytical form for more general cases.

6.5.1 A Simple Case of Preference for RBFN

As per the above discussion, we assume $k = 2$ for RBFN. It is easy to show that NB is weak on binary classification tasks where there are an equal number of non-linearly separable target values, as illustrated in Figure 6.7.

OOOOOXXXXXOOOOOXXXXX

Figure 6.7: Non-linearly Separable One-dimensional Binary Classification Task

The analytical form of NB for such a one-dimensional dataset is as follows [52].

$$\begin{aligned} P(y = \mathrm{O}|x) &= \frac{1}{1 + e^{\omega_0 + \omega_1 x}} \\ P(y = \mathrm{X}|x) &= 1 - P(y = \mathrm{O}|x) \end{aligned}$$

where

$$\begin{aligned} w_0 &= ln\frac{1 - P(y = \mathrm{O})}{P(y = \mathrm{O})} + \frac{\mu_{\mathrm{O}}^2 - \mu_{\mathrm{X}}^2}{2\sigma^2} \\ w_1 &= \frac{\mu_{\mathrm{X}} - \mu_{\mathrm{O}}}{\sigma^2} \end{aligned}$$

Since $P(y = \mathrm{O}) = P(y = \mathrm{X})$ and $\mu_{\mathrm{O}} = \mu_{\mathrm{X}}$, it follows immediately that $\omega_0 = \omega_1 = 0$, so that $P(y = \mathrm{O}|x) = P(y = \mathrm{X}|x) = 0.5$, and NB's accuracy is 50%.

On the other hand, the analytical form of RBFN is as follows (see section 6.3.2).

$$\begin{aligned} P(y = \mathrm{O}|x) &= \frac{e^{\omega_0 + \sum_{i=1}^{4} \omega_i K_i(x)}}{1 + e^{\omega_0 + \sum_{i=1}^{4} \omega_i K_i(x)}} \\ P(y = \mathrm{X}|x) &= 1 - P(y = \mathrm{O}|x) \end{aligned}$$

where two of the kernel functions, say K_1 and K_2, are associated with class o, and the other two, say K_3 and K_4, are associated with class x. Given the geometry of the problem, K_1 will

Table 6.3: Summary of T-test for Accuracy Difference Between RBFN and NB

Winner	**# Datasets**
RBFN	18
NB	5
Tie	62

capture the first *ooooo* sequence, K_3 will capture the first $xxxxx$ sequence, $K2$ will capture the second *ooooo* sequence, and K_4 will capture the second $xxxxx$ sequence. Hence, by training the ω_i's, RBFN will be able to discriminate between the two classes perfectly, such that RBFN's accuracy is 100%.

6.5.2 Building a Selection Metamodel

From a practical standpoint, in all cases where NB has significantly higher predictive accuracy, as well as those where there is no significant difference between NB's and RBFN's predictive accuracy, one should run NB since this will produce the best accuracy in the least amount of time. Conversely, in all other cases, where RBFN does perform significantly better than NB, one might wish to run RBFN in spite of its much greater computational cost. We propose to use metalearning to provide a general mechanism for users to make such a decision. The metamodel is built to discriminate between NB and RBFN.

As stated above, there is some variance in the performance of NB and RBFN on individual tasks. We wish to know when these differences are significant. To find out, we run an unpaired student's t-test on each of our datasets using 10-fold cross-validation, with a p-value of 0.05. A summary of the results is in Table 6.3.

At the metalevel, each of our 85 datasets is characterized by its values over a predefined set of metafeatures, as in other metalearning for classification approaches (e.g., see [2, 18, 59]). Our set of 38 metafeatures consists of a combination of statistical measures and a small set of landmarkers, including the following.

- `lgE`: log of the number of examples

- `lgREA`: log of the ratio of the number of examples to the number of attributes
- `numClasses`: number of target classes
- `numInstsPerClass`: ratio of the number of examples to the number of target classes
- `RMajorClass`: probability of the majority class
- `landmarkerMajorityGuesser`: majority class landmarker
- `landmarker1NN`: 1-NN landmarker
- `landmarkerNaiveBayes`: NB landmarker
- `meanCovarianceMatrix`: average over all target values of the class-dependent means of attribute pair covariances
- `normalizedKurtosis`: normalized kurtosis
- `entireEntropy`: class entropy

For each dataset, its corresponding meta-example is generated and labeled with one of two target values: `rbfn` when RBFN significantly outperforms NB (18 instances), and `nb` otherwise (67 instances). The default accuracy, i.e., the accuracy when one predicts the most frequent class, is $\frac{67}{85} = 78.8\%$. For the sake of comprehensibility, we use Weka's J48 as the metalearner, with the following parameter values:

- confidenceFactor: 0.25
- pruning: true
- minNumObj: 2 (the minimum number of instances per leaf)

Figure 6.8 shows the induced decision tree. Interestingly, the highest decision nodes in the tree have to do with the size of the dataset. This is consistent with findings involving NB and logistic regression, where it is stated that NB usually performs better for smaller number of training examples, while logistic regression performs better with larger training sets. Recall that Weka's RBFN uses logistic regression to learn the output layer's weights. Our selector

```
J48 pruned tree
------------------

lgE <= 7.3025
|   lgREA <= 1.4032: nb (40.0)
|   lgREA > 1.4032
|   |   lgREA <= 1.7619: rbfn (5.0/1.0)
|   |   lgREA > 1.7619
|   |   |   meanCovarianceMatrix <= -0.0026: rbfn (2.07/0.07)
|   |   |   meanCovarianceMatrix > -0.0026
|   |   |   |   meanCovarianceMatrix <= 0.0087: nb (23.79)
|   |   |   |   meanCovarianceMatrix > 0.0087
|   |   |   |   |   numInstsPerClass <= 153: nb (2.14)
|   |   |   |   |   numInstsPerClass > 153: rbfn (2.0)
lgE > 7.3025: rbfn (10.0)
```

Figure 6.8: Decision Tree Selector

Table 6.4: Confusion Matrix for Decision Tree Selector

		Pred.	
		NB	RBFN
Actual	NB	63	4
	RBFN	7	11

tree suggests that when there are many examples, the predicted winner is RBFN (top branch: `lgE > 7.3025: rbfn (10.0)`). Conversely, when there are few examples, the predicted winner is NB (top branch: `lgE <= 7.3025 and lgREA <= 1.4032: nb (40.0)`).

The tree's confusion matrix, using 10-fold cross-validation, is shown in Table 6.4. Its overall accuracy is 87%, a significant increase over the default, which suggests that the metamodel's predictions will yield performance improvements in the practical use of RBFN and NB.

Looking closer at the selection task, we notice that the types of error metamodel makes have different costs. When RBFN is mistakenly predicted as NB, the user loses accuracy but saves in training time. On the other hand, when NB is mistakenly predicted as RBFN, the user loses accuracy *and* incurs unnecessary additional training time. Clearly, the second type of error is more costly than the first. At the risk of sacrificing comprehensibility for the sake of a better model, we consider ensemble-like learning algorithms for the metalearner.

Table 6.5: Confusion Matrix for Best Selector Model

Actual \ Pred.	NB	RBFN
NB	66	1
RBFN	6	12

Our goal is to reduce as much as possible the second type of errors while not increasing the first type. Our best metamodel is obtained by applying the idea of rotation forest with J48graft as the base learner and the default 50% of instances to be removed. The model's overall accuracy is 91.8%, and its improved confusion matrix is shown in Table 6.5. Not only is this model more accurate, it only makes one false prediction for NB.

6.6 Discussion

Since we use Weka's default implementations, it is worthwhile to investigate whether the behavior similarity between RBFN and NB is restricted to the assumptions of these implementations. We first consider the assumption of Gaussian kernels and then the assumption of class-dependency of said kernels. In both cases, we use our idea of clustering. To test the impact of the choice of basis functions, we perform clustering with five additional implementations of RBFN, as follows. Each implementation is denoted by a pair consisting of 1) the unsupervised learning algorithm used to build the hidden layer (EM or k-Means) together with the type of radial basis function used in the hidden layer (Gaussian or Thin Plate Spline), and 2) the supervised learning algorithm used to compute the weights of the output layer (Logistic Regression, Single Layer Perceptron, LibLINEAR or LibSVM).

- EMG+Log
- KG+LibLINR
- KG+SLP
- KG+LibSVM

- KThin+Log

Figure 6.9 shows the dendrogram obtained by averaging COD over all 85 continuous datasets using the same clustering algorithm as in Section 6.1. The label RBFN corresponds to Weka's default implementation (KG+Log). The dendrogram suggests that the similarity between RBFN and NB is affected more by the choice of basis functions (or kernels) than by the choice of the supervised weight update method. Four out of the five Gaussian kernel RBFNs, with different supervised learning algorithms, cluster together first, followed by NB. The fifth Gaussian kernel RBFN, with a rather different supervised learning algorithm (SVM), clusters a little later. Finally, the non-Gaussian, thin plate spline kernel RBFN, with the standard logistic regression algorithm as supervised learning algorithm, is most unlike NB. Hence, conclusions about the similarity between RBFN and NB are valid as long as both use Gaussian kernels.

The other aspect of Weka's implementation of RBFN that likely impacts the observed similarity between RBFN and NB is the fact that kernels are class-dependent rather than shared across classes. To verify that this is the case, we perform clustering with three variations of shared kernels as follows.

- RBFN+Raw1. The hidden layer consists of a single hidden node.
- RBFN+Raw2. The hidden layer consists of two hidden nodes.
- RBFN+Raw. The hidden layer consists of m hidden nodes. This is similar to $k = 1$ in the standard Weka implementation, except that here there is no one-to-one correspondence between a hidden node and a target class.

Figure 6.10 shows the resulting dendrogram. Again, the label RBFN corresponds to Weka's default implementation. The dendrogram clearly shows that there is no similarity between RBFN and NB when the basis functions are shared across classes. This should not be completely surprising given the analysis of section 6.4. Interestingly, when considering accuracy over our 85 datasets, RBFN is significantly (p=0.05) better than RBFN+Raw on

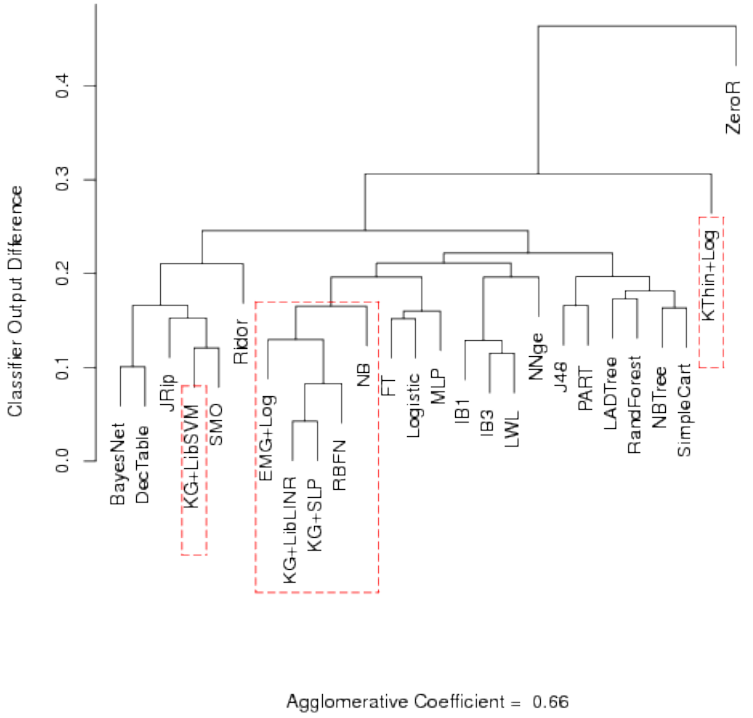


Figure 6.9: COD-based Clustering with Additional RBFN Implementations

27 datasets, while RBFN+Raw wins only once, and there is no significant difference on the remaining 57 datasets.

Hence, we are certainly not claiming that RBFN and NB behave similarly in general. We have simply shown that the assumptions embedded in Weka's implementations of these algorithms, specifically the use of Gaussian kernels and the choice of 2 class-dependent basis functions per class, make them behave rather similarly over a broad range of datasets, and certainly more similarly than any other algorithm in our wide selection of learning algorithms. Interestingly, Weka's assumptions are rather reasonable.

- Gaussian kernels seem appropriate for classification tasks.

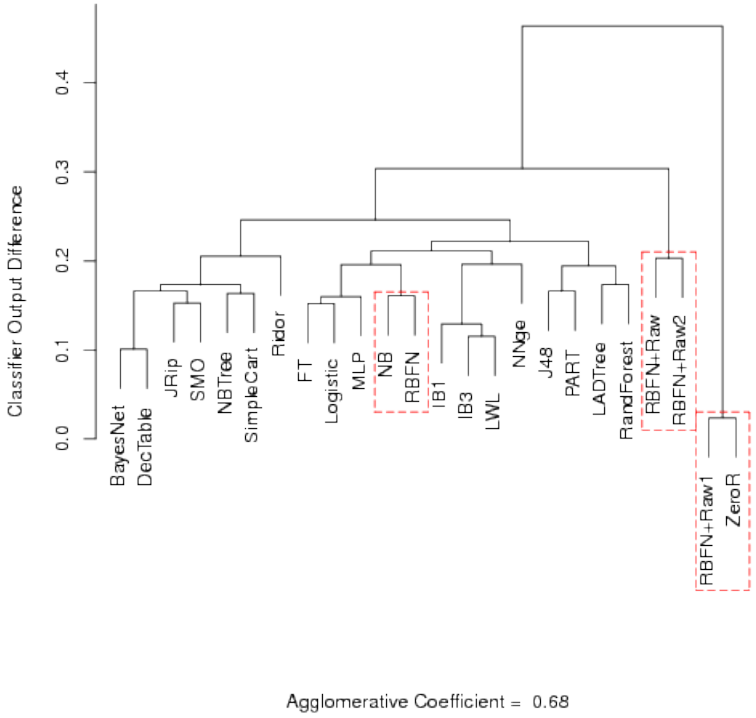


Figure 6.10: COD-based Clustering with Shared Kernels Implementations

- Class-dependent kernels are natural and seem to outperform kernels shared across classes.[5]
- $k = 1$ makes little sense and $k > 2$ incurs unnecessary computational cost for no significant gain in accuracy.

As a result, our finding has important practical consequences, all of which having to do with being careful about the type of tasks under study.

- *Use of default parameter values.* Most practitioners, who lack the needed know-how, confine their use of tools to their default parameter settings. As seen here, there is potential "danger" in doing so. Indeed, if one is considering the use of RBFN, but the

[5]More work is clearly needed to validate this preliminary finding, but our point here is simply that Weka's implementation is at least legitimate, i.e., not unwittingly or purposely sub-optimal.

task at hand is such that the similarity is strong and NB would give a similar answer, it would be cheaper to use NB. The metamodel described in section 6.5.2 is an attempt at avoiding this problem for RBFN and NB.

- *Ensemble learning.* It has long been known that the value of an ensemble resides in the diversity of its constituent learning algorithms. Researchers who design ensembles do not always take the time to verify the actual level of diversity in their selection, but may rely on existing taxonomies (e.g., Weka's classification of algorithms in different model classes) or preferences. Again, as pointed out above, current taxonomies and perceptions about RBFN and NB would likely suggest that both be used in an ensemble. Yet, the foregoing analysis clearly shows that in many cases there would be little, if any, added value in including both since their behaviors are so similar on a broad set of learning tasks.
- *Metalearning for algorithm selection.* Much of the work in metalearning has focused on building models that can select (or rank) classification algorithms. There are two issues here. First, even when differences are imperceptible in the aggregate (as in the present study), they may matter at the task level, and further efforts should thus be expanded in finding automatic mechanisms to map learning tasks to learners effectively. Second, when strong similarities exist among algorithms, it may be counterproductive to build models that try hard to tell them apart. Instead, one may consider clustering algorithms based on similarity and devising novel selection mechanisms where the metalearner selects among cluster of behaviorally similar algorithms rather than among individual algorithms. This is the subject of current work [46].

Beyond the present study of RBFN and NB, we would point out that one would not expect that the amount of diversity among learning algorithms continually increases. The set of problems we attempt to solve, or test our algorithms against (e.g., UCI), varies (increases) much more slowly than the number of new algorithms or variants thereof being created. Given this almost fixed set against which to test, it should be clear that one would

expect some algorithms to have similar, yet uncovered, performance or behavior on that set. In this paper, we have focused on a specific pair of algorithms, highlighted by the clustering we performed on our selection of 21 algorithms. We think that the approach has merit and that computing natural clusterings may bring to light interesting, otherwise difficult to anticipate, similarities among learning algorithms. Furthermore, results on a slightly extended set of algorithms and a larger collection of datasets from UCI seem to indicate that there is strong pairwise correlation in terms of accuracy performance among current learners. As a result, one may wonder whether designing yet another learning algorithm for UCI is as promising an area of research as designing a new learning algorithm for an as-yet unexplored area of the learning space.

6.7 Conclusion

In this paper, we have offered a detailed comparison of the behavior RBFN and NB, based on a combination of analytical tools and empirical results. We have shown that RBFN and NB indeed exhibit a high level of similarity across a broad range of learning tasks, and shed some light as to why this may be and what the scope of applicability of that observation is. We have concluded that the observed similarity is a direct result of the assumptions made by Weka's implementations of RBFN and NB. However, we have also argued that said implementations were natural and reasonable, so that the observed similarity is indeed of practical import.

Having established that the observed similarity in no way suggests that the two algorithm have identical behavior, but is true in the aggregate, we turned to metalearning for the development of a selection metamodel that could discriminate between RBFN and NB, and in particular accurately predict when the increase in training time caused by RBFN would be worthwhile, i.e., result in higher predictive accuracy for the learning task under consideration.

Finally, we have discussed some of the consequences of our finding, both specific to RBFN and NB, and more generally across other learning algorithms. We argue that discovering similarity (resp. difference) among learning algorithms may prove useful in ensemble learning and metalearning, and will help us get a better understanding of our science. In cases where similarity in generalization performance is not matched by similarity in computational complexity, as is the case for RBFN and NB, we may also be able to use a faster model for the same performance.

Chapter 7

Clustering-based Metalearning for Algorithm Selection

Abstract

Classification algorithm selection is one of the important open research questions in data mining, and one whose answer has tremendous value for practitioners. In recent years, metalearning has emerged as a viable solution to this problem. To make algorithm selection most useful, one would ideally like the largest possible set of candidate algorithms to select from. However, given the relatively small number of examples at the metalevel, this makes the metalearning task challenging, both because the ratio of examples to classes is very small, and because there are serious risks of overfitting due to underlying similarities among algorithms. To alleviate these problems, we propose to 1) cluster algorithms based on behavior similarity, and 2) redefine the metalearning task as mapping classification tasks to clusters of behaviorally-similar algorithms. Experiments with a wide range of classification tasks and algorithms demonstrate promise. In particular, the clustering-based selection model is more effective than typical selection and ranking models.

7.1 Introduction

One of the challenges faced by data mining practitioners is the selection of the most accurate algorithm for their classification tasks. Given that each algorithm performs well only on a subset of classification task —a direct consequence of the No Free Lunch theorems [66, 85, 86], and that there is a growing number of available algorithms, finding the best algorithm for a particular classification task is indeed becoming increasingly difficult.

While hiring human experts may help, it is also costly and often biased. Since no expert can be expected to know all algorithms, decisions tend to be influenced by personal experience and preferences. Furthermore, comparatively little is known about each algorithm's area of expertise, or what characterizes tasks on which it performs well and tasks on which it performs poorly. Hence, human decisions run the risk of being suboptimal. Recent research suggests that automatic tools can sometimes bring up solutions that have thus far eluded human experts (e.g., see [11]). As an alternative, one could simply run all available algorithms on the given task and choose the algorithm with the best performance. However, this is not without problems either. In particular, one may not have access to the candidate algorithms, and even if they do, it might be computationally prohibitive to execute all possible algorithms.

For the practitioner, what is needed is an automatic system capable of returning the most suitable algorithm for his/her task. Metalearning, or the use of data collected from the application of data mining to build metamodels that map classification tasks to algorithms, has proven a viable solution for the design and implementation of such systems [16]. Metalearning for algorithm selection is typically formulated as a special case of Rice's general framework for algorithm selection [63, 71]. Let L be a set of learning algorithms for classification and T be a set of classification tasks, such that for each $t \in T$, $b_L(t)$ represents the algorithm in L that gives the best predictive accuracy on t. Since classification tasks may be unwieldy to handle directly, some characterization of tasks by a set of metafeatures is generally used, such that for each $t \in T$, $c(t)$ denotes the characterization of t. Then, metalearning

takes the set $m = \{< c(t), b_L(t) >: t \in T\}$ as a training set and induces a metamodel that, for each new classification task, t, predicts the model from L that will perform best on t. Alternatively, one may induce a metamodel that predicts a complete ranking of algorithms from L, thus providing the user additional information when the algorithm predicted best exhibits what appears to be a poor performance (e.g., see [12, 18]).[1]

In order to maximize value for the user, the set L should be such that it offers as broad as possible a coverage of the task space so that any task presented by the user will fall into the area of expertise of at least one learner in L. There are two possible approaches to achieving maximum (under L) coverage: 1) know each learner's area of expertise and select one learner per area, or 2) use all learners in L. It should be clear that if we knew the area of expertise of all learning algorithms, the problem of algorithm selection would be largely solved. Hence, the only practical alternative is to make L large and consider all of the learning algorithms in L. However, this too presents some challenges. First, because the set m of training examples at the metalevel (i.e., documented base-level learning tasks) is relatively small, the ratio $\frac{|m|}{|L|}$ of training examples to target classes decreases as the size of L increases. This, in turn, makes it difficult for any metalearner to induce an accurate metamodel. Second, there is a significant risk of overfitting due to the likelihood of similarities among learning algorithms in L.[2] To illustrate, consider the simple metalearning task shown in Table 7.1. This task consists of three training (meta-)examples, each corresponding to a specific base-level learning task characterized by two continuous-valued metafeatures. Each example is also labeled with the algorithm known to perform best on the associated learning task. Let us further assume that the behavior of algorithm l_A is very similar to that of algorithm l_B. Since l_A and l_B are different target labels, the metalearner would try to induce a model that discriminates between instances 1 and 2. In that attempt, it risks overfitting. Indeed,

[1] In the single algorithm prediction approach, the user has no further information as to what other algorithm to try. In the ranking approach, the user may try the second best, third best, and so on, in an attempt to improve performance.

[2] While this likelihood clearly increases with $|L|$, it is also non negligible for smaller values of $|L|$ since, as stated above, we generally do not know individual areas of expertise.

Table 7.1: Simple Metalearning Task

Example	**Feature 1**	**Feature 2**	**Best Alg**
1	0.45	0.68	l_A
2	0.43	0.66	l_B
3	0.25	0.36	l_C

suppose that a new, previously unseen dataset is presented to the system with metafeatures $< 0.44, 0.67 >$. What would the metamodel output in this case? Depending on how it handled the discrimination between instances 1 and 2 during learning, it would return either l_A or l_B. However, since the behaviors of l_A and l_B are similar, the actual best algorithm for this new instance may also be recorded as either l_A or l_B. As a result, there is a 50% chance that the metamodel predicts l_A when l_B is the new instance's actual label, and 50% chance that it predicts l_B when l_A is the correct answer.

As an effective way to overcome the foregoing challenges, we propose a new algorithm selection model based on clustering. The basic idea is simple. We group the target values in the metadataset in terms of the behavior similarity of the corresponding algorithms, and induce a metamodel that predicts clusters rather than individual algorithms. Returning to the example of Table 7.1, if l_A and l_B were to be grouped together, as per our proposed approach, then both instances 1 and 2 would have the same label. The metalearner would not have to try to needlessly separate them, and the correct answer would be returned for the new instance since no matter what its actual label, the corresponding algorithm would be in the cluster predicted by the metalearner. This novel approach does not only simplify the metalearning task (i.e., improving the ratio of training examples to target classes) and greatly reduce the risk of overfitting, it is also more accurate than traditional algorithm-based selection.

The paper is organized as follows. In section 7.2, we briefly review previous work on metalearning for algorithm selection. In section 7.3.3, we present our behavior similarity metric, describe the resulting clustering of a broad selection of learning algorithms, and introduce our clustering-based algorithm selection system. In section 7.4, we compare

clustering-based selection with algorithm-based selection and demonstrates its superiority. Finally, section 7.5 concludes the paper.

7.2 Related Works

Metalearning for classification algorithm selection probably finds its origins in STABB, which showed that a learner's bias could be adjusted dynamically [77], and VBMS, which learned to select the best among (three) algorithms as a function of (two) dataset characteristics [62]. Later, the MLT project (ESPRIT Nr. 2154) produced a user guidance system, called Consultant-2, which although built through knowledge engineering rather than metalearning, stands out as the first automatic tool that systematically relates application and data characteristics to classification learning algorithms [23, 40]. About the same time, the StatLog project (ESPRIT Nr. 5170) extended VBMS by considering a larger number of dataset characteristics, together with a broad class of candidate algorithms for selection [15, 30, 51]. StatLog identified 16 metafeatures and used them in an attempt to find relationships with accuracy. It considered 23 machine learning algorithms and 22 datasets. Continuing in the tradition, the METAL project (ESPRIT Nr. 26.357) was the first large-scale project exclusively focused on the design of automatic user guidance systems for algorithm selection via metalearning [16, 80]. METAL covered 53 datasets, and made a number of significant contributions to metalearning, including designing novel task characterizations, such as, advanced statistics [26, 41, 48], landmarking [7, 28, 59], and model-based approaches [6, 9, 56]. In addition, the project introduced the idea of rankings rather than best-in-class selection to give more information to the user [18], and implemented it in the Data Mining Advisor (DMA), a Web-enabled prototype assistant system [33].

In addition to work done in the context of the foregoing projects, others have published results on similar attempts. For example, another rather comprehensive attempt at algorithm selection was carried out in a set of experiments involving 100 datasets, 31 metafeatures, and 8 learning algorithms [2]. In one experiment, the authors compared the performance of

the learning algorithms with respect to a multi-criteria performance measure, called *relative weighted performance measure*, which combines weighted ranking of average accuracy and computational time. When equal weights are given to both components, the very simple (and fast) OneR (that uses a minimum-error attribute for prediction) algorithm turned out to be the best one since its computational advantage far outweighed its poor accuracy. Unfortunately, the proposed measure could hardly distinguish the performance of C4.5, PART, KD, and NB. Furthermore the computational cost associated with extracting the values of metafeatures is ignored. Finally, we note that a few researchers have attempted to predict actual algorithm performance directly [10, 34, 74].

7.3 Clustering-based Metalearning

In this section, we discuss the design of our proposed clustering-based metalearning system for algorithm selection.

7.3.1 Choice of Learning Algorithms

To facilitate access to our results by as wide an audience as possible, we consider well-established learning algorithms from Weka, one of the most popular and richest open-source data mining software package [83]. For simplicity, and to cater to novice users who are unlikely to have the know-how to tune parameters, only the default versions of the following 21 algorithms are taken into consideration.

1. NaïveBayes (NB)
2. Bayesian Network (BayesNet)
3. Logistic Regression (Logistic)
4. Radial Basis Function Network (RBFN)
5. Multilayer Perceptron (MLP)
6. Sequential Minimal Optimization (SMO)

7. One-nearest Neighbor (IB1)
8. Three-nearest Neighbor (IB3)
9. Locally-weighted Learning (LWL)
10. Classification and Regression Tree (SimpleCart)
11. Multi-class Alternating Decision Tree (LADTree)
12. Functional Tree (FT)
13. C4.5 Decision Tree (J48)
14. NB Tree (NBTree)
15. Random Forest (RandForest)
16. Decision Table (DecTable)
17. Ripper (JRip)
18. Nearest Neighbor with Generalization (NNge)
19. Partial Tree Decision Rule Learning (PART)
20. Ripple-Down Rule Learning (Ridor)
21. Mode Predictor or Majority Class Learner (ZeroR)

We note at the onset that Weka's algorithms are organized according to an internal taxonomy that consists of a number of model classes, including tree-based, rule-based, instance-based, probability-based and function-based approaches. It would, of course, be possible to let this taxonomy be the basis for a clustering of our selected algorithms. However, recall that our metamodel will select clusters rather than algorithms. Hence, our objective is to group together algorithms whose behavior is similar, so that when a cluster is selected, there will be little difference in the performance of any of the algorithms in the cluster on the learning task under consideration. Weka's taxonomy is human-generated, and thus may focus on intuitive criteria that miss underlying, non-obvious behavior similarity

among algorithms. We found one such striking similarity between the probability-based, generative naïve Bayes learning algorithm and the function-based, discriminative radial basis function network learning, which is the topic of a separate study [45]. As a result, we prefer to rely on automatic clustering.

7.3.2 Choice of Behavior Distance Measure

Our chosen measure of behavior distance between algorithms is the classifier output difference (COD), one of the diversity measures typically used in ensemble learning [58]. Given two learning algorithms l_1 and l_2, $COD(l_1, l_2)$ is the probability that l_1 and l_2 make different predictions. As we do not typically have access to actual probabilities, we use frequency-estimates for COD values as follows. Let h_1 be the model induced by l_1 and h_2 be the model induced by l_2 on some learning task t. Then

$$COD(l_1, l_2, t) = \frac{\mid \{x \in t : h_1(x) \neq h_2(x)\} \mid}{\mid t \mid}$$

In practice, we compute COD on a number of learning tasks and average the values, so that:

$$COD(l_1, l_2) = \frac{1}{\mid T \mid} \sum_{t \in T} COD(l_1, l_2, t)$$

While global measures, such as accuracy, provide only an idea of average performance over all instances, COD captures local variations among instances. For example, even though l_1 and l_2 may have the same overall accuracy on some task, they may be acting very differently on that task, with l_1 misclassifying examples that l_2 classifies correctly, and vice versa. There are two reasons why we favor COD for clustering classification learning algorithms. In recent work on unsupervised metalearning, we have indeed shown that [47]:

1. Unlike all other considered local measures, COD satisfies the properties of a metric.

2. COD is strictly stronger than accuracy, in the sense that if two algorithms are close based on COD, they also have similar accuracy.

We note that computing COD over a set of algorithms is computationally intensive. Each algorithm must be run against each dataset, usually with cross-validation, and all predictions for all training instances must be recorded. This takes at least the running time of the most expensive learning algorithm in L on the heaviest learning task, assuming parallel computation is available. Pairwise comparisons of predictions and averaging must subsequently be performed. However, this cost need only be incurred once.

7.3.3 Clustering of Classification Learning Algorithms

Using COD distance information averaged over 129 datasets from various sources including the UCI Machine Learning Repository [4] and the Gene Expression Machine Learning Repository [75], we clustered our 21 algorithms using complete-linkage, hierarchical agglomerative clustering (see [47] for details). The result is shown in Figure 7.1.

Although it shows all possible clusterings, the hierarchical clustering algorithm does not provide information as to which of these is preferable over the others. The choice of a specific clustering is typically made by selecting a level at which to cut through the dendrogram, and defining the clusters as the groups of algorithms hanging from the subtrees whose top branches intersect with the horizontal line corresponding to the chosen level, as illustrated on a simple example in Figure 7.2.

It is clear that the selection of a level, or cut point, greatly impacts the actual clustering, and thus the proposed metalearning task. If the level is too low (i.e., the corresponding line cuts through the dendrogram just above the leaf nodes), then each cluster contains a single learning algorithm, and our approach degenerates into the typical algorithm selection approach. Conversely, if the level is too high (i.e., the corresponding line cuts through the dendrogram just above its root), then all classification algorithms belong to a single cluster,

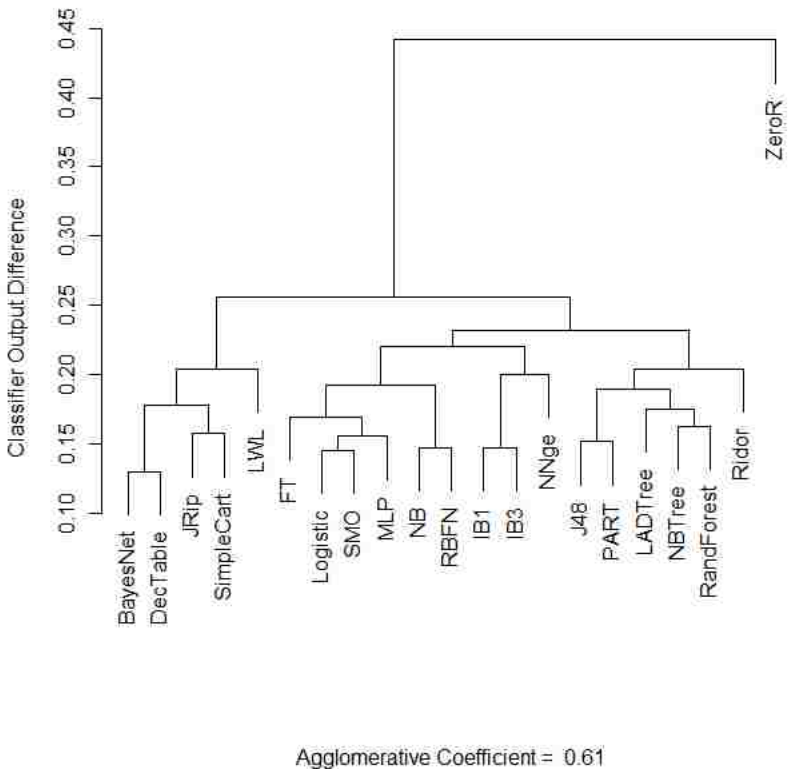


Figure 7.1: *COD*-based Clustering of Learning Algorithms

and our approach is no better than no algorithm selection at all. Finding an appropriate level at which to cut is critical to the success of our approach.

7.3.4 Choice of Level

Recall that L denotes our set of 21 classification learning algorithms, T denotes our set of 129 classification tasks, and for each $t \in T$, $b_L(t)$ denotes the algorithm in L that gives the best predictive accuracy on t, while $c(t)$ denotes the characterization of t by some set of relevant metafeatures. At the metalevel, each $t \in T$ gives rise to a training meta-example. In the traditional algorithm selection setting, meta-examples are of the form $< c(t), b_L(t) >$. In our proposed clustering-based selection setting, given a clustering $\mathbb{C}$, each cluster in $\mathbb{C}$ is

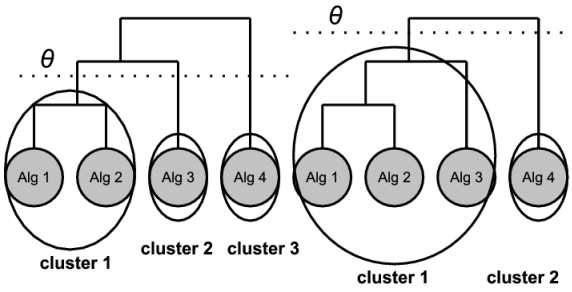


Figure 7.2: Effect of Level (here shown as θ) on the Resulting Clustering

given a label, and meta-examples are of the form $< c(t), cl(b_L(t)) >$, where $cl(b_L(t))$ is the label of the cluster to which $b_L(t)$ belongs.

From the meta-examples, the metalearner induces a metamodel that predicts a cluster in $\mathbb{C}$ from which a specific classification algorithm may be chosen. The accuracy of the metalearner is computed as follows.

$$\begin{aligned} Acc_{Meta}(\mathbb{C}) &= \frac{\mid \{t \in T_e : cl_{pred}(t) = cl(b_L(t))\} \mid}{\mid T_e \mid} \\ &= \frac{\mid \{t \in T_e : b_L(t) \in cl_{pred}(t)\} \mid}{\mid T_e \mid} \end{aligned}$$

where T_e is the test set and $cl_{pred}(t)$ is the cluster predicted by the metalearner for t. In other words, accuracy depends on the number of times the predicted cluster contains the best algorithm. When all clusters are singletons, we obtain the standard accuracy of the corresponding algorithm selection metalearner.

Assume for now that, given the predicted cluster, the user simply picks one algorithm at random from that cluster to execute on his/her target task. We revisit this issue later and show how to assist the user further by providing a ranking of algorithms. For the present discussion, the selection approach is sufficient. Note that most clusters will not be singletons so that there will be variance in the accuracy on the user's target task, from the best performing algorithm to the worst performing algorithm in the selected cluster. The key idea behind our behavior-based clustering is to try to minimize this variance by ensuring

that algorithms in a cluster are as similar as possible (i.e., relatively small COD) so that any one of them is likely to be as good as the best one, provided the best one is in the group. Recall that small values of COD entail similar accuracies.

Because we use complete linkage in our hierarchical clustering, the value of COD at each merge point in the dendrogram corresponds to the largest pairwise difference in algorithm behavior within the newly formed cluster. For a cluster cl, we refer to this value as $mCOD(cl)$. Since, for a particular clustering, we do not know a priori which cluster will be predicted, and indeed, different clusters are predicted for different test examples, the quantity we try to minimize is actually the maximum of the COD values over all clusters in the clustering, i.e.,

$$COD(\mathbb{C}) = \max_{cl \in \mathbb{C}} \{mCOD(cl)\}$$

On the other hand, as with any learning task, we also wish to maximize $Acc_{Meta}(\mathbb{C})$. Hence, our choice of cut point is governed by the natural tension between minimizing $COD(\mathbb{C})$ and maximizing Acc_{Meta}. It is easy to see that these two goals cannot be reached simultaneously and that a compromise must be found. Indeed, it is easy to maximize $Acc_{Meta}(\mathbb{C})$ by choosing a very high cut point, such that, as mentioned above, all algorithms are in a single cluster. In this case, $Acc_{Meta}(\mathbb{C})$ is trivially 100%. However, $COD(\mathbb{C})$ is then maximal and the metamodel is essentially useless because of the extreme diversity among algorithms in the predicted cluster. Similarly, it is easy to minimize $COD(\mathbb{C})$ by choosing a very low cut point, such that, as mentioned above, all algorithms are in singleton clusters. In this case, $COD(\mathbb{C})$ is trivially 0. However, the metamodel's accuracy will suffer due to overfitting and the low ratio of examples to target class values.

Our approach to selecting an adequate cut point through the COD-generated dendrogram takes this tension into account naturally. It is adapted from one of the standard techniques, consisting in cutting where the gap between two successive merges is largest [49]. Here, we proceed similarly, except that we set lower and upper bounds on clustering size to ensure that the selected clustering is neither too sparse nor too dense. Thus, we compute

the gaps $g_i = COD(\mathbb{C}_i) - COD(\mathbb{C}_{i-1}]$ between all successive merges, find the largest value $k = \text{argmax } g_i$ for $6 \leq i \leq 16$, and set $COD(\mathbb{C}_{k-1})$ as our cut point (note that clusterings are number from 0 to 21, from bottom to top). The corresponding clustering is then used to label the training meta-examples.

7.4 Experimental Results

In this section, we build clustering-based metamodels for algorithm selection and ranking, and compare their performances with metalearning for algorithm selection and ranking.

7.4.1 Metalearning of Algorithm Selection Model

At the metalevel, each of our 129 datasets is characterized by its values over a pre-defined set of metafeatures, as in other metalearning for classification approaches (e.g., see [2, 18, 59]). Our set of 22 metafeatures consists of a combination of statistical measures and a small set of landmarkers, including the following.

- `lgE`: log of the number of examples
- `lgREA`: log of the ratio of the number of examples to the number of attributes
- `numClasses`: number of target classes
- `numInstsPerClass`: ratio of the number of examples to the number of target classes
- `landmarkerMajorityGuesser`: majority class landmarker
- `landmarker1NN`: 1-NN landmarker
- `landmarkerNaiveBayes`: NB landmarker

We do not attempt to optimize the choice of a metalearner, but simply choose Weka's decision tree learning algorithm J48 as our metalearner. We will use the same metalearner throughout so that comparisons across metamodels are fair. Again, recall that metalearning for algorithm selection is a special case of cluster-based metalearning, where all clusters are

singletons (i.e., level 0 in our dendrogram). Furthermore, there are two level at which performance may be measured. At the metalevel, we wish to see how accurate our metalearner is at predicting the target cluster. At the base level, we wish to see how the metamodel's prediction impact accuracy on the user's tasks of interest. We propose the following, complementary metrics to compare the performance of cluster-based metalearning strategies.

- $Acc_{Meta}(\mathbb{C})$. This is accuracy of metalearning based on clustering $\mathbb{C}$, measured by 10-fold cross-validation. It is measured at the metalevel across all training meta-examples.
- $WCAcc_{Base}(\mathbb{C})$. For each test meta-example t, we measure the absolute value of the difference between the accuracy on t of the best algorithm in t's target cluster and the accuracy on t of the worst-performing algorithm in t's predicted cluster. We then average over all meta-examples. This measures the worst performance, at the base level, of clustering-based selection, i.e., when the user always picks the poorest algorithm in the predicted cluster.[3]
- $BCAcc_{Base}(\mathbb{C})$. For each test meta-example t, we measure the absolute value of the difference between the accuracy on t of the best algorithm in t's target cluster and the accuracy on t of the best-performing algorithm in t's predicted cluster. We then average over all meta-examples. This is a more optimistic measure of performance at the base level, since it assumes that the user always picks the best algorithm in the predicted cluster.
- $AVG^{+}Acc_{Base}(\mathbb{C})$. For each test meta-example t, we measure the absolute value of the difference between the accuracy on t of a randomly selected algorithm in t's target cluster and the accuracy of a randomly selected algorithm in t's predicted cluster. We repeat the random selection 100 times, average the results for t, and finally average over all meta-examples. This measures a kind of average performance, at the base level, of clustering-based selection, i.e., when the user would always pick at random.

[3]Note that this really worst-case in that even if the predicted cluster is the target cluster, there is a non-zero error accrued since we calculate the accuracy difference between the best and worst algorithms in the cluster.

- $AVG^{-}Acc_{Base}(\mathbb{C})$. For each test meta-example t, we measure the absolute value of the difference between the accuracy on t of the best algorithm in t's target cluster and the accuracy of a randomly selected algorithm in t's predicted cluster. We repeat the random selection 100 times, average the results for t, and finally average over all meta-examples. This measures a kind of pessimistic average performance of clustering-based selection, i.e., when the user picks at random in the predicted cluster but is gauged against the best algorithm.
- $AVG^{-}COD_{Base}(\mathbb{C})$. For each test meta-example t, we measure the COD value between the best algorithm in t's target cluster and a randomly selected algorithm in t's predicted cluster. We repeat the random selection 100 times, average the results for t, and finally average over all meta-examples. This is similar to the previous metric but in behavior space rather than accuracy space. These two metric together provide a nice assessment of the overall relative effectiveness of clustering metamodels.

Starting from the dendrogram of Figure 7.1, and using our cut point selection method, we obtain a clustering of 10 clusters as shown in Figure 7.3 (cut point: 0.178). Table 7.2 shows the performance of the corresponding cluster-based metalearner ($|\mathbb{C}| = 10$) compared to the performance of a standard algorithm selection metamodel ($|\mathbb{C}| = 21$), with respect to our metrics. Standard deviations are given in parentheses and the column Wins record the percentage of times the clustering approach outperforms the algorithm selection approach on the 100 random trials.

Table 7.2: Clustering vs. Algorithm Selection

Metric	$\|\mathbb{C}\| = 10$	$\|\mathbb{C}\| = 21$	**Wins**
$Acc_{Meta}(\mathbb{C})$	32.62%	22.30%	
$WCAcc_{Base}(\mathbb{C})$	7.66	3.47	
$BCAcc_{Base}(\mathbb{C})$	2.38	3.47	
$AVG^{+}Acc_{Base}(\mathbb{C})$	3.09 (1.95)	3.47	70%
$AVG^{-}Acc_{Base}(\mathbb{C})$	4.44 (2.39)	3.47	40%
$AVG^{-}COD_{Base}(\mathbb{C})$	0.14 (0.10)	0.13	50%

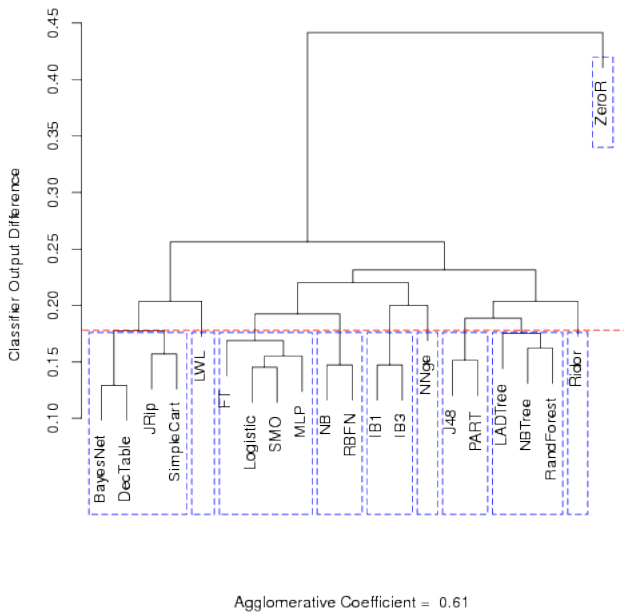


Figure 7.3: Selected Clustering for Metalearning. With a cut point of 0.178, 10 clusters are obtained.

The results on $Acc_{Meta}(\mathbb{C})$ show that even though the cluster-selection is somewhat superior, both metamodels suffer from low accuracy. While the poor performance of the algorithm selection metamodel may be explained in part by the small ratio of training meta-examples to target values (129/21=6) and possible overfitting, this is less of an issue for the cluster-selection metamodel. We suspect that the problem lies mostly in the metafeatures. While we used a reasonably broad range of well-known metafeatures, the accuracy of the metamodel suggests that they may not have the predictive power needed in this context. As pointed out by Rice, "the determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem" [63]. Since our main goal was to compare the relative performance of cluster selection and algorithm selection rather than to derive a good metamodel, our comparison is at least fair. However, more work is needed if we are to operationalize any of our metamodels.

Results over our other metrics suggest that cluster-selection performs similarly to algorithm selection with a slight disadvantage in $WCAcc_{Base}(\mathbb{C})$ and a slight advantage in $BCAcc_{Base}(\mathbb{C})$ and $AVG^{+}Acc_{Base}(\mathbb{C})$. That is, on average, cluster selection is expected to produce slightly better results at the base level. In fact, it will likely do so about 70% of the time. For practitioners who worry about the worst cases, the result on $WCAcc_{Base}(\mathbb{C})$ can be problematic, however. It is desirable that users of our clustering-based metamodel be given additional guidance to avoid the worst selection. We will consider one way to do using ranking in the next section.

Before we do that, we wish to point out that our description and implementation of, as well as experiments with, clustering-based metalearning has relied on COD data collected indiscriminately over our set of 129 datasets. It is well-known, however, that a number of rather simple factors, such as attribute types, skewness, and missing data, have a significant impact on the behavior of many learning algorithms. For example, it is well accepted that neural network learning is more adapted to continuous inputs and decision tree learning to discrete inputs. As a result, it is likely that different, and likely improved, clustering, and hence metamodels, would be obtained if datasets were to be split along any of these dimensions. We clearly do not have space to, nor can we realistically, consider all possibilities. However, for the sake of illustration, and because it is such a simple test to run on any dataset, we consider splitting our 129 datasets on the basis of the attribute types they contain: continuous-only vs. discrete-only. The same approach can be applied along any other such a priori discrimination criterion.

The corresponding dendrograms, together with the selected clusterings are on Figures 7.4 (xut point: 0.173, 10 clusters) and 7.5 (cut point: 0.093, 15 clusters), respectively. Tables 7.3 and 7.4 show the performance of the corresponding cluster-based metalearners compared to the performance of a standard algorithm selection metamodel, with respect to our metrics.

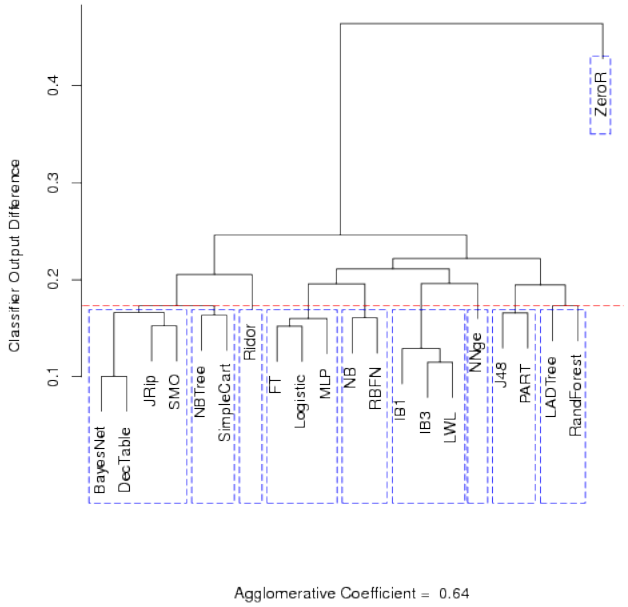


Figure 7.4: Selected Clustering for Metalearning on Continuous-only Datasets With a cut point of 0.173, 10 clusters are obtained.

Table 7.3: Clustering vs. Algorithm Selection: Continuous-only Datasets

Metric	$\|\mathbb{C}\| = 10$	$\|\mathbb{C}\| = 21$	**Wins**
$Acc_{Meta}(\mathbb{C})$	33.75%	15.97%	
$WCAcc_{Base}(\mathbb{C})$	5.50	3.14	
$BCAcc_{Base}(\mathbb{C})$	1.72	3.14	
$AVG^{+}Acc_{Base}(\mathbb{C})$	3.04 (2.42)	3.14	50%
$AVG^{-}Acc_{Base}(\mathbb{C})$	4.02 (2.65)	3.14	50%
$AVG^{-}COD_{Base}(\mathbb{C})$	0.16 (0.15)	0.15	56%

These results show some improvement in the quality of the clustering-based metamodels, most markedly in the discrete-only case, where the value of the worst-case metric $WCAcc_{Base}(\mathbb{C})$ is almost the same for cluster-selection and algorithm selection. Furthermore, the clustering-based metamodel outperforms the algorithm selection metamodel on $BCAcc_{Base}(\mathbb{C})$ as well as on all three of our other base-level metrics at least 67% of the time. Note that due to the nature of the datasets in each case, we are able to extend the set of metafeatures to include type-specific metafeatures (e.g., skewness and kurtosis for

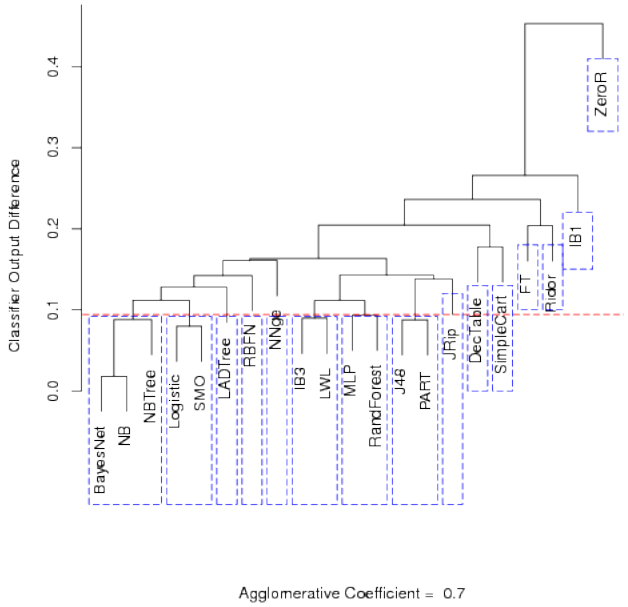


Figure 7.5: Selected Clustering for Metalearning on Discrete-only Datasets With a cut point of 0.093, 15 clusters are obtained.

Table 7.4: Clustering vs. Algorithm selection: Discrete-only Datasets

Metric	$\|\mathbb{C}\| = 15$	$\|\mathbb{C}\| = 21$	**Wins**
$Acc_{Meta}(\mathbb{C})$	28.33%	26.66%	
$WCAcc_{Base}(\mathbb{C})$	6.60	6.47	
$BCAcc_{Base}(\mathbb{C})$	4.08	6.47	
$AVG^{+}Acc_{Base}(\mathbb{C})$	4.26 (4.38)	6.47	80%
$AVG^{-}Acc_{Base}(\mathbb{C})$	4.59 (4.47)	6.47	80%
$AVG^{-}COD_{Base}(\mathbb{C})$	0.08 (0.07)	0.10	67%

continuous-only datasets, and meanAttributeChiSquare and averageValueperAttribute for discrete-only attributes).

7.4.2 Metalearning of Algorithm Ranking Model

Up till now, our proposed metalearner returns a cluster of learning algorithms and leaves it up to the user to select an algorithm within that cluster. As we have discussed, this means that performance at the base level may be sub-optimal if the wrong algorithm is chosen (especially when the clusters are large and/or exhibit poor intra-cluster similarity).

Furthermore, as has been argued in the context of algorithm selection, there are significant advantages to returning not a prediction of best 1-of-N, but rather a ranking of all available algorithms so that if the predicted best fails to give satisfaction, the user may try the next best one and so forth (e.g., see [12, 18]). We show how the ranking idea may be beneficially applied in the cluster selection context.

The construction of a ranking model bears resemblance with the notion of collaborative filtering. Indeed, the ranking of learning algorithm l on query dataset or learning task t is determined by the combination of rankings of l on t's neighbors. One simple way to accomplish this, as used for algorithm selection in [18], is to first find neighbors using k-nearest-neighbor (k-NN) and then compute average rankings over the k neighbors. Figure 7.6 shows an example of finding the ranking of J48 on some query learning task t_q where $k = 2$. The ranking of J48 on t_q is obtained its rankings on t_3 and t_4. If a simple, unweighted average is used, the ranking of J48 on t_q is $(3 + 5)/2 = 4$.

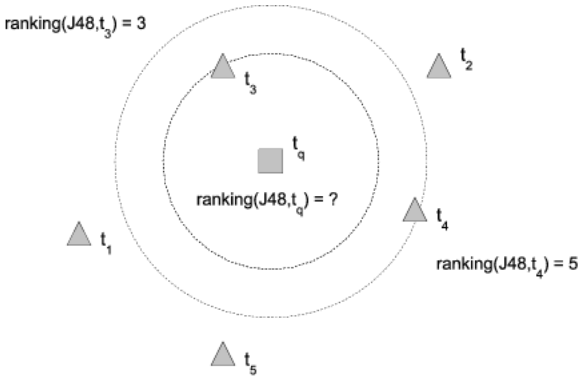


Figure 7.6: Finding the Ranking of J48 with 2 Nearest Neighbors

In the case of algorithm ranking, if there are N algorithms under study, all of them must be ranked. Since we are predicting clusters, we only need to rank the M algorithms found in the cluster, where in all cases except the degenerate one, $M << N$. The procedure to estimate rankings is the same, however. We use k-NN and average. The computed rankings can then be compared with the true rankings, using correlation measures. The two most common rank correlation measures are Spearman's rank correlation coefficient and Pearsons'

Table 7.5: Clustering (CR_k) vs. Algorithm Ranking (AR_k)

k	**1**		**2**		**3**	
	CRk	ARk	CRk	ARk	CRk	ARk
All	0.21	0.00	0.31	0.16	0.34	0.34
Cont.	0.51	0.13	0.54	0.25	0.53	0.19
Disc.	0.76	0.33	0.95	0.47	0.75	0.22

rank correlation coefficient. We will prefer Pearson's rank correlation coefficient here since it allows ties while Spearman's rank correlation does not. Pearson's rank correlation ranges between -1 and +1, where +1 means that the two rankings are identical, -1 means that the two rankings are the reverse of each other, and 0 means that there is no relation between the two rankings.

To examine the effectiveness of the clustering metamodel, we again compare it against an algorithm ranking model. Since the algorithm ranking metamodel considers all 21 algorithms, it estimates the ranking of all algorithms using k-NN. To make it comparable to the "sub-ranking" performed by the cluster-based model, it then picks only the top M algorithms (where M is the size of the predicted cluster) and computes the correlation between this estimated sub-ranking and the true ranking.

We run k-NN with k=1, 2, and 3. For each k, we compare the Pearson rank correlation coefficient of both metamodels. The results are in Table 7.5 for all three clusterings considered in the previous section.

Across all underlying data types and all selected values of k, the correlation coefficient of the clustering-based metamodel is larger than or equal to that of the algorithm ranking metamodel. Note how the effect is significantly increased when the datasets are split based on the nature of the attributes, with correlation values above 0.5, and reaching almost perfect agreement with the true ranking for discrete datasets and $k = 2$. These results suggest that, not only is the computation more efficient (remember that $M < N$), but the ranking of algorithms in the cluster is also closer to the true ranking than the one estimated by the complete ranking model.

7.5 Conclusion

In this paper, we leverage results on behavior-based clustering of learning algorithms to propose novel algorithm selection and ranking metamodels based on the prediction of clusters. The basic idea is for the metalearner to induce a model that matches a dataset to the cluster of learning algorithms that behave most similarly with the best algorithm for that dataset. At first glance, this approach may appear to be less effective than single algorithm selection since multiple algorithms are recommended to the user. However, clustering-based metalearning has a number of advantages over metalearning for single algorithm selection (and/or ranking).

- By clustering algorithms in terms of instance-level behavior, the metalearning process is simplified (i.e., higher training examples to target values ratio) and the tendency of overfitting that plagues single algorithm selection is decreased. Our results, both with selection and ranking, show that the cluster-based model is at least as good as the algorithm-based model, and often better.
- As with algorithm selection, the weakness of clustering selection with respect to the difficulty of finding the best algorithm in a predicted cluster can be compensated for by ranking algorithms in the cluster. Our results suggest that a clustering selection model with ranking is typically more accurate than the traditional algorithm ranking model.
- The clustering-based approach lends itself naturally to the building of specialized metamodels, based on simple a priori criteria about dataset characteristics. We have illustrated this idea using the types of the attributes (continuous vs. discrete) to construct improved metamodels. Other such criteria may similarly be used. No new computation is required, only restricting the global COD distance matrix to the subset of datasets that satisfy the selected criteria.

- The clustering-based approach challenges the Procustean approach to algorithm selection typically adopted, where one first selects a model class and then an algorithm within that class [79]. Indeed, in that approach, the model classes and the algorithms are fixed, generally as part of a taxonomy or ontology (e.g., see [36]). Here, however, the classes are built from clustering based on behavior similarity rather than expert-driven decisions. The clustering shown here highlights at least one significant discrepancy between automatic clustering and human-designed ontology. Indeed, whereas NB and RBFN are in different classes in Weka's underlying taxomomy (i.e., one is probability-based and the other function-based) as well as in the data mining ontology (i.e., one is generative and the other is discriminative), they actually merge together first in our COD-based clustering. This particular finding is the subject of a separate study [46].
- When new algorithms are introduced, it will not generally be necessary to rebuild the metamodel from scratch, as is the case in algorithm selection. Indeed, the addition of a new algorithm in the algorithm selection context corresponds to the addition of a target value to the metalearning task. This, of course, changes the nature of the task, which must be re-learned in light of the new information. Within the clustering-based context, one simply needs to find what cluster the new algorithm belongs to and to add it to said cluster. Only when a new algorithm is sufficiently different from all other existing ones will a new cluster (and hence a new target value) need to be created for it. In this regard, the clustering-based approach supports the possibility of incremental metalearning. We have yet to experiment with this.

Despite our promising results, there is room for improvement and future research. In particular, the accuracy of our metamodels is rather low, suggesting that the metafeatures we employ are not sufficiently predictive for the metalearning task at hand. Discovering effective metafeatures would be a key factor in improving the usefulness and operationalization of our metamodels. When constructing our dendrograms, we have only considered the default parameter settings of all learning algorithms. Work is needed to better understand

and quantify the impact that certain parameter settings have on learning algorithms, and whether/how they impact our clusterings. Finally, on the more practical side, our selection metamodel suffers from the fact that if the actual best algorithm is not in the predicted cluster, all others in that cluster will also perform poorly since they are similar to each other, and the user is left with no recourse, being stuck with only poor choices to select from. One possibility that may be investigated consists in two-level ranking system, wherein the system first rank clusters, and then ranks algorithms within the clusters (as proposed here). With such a ranking, the user could "escape" from a bad cluster and try the next best one.

Chapter 8

Conclusion

Metalearning seeks to increase human understanding of the characteristics of learning tasks, algorithms, and the relationship between them. By doing this, we make the best use of current algorithms and are able to design new algorithms that compensate for the weaknesses of existing ones. The results presented here are by no means complete, but are incremental to the ultimate goal of metalearning, which is to obtain the complete knowledge of the behavior of algorithms and their relationship to learning tasks.

Some of the important factors for the success of metalearning are the size of base-level tasks and the quality of metafeatures. While a feature of an individual dataset represents a characteristic of corresponding specific learning task, a metafeature on multiple datasets represents a general characteristic of associated learning tasks. An important question is how to obtain useful metafeatures given a particular metalearning task. There is no systematic approach to discover good metafeatures, rather they are found based on the researcher's insight, creativity, and/or prior knowledge of the metalearning task at hand.

Our experiments used about a hundred datasets, which is small compared to all learning tasks, but a decent size considering other related research. As for metafeatures, we found that certain ones (e.g., mean attribute entropy and hardness) are related to the performance of algorithms. These metafeatures allow us to estimate the difficulty of learning tasks. They are especially useful and efficient with large volumes of data, such as DNA sequence, business transaction, and medical data. Applying algorithms on these data can be expensive with limited resources.

Hierarchical clustering with the COD metric identified algorithms that behaved similarly. Some of them are NB and RBFN. NB is a probability model, based on Bayes' theorem but RBFN is a type of neural networks. Their similarity in behavior turned out to be largely due to their structural similarity which is affected by parameter choices and implementation. From this analysis, we learned that 1) Weka RBFN usually performs at least as well as typical RBFN (i.e., RBFN where Gaussian kernels are shared across classes) and is similar to NB, 2) even though their predictive behaviors are quite similar, NB has a significant advantage in execution time. 3) The similarity of NB and RBFN provides a counter-example of the current European data mining ontology project [36].

As an alternative solution to the algorithm selection problem, we propose a new metamodel that returns a cluster of best-fit algorithms for the given task. This metamodel has several advantages over current algorithm selection models. It simplifies metalearning tasks and reduces the risk of overfitting to training data. Furthermore, when a new learning algorithm is added to the pool of candidate algorithms, we can reuse the previous metamodel by finding the cluster to which the new algorithm belongs. This approach is computationally more efficient than rebuilding a metamodel from scratch.

The work done in this dissertation represents a valuable contribution, and yet there are many exciting and important metalearning works. The following are some of the important metalearning tasks that remain.

1. Analysis of the sensitivity of algorithms to various parameter settings

   In this dissertation, we covered the behavior of learning algorithms with default parameters. This provides a starting point to the understanding of algorithms with various parameters. In practice, the parameters of the target learning algorithm are often changed in order to obtain better predictive performance. However, little is known of the amount of impact of parameters on the performance of learning algorithms.

2. Discovery of more effective and efficient data characteristics

The solution of useful data characteristics or metafeatures is a key to constructing an effective metamodel. Until now, most data characterization methods hinge on statistics, information theory, and landmarking. Even though some of them are effective on certain metalearning tasks, we are still lacking in effective and efficient metafeatures.

3. Construction of the metamodel that predicts the entire learning process, instead of just predicting the best algorithm for the given task.

   With our selection metamodel, we focused on a portion of the entire learning process. Some other important pieces of the process are data pre-processing and feature selection. Since difficult datasets often have highly-dimensional features and contain noisy and missing entries, there are various ways of handling them. This know-how can contribute to build the optimal metamodel.

4. Search for the optimal algorithm that induces the best metamodel for the given metalearning task.

   The choice of an algorithm that induces the metamodel for selection is usually undisciplined. Together with the choice of metafeatures, good selection of the algorithm may have a profound impact on the result of the associated metalearning task.

References

[1] K. Ali and M. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24(3):173–202, 1996.

[2] S. Ali and K.A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

[3] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[4] A. Asuncion and D.J. Newman. UCI machine learning repository [http://archive.ics.uci.edu/ml/]. University of California, Irvine, School of Information and Computer Sciences, 2007.

[5] K. Bailey. *Methods of social research,fourth edition*. Simon and Schuster, 2007.

[6] H. Bensusan. God doesn't always shave with occam's razor - learning when and how to prune. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 119–124, 1998.

[7] H. Bensusan and C. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (LNAI 1910)*, pages 325–330, 2000.

[8] H. Bensusan and C. Giraud-Carrier. Harmonia loosely praestabilita: Discovering adequate inductive strategies. In *Proceedings of the Twenty-second Annual Meeting of the Cognitive Science Society*, pages 609–614, 2000.

[9] H. Bensusan, C. Giraud-Carrier, and C. Kennedy. A higher-order approach to meta-learning. In *Proceedings of the ECML-2000 Workshop on Meta-learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 109–118, 2000.

[10] H. Bensusan and A. Kalousis. Estimating the predictive accuracy of a classifier. In *Proceedings of the Twelfth European Conference on Machine Learning*, volume 2167 of *LNCS*, pages 25–36, 2001.

[11] A. Bernstein, F. Provost, and S. Hill. Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, 2005.

[12] H. Berrer, I. Paterson, and J. Keller. Evaluation of machine-learning algorithm ranking advisors. In *Proceedings of the PKDD-2000 Workshop on Data-Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, 2000.

[13] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[14] H. Blockeel and J. Vanschoren. Experiment databases: Towards an improved experimental methodology in machine learning. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (LNCS 4702)*, pages 6–17, 2007.

[15] P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the Seventh European Conference on Machine Learning*, pages 83–102, 1994.

[16] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer, 2008.

[17] P. BRAZDIL and R. HENERY. Analysis of results. In D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors, *Machine Learning, Neural and Statistical Classification*, chapter 10. Ellis Horwood, New York, 1994.

[18] P. Brazdil, C. Soares, and J. Pinto da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[19] C.E. Brodley and T Lane. Creating and exploiting coverage and diversity. In *AAAI-96 Workshop Integrating Multiple Learned Models*, pages 8–14, 1996.

[20] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[21] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.

[22] Y-S Chung, D.F. Hsu, and C.Y. Tang. On the relationships among various diversity measures in multiple classifier systems. In *International Symposium on Parallel Architectures, Algorithms and Networks*, pages 184–190, 2008.

[23] S. Craw, D. Sleeman, N. Granger, M. Rissakis, and S. Sharma. Consultant: Providing advice for the machine learning toolbox. In *Research and Development in Expert Systems IX (Proceedings of Expert Systems'92)*, pages 5–23. SGES Publications, 1992.

[24] T.G.. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[25] C. Elkan. Boosting and naive bayesian learning. Technical Report CS97-557, Department of Computer Science and Engineering, University of California, San Diego, 1997.

[26] R. Engels and C. Theusinger. Using a data metric for offering preprocessing advice in data-mining applications. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, pages 430–434, 1998.

[27] J. Fleiss. *Statistical Methods for Rates and Proportions*. John Wiley & Sons, 1981.

[28] J. Fuernkranz and J. Petrak. An evaluation of landmarking variants. In *Proceedings of the ECML/PKDD-01 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-learning*, pages 57–68, 2001.

[29] J. Gama and P. Brazdil. Characterization of classification algorithms. In *Proceedings of the 7th Portugese Conference in AI*, pages 83–102, 1995.

[30] J. Gama and P. Brazdil. Characterization of classification algorithms. In *Proceedings of the Seventh Portuguese Conference on Artificial Intelligence*, pages 189–200, 1995.

[31] E Gatnar. A diversity measure for tree-based classifier ensembles. In *Data Analysis and Decision Support*, pages 30–38, 2005.

[32] G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification processes. *Journal of Image Vision and Computing*, 19:600–707, 2001.

[33] C. Giraud-Carrier. The data mining advisor: Meta-learning at the service of practitioners. In *Proceedings of the Fourth International Conference on Machine Learning Applications*, pages 113–119, 2005.

[34] S.B. Guerra, R.B.C. Prudêncio, and T.B. Ludermir. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In *Proceedings of the Eighteenth International Conference on Artificial Neural Networks, LNCS 5163*, pages 523–532, 2008.

[35] S.S. Haykin. *Neural Networks and Learning Machines, Third Edition*. Pearson Education, Inc., 2009.

[36] M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. A data mining ontology for algorithm selection and meta-mining. In *Proceedings of the ECML/PKDD Workshop on Third Generation Data Mining: Towards Service-Oriented Knowledge Discovery*, pages 76–87, 2009.

[37] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

[38] A. Kalousis, J. Gama, and M. Hilario. On data and algorithms: Understanding inductive performance. *Machine Learning*, 54(3):275–312, 2004.

[39] A. Kalousis, J. Gama, and M. Hilario. On data and algorithms: Understanding learning performances. *Machine Learning*, 54(3):275–312, 2004.

[40] Y. Kodratoff, D. Sleeman, M. Uszynski, K. Causse, and S. Craw. Building a machine learning toolbox. In L. Steels and B. Lepape, editors, *Enhancing the Knowledge Engineering Process*, pages 81–108. Elsevier Science Publishers, 1992.

[41] C. K´opf, C.C. Taylor, and J. Keller. Meta-analysis: From data characterization for meta-learning to meta-regression. In *Proceedings of the PKDD Workshop on Data-Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, 2000.

[42] L.I. Kuncheva and C.J. Whitaker. Ten measures of diversity in classifier ensembles: Limits for two classifiers. In *Proceedings of the IEE Workshop on Intelligent Sensor Processing*, pages 1–10, 2001.

[43] L.I. Kuncheva and C.J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[44] J. Lee and C. Giraud-Carrier. New insights into learning algorithms and datasets. In *Proceedings of the Seventh International Conference on Machine Learning and Applications*, pages 135–140, 2008.

[45] J. Lee and C. Giraud-Carrier. A comparison of naive bayes and radial basis function networks. Submitted, 2010.

[46] J. Lee and C. Giraud-Carrier. Clustering-based metalearning for algorithm selection. Submitted, 2011.

[47] J. Lee and C. Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6), 2011.

[48] G. Lindner and R. Studer. Ast: Support for algorithm selection with a cbr approach. In *Proceedings of the Third European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 418–423, 1999.

[49] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[50] METAL: A meta-learning assistant for providing user support in machine learning and data mining. ESPRIT Framework IV LTR Reactive Project Nr. 26.357, 1998-2001.

[51] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[52] Tom Mitchell. Generative and discriminative classifiers:naive bayes and logistic regression. Online, 2005. draft of September 2006.

[53] A.W. Moore and M.S. Lee. Efficient algorithms for minimizing cross validation error. *Proceeding of the 11th International Conference on Machine Learning*, 1994. Morgan Kaufmann.

[54] A. Narasimhamurthy. Evaluation of diversity measures for binary classifier ensembles. In *Proceedings of the Sixth International Workshop on Multiple Classifier Systems (LNCS 3541)*, pages 267–277, 2005.

[55] A.Y. Ng and M.I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Neural Information Processing Systems*, pages 841–848, 2001.

[56] Y. Peng, P.A. Flach, P. Brazdil, and C. Soares. Improved data set characterisation for meta-learning. In *Proceedings of the Fifth International Conference on Discovery Science*, pages 141–152, 2002.

[57] C. Perlich, F. Provost, and J.S. Simonoff. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4(2):211–255, 2003.

[58] A.H. Peterson and T.R. Martinez. Estimating the potential for combining learning models. In *Proceedings of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.

[59] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 743–750, 2000.

[60] R.B.C. Prudêncio and T.B. Ludermir. Active selection of training examples for meta-learning. In *Proceedings of the Seventh International Conference on Hybrid Intelligent Systems*, pages 126–131, 2007.

[61] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[62] L. Rendell, R. Seshu, and D. Tcheng. Layered concept-learning and dynamically-variable bias management. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 308–314, 1987.

[63] J.R Rice. The algorithm selection problem. In *Advances in Computers*, pages 65–118, 1976.

[64] Y.D. Rubinstein and T. Hastie. Discriminative vs informative learning. In *Proceedings of the American Association Artificial Intelligence*, 1997.

[65] W.S. Sarle. Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS User Group International Conference*, 1994.

[66] C. Schaffer. A conservation law for generalization performance. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 259–265, 1994.

[67] M Sirkin. *Statistics for the social sciences*. SAGE, 2006.

[68] David B. Skalak. The sources of increased accuracy for two proposed boosting algorithms. In *In Proc. American Association for Arti Intelligence, AAAI-96, Integrating Multiple Learned Models Workshop*, pages 120–125, 1996.

[69] K.A. Smith, F. Woo, V. Ciesielski, and R. Ibrahim. Matching data mining algorithm suitability to data characteristics using a self-organizing map. In *Proceedings of the First International Workshop on Hybrid Intelligent Systems*, pages 169–179, 2001.

[70] K.A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 2008.

[71] K.A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 2008.

[72] C. Soares. UCI++: Improved support for algorithm selection using datasetoids. In *Proceedings of the Thirteenth Pacific-Asia Knowledge Discovery and Data Mining Conference*, pages 499–506, 2009.

[73] S.Y. Sohn. Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1137–1144, 1999.

[74] S.Y. Sohn. Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1137–1144, 1999.

[75] G. Stiglic and P. Kokol. GEMLeR: Gene expression machine learning repository [http://gemler.fzv.uni-mb.si/]. University of Maribor, Faculty of Health Sciences, 2009.

[76] E.K. Tang, P.N. Suganthan, and X. Yao. An analysis of diversity measures. *Machine Learning*, 65(1):247–271, 2006.

[77] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*, chapter 5. Morgan Kaufmann Publishers. Inc., 1986.

[78] P.M. Vaidya. An o(n log n) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(2):101–115, 1989.

[79] M. van Someren. Model class selection and construction: Beyond the procustean approach to machine learning applications. In G. Paliouras, V. Karkaletsis, and C. Spyropoulos, editors, *Machine Learning and Its Applications: Advanced Lectures (LNCS 2049)*, pages 196–217. Springer-Verlag, 2001.

[80] R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares. Using meta-learning to support data-mining. *International Journal of Computer Science Applications*, I(1):31–45, 2004.

[81] X. Wang, K. Smith, and R. Hyndman. Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery*, 13(3):335–364, 2006.

[82] S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, pages 274–285, 2005.

[83] I.H. Witten and F. Eibe. *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann, San Francisco, CA, 2000.

[84] D.H. Wolpert. Any two learning algorithms are (almost) exactly identical. Technical report, NASA Ames Research Center, 2001.

[85] D.H. Wolpert. The supervised learning no-free-lunch theorems. In *Proceedings of the Sixth On-line World Conference on Soft Computing in Industrial Applications*, pages 325–330, 2001.

[86] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[87] G. Yule. On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London, Ser. A*, 194:257–319, 1900.