2015-04-01

Using Instance-Level Meta-Information to Facilitate a More Principled Approach to Machine Learning

Michael Reed Smith
Brigham Young University - Provo

Using Instance-Level Meta-Information to Facilitate a More Principled Approach to Machine Learning

Michael Reed Smith

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

Doctor of Philosophy

Tony Martinez, Chair
Christophe Giraud-Carrier
Dan Ventura
Quinn Snell
Daniel Zappala

Department of Computer Science

Brigham Young University

April 2015

ABSTRACT

Using Instance-Level Meta-Information to Facilitate a More
Principled Approach to Machine Learning

Michael Reed Smith
Department of Computer Science, BYU
Doctor of Philosophy

As the capability for capturing and storing data increases and becomes more ubiquitous, an increasing number of organizations are looking to use machine learning techniques as a means of understanding and leveraging their data. However, the success of applying machine learning techniques depends on which learning algorithm is selected, the hyperparameters that are provided to the selected learning algorithm, and the data that is supplied to the learning algorithm. Even among machine learning experts, selecting an appropriate learning algorithm, setting its associated hyperparameters, and preprocessing the data can be a challenging task and is generally left to the expertise of an experienced practitioner, intuition, trial and error, or another heuristic approach. This dissertation proposes a more principled approach to understand how the learning algorithm, hyperparameters, and data interact with each other to facilitate a data-driven approach for applying machine learning techniques. Specifically, this dissertation examines the properties of the training data and proposes techniques to integrate this information into the learning process and for preprocessing the training set. It also proposes techniques and tools to address selecting a learning algorithm and setting its hyperparameters.

This dissertation is comprised of a collection of papers that address understanding the data used in machine learning and the relationship between the data, the performance of a learning algorithm, and the learning algorithms associated hyperparameter settings. Contributions of this dissertation include:

- *Instance hardness* that examines how difficult an instance is to classify correctly.
- The *hardness measures* that characterize properties of why an instance may be misclassified.
- Several techniques for integrating instance hardness into the learning process. These techniques demonstrate the importance of considering each instance individually rather than doing a global optimization which considers all instances equally.
- Large-scale examinations of the investigated techniques including a large numbers of examined data sets and learning algorithms. This provides more robust results that are less likely to be affected by noise.
- The *Machine Learning Results Repository*, a repository for storing the results from machine learning experiments at the instance level (the prediction for each instance is stored). This allows many data set-level measures to be calculated such as accuracy,

precision, or recall. These results can be used to better understand the interaction between the data, learning algorithms, and associated hyperparameters. Further, the repository is designed to be a tool for the community where data can be downloaded and uploaded to follow the development of machine learning algorithms and applications.

Keywords: machine learning, supervised learning, classification, meta-learning, instance hardness, machine learning results repository

ACKNOWLEDGMENTS

I am indebted to many people who played a significant role in the completion of this dissertation. First and foremost, I express my gratitude to my Heavenly Father for His tender mercies in granting me light and knowledge in pursuing this research. I am grateful to have had the opportunity of conducting research while living in relative comfort. I recognize that this is not a blessing that everyone receives.

I thank my advisor, Dr. Tony Martinez, who took a chance on me and funded me through out my graduate career. I am grateful for the many lessons in machine learning, doing good research, writing, teaching, and life in general. I also thank Dr. Christophe Giraud-Carrier who co-authored several papers and also provided counsel in regards to my research and the way academia works. I am also grateful for the many other faculty members who take time to work with me and discuss various topics, specifically, Dr. Dan Ventura and Dr. Mark Clement.

Several portions of this dissertation would not have been completed were it not for the support and efforts from several lab-mates and undergraduate students. In this context, I thank (the now) Dr. Michael Gashler, Logan Mitchell, Andrew White, Daniel Saunders, and Michael Walker.

I would be remiss if I did not thank my family, particularly my wife, Katie, who patiently waited for me to finish my schooling while living on a college student's income. I am grateful for her support and encouragement. I thank my children, Jack, Max, Evelyn, and Lucy – for the joy that they bring into my life, for accepting me with all of my flaws, and motivating me to press forward.

Finally, I recognize that I am standing on shoulders of many researchers who have gone before me. Without their work, I would not have been able to accomplish my research.

Table of Contents

List of Figures

List of Tables

Part I

Background and Motivation

Part I provides the background information and motivation for taking a more principled approach to machine learning, the topic of this dissertation.

Chapter 1 provides a high-level overview of machine learning, provides the context for this dissertation, and describes the motivation for this dissertation. Specifically, it presents the dependencies of the induced hypothesis on the learning algorithm, its associated hyper-parameters, and the training data. Meta-learning is then introduced including the difficulties encountered in meta-learning. Finally, the remainder of the dissertation is outlined.

Most machine learning results and data complexity studies have been conducted at the data set-level. Chapter 3 establishes *instance hardness* and the *hardness measures* as a means of measuring and characterizing instance complexity. Chapter 3 was published under the following reference.

Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. "An Instance Level Analysis of Data Complexity", *Machine Learning*, 95(2): 225–256, 2014.

Part II explores ways to incorporate the information from instance hardness into the learning process. Part III concludes this dissertation and provides directions for future research.

Chapter 1

Introduction

Machine learning is the science of facilitating computers to learn from past experience and includes a set of learning algorithms to accomplish this task. Machine learning algorithms allow computers to act without being explicitly programmed for a specific task and allow computers to change their behaviors overtime. As such, machine learning algorithms are used for complex tasks that are difficult to, or cannot be, explicitly programmed or modeled.

Machine learning algorithms are becoming more and more ubiquitous in many domains and applications. For example, machine learning techniques are being used for individualized spam filters that update the filters as spammers modify their spamming tactics [2]. They are used to recommend items to customers on sites such as Amazon.com [5] or Netflix [7]. As more and more data is collected, making it less feasible for humans to analyze the data, machine learning will be used in an increasing number of domains and further integrated into everyday tasks. However, the success of the application of machine learning techniques depends on which machine learning algorithm is being used, its associated hyperparameters, and the data that is being used for training. Currently, learning algorithm selection, hyperparameter selection, and data preprocessing are generally done using a heuristic approach such as trial and error or using the expertise of a machine learning practitioner. This dissertation proposes a more *principled* approach to machine learning, where principled refers to first understanding the relationship between the learning algorithms, their hyperparameters, and the training data (the principles of machine learning) and then adhering to these principles in the application of machine learning. This dissertation is not the first attempt at more

principled machine learning. This dissertation, however, is unique in that the examination of the principles of machine learning begins at the instance-level. This dissertation shows that understanding the data better and incorporating this knowledge into the learning process significantly improves the performance of machine learning algorithms.

1.1 Supervised learning

Machine learning algorithms can use either *supervised* or *unsupervised* learning. In *supervised machine learning*, a learning algorithm is presented with a set of training examples or instances consisting of a pair of vectors X and Y where X is the set of features representing an object or instance and Y is the target output value. Given a set of training instances $< X, Y >$, the goal of supervised machine learning algorithms is to induce a *model* or *hypothesis* h from the space of possible hypotheses H such that $h : X \rightarrow Y$. Depending on the chosen learning algorithm, the hypothesis h can be inferred in a number of ways such as generating a set of rules, calculating probabilities, or training the weights in a neural network. This dissertation focuses on supervised learning algorithms for classification (as opposed to regression).

To determine if a hypothesis h from H is better than another, some scoring measure is also needed. Common scoring measures are accuracy, precision, and correlation. A common problem faced in machine learning is overfit, where a model will learn the nuances of the data rather than learn the underlying concepts of the task. One example of overfitting the data is when a learning algorithm memorizes the training data. This is observed when a model achieves a high score when evaluated on the data used for training, but produces low scores when it is used on novel data. Thus, a set of test instances is also provided or is partitioned from the training set for scoring. Ideally, the set of test instances is representative of the task that is being modeled. With this assumption, the models are evaluated on the test instances rather than on the training data. Because of overfit, there is a trade-off between the complexity of the model and a model's score (known as regularization [1]). As the model

becomes more complex, there is a greater risk of overfitting the data. Simpler models are often better able to generalize on novel data instances.

The free parameters available for inducing a hypothesis include the learning algorithm g, the training data T that g is trained on, and the learning algorithm's associated hyperparameters λ. The hyperparameters refer to the user-set parameters available to a learning algorithm such as the number of hidden nodes in a multilayer-perceptron or the kernel function for a support vector machine. Thus, an induced hypothesis h is the result of training g on T with hyperparameters λ:

$$h = g(T, \lambda).$$

This dissertation primarily examines understanding the individual training instances and then incorporating this information into the learning process. Selecting the learning algorithm and setting its hyperparameters are also examined.

1.2 Noisy, Outlier, and Detrimental Instances

Real-world data sets are often noisy. The work in this dissertation seeks to identify and properly handle instances that result in lower performing induced models. However, the notion of noisy instances and related notions of outlier and detrimental instances are not formally defined. Noisy instances and outliers are similar concepts and are often used interchangeably. Noise is introduced into the data by a system or process that corrupts the data in some way. For example, noise can be introduced from hardware failures, faulty measurement readings, typos, or precision errors. Therefore, noisy instances are those instances that have noise in their measured features. Outliers are instances where the observed values are an abnormal distance from the other observed values. For example, in statistics an abnormal distance is often defined as being more than two or three standard deviations from the mean. Outliers can occur for various reasons such as noise in the data or the instance is valid but an ex-

ception to the rule and is infrequently observed. To determine if an instance is noisy or an exception, generally a domain expert is required.

One of the purposes of this dissertation is to identify and characterize instances that are detrimental. Detrimental instances are those instances that result in a lower quality induced model if they are used for training. The quality of the induced model h is characterized by its empirical error for a specified error function $\mathcal{E}$ on a test set V:

$$E(h, V) = \frac{1}{|V|} \sum_{\langle x_i, y_i \rangle \in V} \mathcal{E}(h(x_i), y_i)$$

where V can be T or a disjoint set of instances. In k-fold cross-validation, the empirical error is the average empirical error from the k folds (i.e., $1/k\ E(h_i, V_i)$). In many cases, outliers or noisy instances are detrimental. However, other instances, such as exceptions, can be detrimental for inducing a model of the data even if they are labeled correctly. Formally, a set $\mathcal{D}$ of detrimental instances is a subset of the training data T that, when used in training, increases the empirical error $E(\cdot, \cdot)$ on a validation set V, i.e., $E(g(T, \lambda), V) > E(g(T - \mathcal{D}, \lambda), V)$ regardless of if the detrimental instances are outliers, exceptions, or neither. The challenge, then, is how to identify detrimental instances. This dissertation addresses this with instance hardness which is described below.

Generally in machine learning, a learning algorithm is trained on all of the training instances. Seeking to increase the classification accuracy on all of the instances in the dataset (when noisy/detrimental instances exist) encourages the learning algorithms to overfit the data. As a result, most models have a mechanism to avoid overfitting such as early stopping or pruning to avoid learning the noisy or outlier training instances. However, most learning algorithms use all of the training instances to generate a model of the data, including the detrimental instances. Thus, the detrimental instances influence the induced model. For example, a decision tree will use all of the instances to determine which attribute to split on. A detrimental instance could cause the learning algorithm to pick a sub-optimal attribute

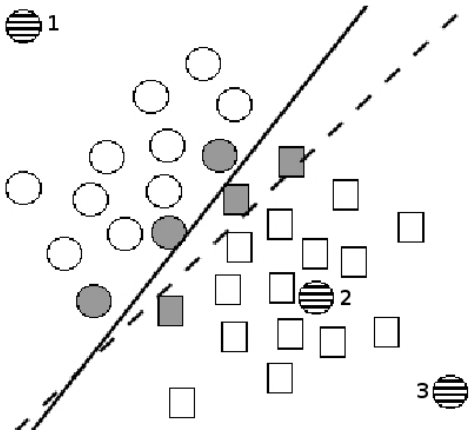

Figure 1.1: A hypothetical 2-dimensional dataset that illustrates how noisy instances (circles with striped fill) affect the model generated by a learning algorithm. The solid line represents the true classification border and the dashed line represents the classification from a learning algorithm that is affected by noisy instances. The filled in instances represent border points.

to split on. Also, when training, many learning algorithms focus on learning the instances that are misclassified and thus seek to learn the noisy instances. For example, multilayer perceptrons update the weights between perceptrons based on the error associated with a given instance. Noisy instances have high error and will have more of an impact on the weight values than the non-noisy instances.

The potential impact of training with detrimental instances is demonstrated in the hypothetical two-dimensional data set shown in Figure 1.1. The solid line represents the true classification boundary (separating the circles from the squares) and the dotted line represents the classification boundary that results from training with the noisy instances (circles with striped fill). The instances with solid fill represent the instances that lie on the border and are consequently most affected by the presence of the outliers. In this example, instances 2 and 3 are outliers that "pull" the classification boundary as the learning algorithm attempts to classify them correctly. As a consequence, two instances are misclassified. In this case, it would be better to simply ignore the noisy instances. Instance 1, although technically an outlier as it is different from the other circle instances, illustrates the point that there are many different types of noisy instances–some that affect classification and others that do not (since it is not misclassified, many learning algorithms, such as a support vector

machine, will not penalize the model). In other learning algorithms, such as a multilayer perceptron trained with backpropagation, instance 1 may shift the classification boundary toward the circles. This will not affect the classification of the outlier instances but may have an impact on the border points that are close to the classification boundary–causing them to be misclassified. However, removing some of these border points that are close to the classification boundary has the advantage of creating simpler classification boundaries and can help avoid overfitting the training data [6].

Besides having a set of instances that are misleading, some instances are more important for learning the hypothesis than other instances are. This is also illustrated in Figure 1.1. The border points (filled-in instances) can be considered more important for defining the classification boundary. In some learning algorithms, such as support vector machines, the instances that define the border are explicitly found when inferring a model of the data. However, the presence of noise and outliers affects the inferred classification boundary and, in turn, the identification of the boundary points.

The question that then remains is how to identify detrimental instances. The brute force approach would be to try every subset of possible training instances and observe which subset performs the best. This approach, of course, is infeasible. This dissertation presents instance hardness[1] and uses instance hardness to identify detrimental instances automatically (without the use of a domain expert). The underlying assumption of using instance hardness is that instances that are frequently misclassified have the largest negative impact on the induced model. To illustrate, consider a multilayer perceptron trained with backpropagation. Instances that have the largest error (i.e. outliers or exceptions) will have the greatest effect on the weight updates. Therefore, instances that have high degrees of error are likely candidates for detrimental instances. When inferring a model from a data set, often all of the instances are initially used indiscriminately, placing a dual task on the machine learning algorithm of determining the importance of the instances while simultaneously inducing a

[1] Chapter 3

model of the data. This dissertation shows some techniques to deal with this issue including filtering[2], instance weighting[3], and ordering the instances[4].

The quantity and nature of the noise in the investigated data sets is unknown. It is assumed that most real-world data sets are noisy to a certain degree, but without some domain knowledge that assumption cannot be confirmed. Thus, in addition to examining the proposed techniques on the unmodified data sets, the techniques are also examined on data sets that have had artificial noise added to verify their effectiveness in the presence of noise[5] as well as without artificial noise.

1.3 Meta-learning

The field of meta-learning seeks to understand the interaction between the training data, the learning algorithms, and their associated hyperparameters as they affect the induced hypothesis. Like traditional machine learning, meta-learning learns from previous examples. Meta-learning differs from traditional machine learning in that meta-learning learns from the performance of multiple applications of a learning algorithm over many data sets. Traditional machine learning learns from the experience on a specific task. Generally speaking, meta-learning is concerned with learning how to learn and can guide the application of machine learning techniques for a given problem. Ultimately, meta-learning aims to automate the process of machine learning. Given a data set, a meta-learning system would determine how to preprocess the training data and select a learning algorithm with its associated hyperparameters that induces the most appropriate hypothesis for that task. Ideally, the optimal hypothesis is desired. However, the optimal hypothesis is dependent on the test data which may not be represented by the training data and finding the optimal hypothesis is NP-hard [3]. Of course, intermediary steps in meta-learning include an advisory role in suggesting step(s) to take.

[2]Chapter 4
[3]Chapter 5
[4]Chapter 6
[5]Chapters 5 and 7

Meta-learning faces several difficulties in reaching its goal, however. The search space for meta-learning is at least exponential and usually infinite. The hypothesis space to search includes all learning algorithms, all possible hyperparameter settings, and all possible variations of the training data. If a learning algorithm has a hyperparameter with continuous values, there are an infinite number of settings to examine. Although generally not considered in meta-learning, this dissertation includes learning about the data and choosing a subset of the training data in meta-learning since the training data has a large bearing on the induced model. The variations of the training data include choosing a subset of the instances to train on, selecting which features to include and/or creating additional features using techniques such as principal component analysis [4]. Very little work in meta-learning has focused on understanding the relationship between training instances which is what this dissertation examines.

1.4 Overview of the Dissertation

Part I of this dissertation provides the background and motivation for this dissertation. It consists of three chapters, including this one. Chapter 2 presents related work in the areas of data complexity, data preprocessing, and meta-learning and how this dissertation falls within the context of previous works. Chapter 3 introduces instance hardness which is used to determine how difficult an instance is to correctly classify. It also presents several measures that serve as instance-level meta-features describing characteristics about each instance.

With the exception of the final chapter, the remainder of this dissertation consists of a collection of papers that have either been published or are in submission for publication in journals or conference proceedings. The references for these papers are listed in the following section and also appear at the beginning of the corresponding chapters.

Having established the problem of finding an appropriate hypothesis for a given problem and the notions of instance hardness and detrimental instances, Part II examines the application of instance-level information into the learning process. Chapters 4 - 7 show sev-

eral results of integrating instance hardness into the learning process. Chapter 4 proposes a method for filtering (removing instances), Chapter 5 proposes a method for weighting instances, and Chapter 6 proposes a method for curriculum learning. Chapter 7 examines the use of classifier diversity for being robust to class noise in filtering and ensembles.

Part III concludes this dissertation. It consists of a three chapters that further motivate the study of instance-level features and a describes a repository to provide data for doing meta-learning at the instance-level. Chapter 8 examines the potential benefits of hyperparameter selection and filtering and shows that the quality of the data has a much larger potential impact than hyperparameter selection. Chapter 9 presents a repository for machine learning results. The goal of the repository is to be a community-based resource for storing and accessing previous machine learning experiments. This data will facilitate the promulgation of meta-learning at the instance and data set-levels. Chapter 10 provides a summary of the contributions of this dissertation as well as some remaining challenges and directions for future work.

In the chapters that follow, a large number of learning algorithms and data sets were used for a more robust investigation. Some experiments were more computationally expensive than others and the learning algorithms and data sets that were used were adjusted accordingly. Data sets and learning algorithms were excluded if they did not finish running due to memory overflow or running time[6] on the Fulton super computer provided by Brigham Young University[7].

1.5 Publications

Chapters 3 - 9 of this dissertation are works that have been published or are in submission for review as a results of this dissertation. The references for these works are listed here according to the part and chapter in which they appear in this dissertation.

[6]The maximum runtime allowed was one week.

[7]`https://marylou.byu.edu/`

I. Background and Motivation

2. Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. "An Instance Level Analysis of Data Complexity", *Machine Learning*, 95(2): 225–256, 2014.

II. Improving Machine Learning by Integrating Meta-information about Individual Training Examples into the Learning Process

3. Michael R. Smith, and Tony Martinez. "Improving Classification Accuracy by Identifying and Removing Instances that Should Be Misclassified", In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2011)*, pages 2690–2697, August 2011.
4. Michael R. Smith, and Tony Martinez. "Reducing the Effects of Detrimental Instances", In *Proceedings of the 13th International Conference on Machine Learning and Applications*, pages 183–188, 2014.
5. Michael R. Smith, and Tony Martinez. "A Comparative Evaluation of Curriculum Learning with Filtering and Boosting in Supervised Classification Problems", Computational Intelligence, to appear, 2014.
6. Michael R. Smith, and Tony Martinez. "Becoming More Robust to Label Noise with Classifier Diversity", In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2015)*, to appear, July 2015.

Additional work that further examines filtering building on the methodology used to calculate instance hardness can be found in the following references.

- Michael R. Smith, and Tony Martinez. "An Extensive Evaluation of Filtering Misclassified Instances in Supervised Classification Tasks", in submission, 2015.

III. Conclusion

7. Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. "The Potential Benefits of Data Set Filtering and Learning Algorithm Hyperparameter Optimization", in submission 2015.

8. Michael R. Smith, Andrew White, Christophe Giraud-Carrier, and Tony Martinez. "An Easy to Use Repository for Comparing and Improving Machine Learning Algorithm Usage", *The ECAI Workshop on Meta-learning & Algorithm Selection (MetaSel)*, pages 41–48, 2014.

Finally, other work that deals with meta-learning at the data set-level that is tangential to this work can be found in the following references.

- Michael R. Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. "Recommending Learning Algorithms and Their Associated Hyperparameters", *The ECAI Workshop on Meta-learning & Algorithm Selection (MetaSel)*, pages 39–40, 2014.
- Michael R. Smith, Michael S. Gashler, and Tony Martinez. "A Hybrid Latent Variable Neural Network Model for Item Recommendation", In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2015)*, to appear, July 2015.

References

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern Recognition and Machine Learning*, volume 1. springer New York, 2006.

[2] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.

[3] Vitaly Feldman, Venkatesan Guruswami, Prasad Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. In *50th Annual IEEE Symposium on Foundations of Computer Science*, pages 385–394, 2009.

[4] Ian T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[5] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[6] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[7] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.

Chapter 2

Related Work

The work presented in this dissertation builds upon the work of others and would not have been possible without their contributions. This chapter briefly surveys the related work and places this dissertation within the context of previous work. More specific related works are provided in each chapter of this dissertation.

2.1 Meta-Learning

The success of machine learning on a given task depends on, among other things, which learning algorithm is selected and its associated hyperparameters. Selecting an appropriate learning algorithm, setting its hyperparameters, and properly handling/preprocessing the data for a specific task can be challenging, especially for users who are not experts in machine learning. However, choosing a set of meta-features that are predictive of algorithm performance is difficult. Most previous meta-learning work has focused on selecting a learning algorithm or a set of hyperparameters based on meta-features used to characterize datasets in isolation (e.g., see [1–3, 5, 24, 28]). Recent work, however, has begun to consider them in tandem. For example, Auto-WEKA simultaneously chooses a learning algorithm and sets its hyperparameters using Bayesian optimization over a tree-structured representation of the combined space of learning algorithms and their hyperparameters [29]. This approach searches the search space, but does not learn from previous results. Also, most of the previous work in meta-learning does not consider preprocessing the data or how to

handle the data. This dissertation provides a foundation for analyzing and understanding the data in machine learning as well as a set of instance-level meta-features.

2.2 Data Complexity

This dissertation focuses on understanding data at the instance-level. Previous work has presented measures to characterize the overall complexity of a data set (primarily using binary classification problems) [13]. Complexity can be defined as the *Kolmogorov complexity*, which states that a pattern is simple if it can be generated by a short program or if it can be compressed [17, 20]. Essentially, the Kolmogorov complexity measures if a pattern has some regularity. In machine learning, the pattern refers to a data set. Further, the data complexity measures have been used for data pruning, or removing complex instances, which are assumed to be noise [19]. Data set measures have also been used in meta learning [6] as well as to understand under what circumstances a particular learning algorithm will perform well [22]. However, data complexity measures characterize the overall complexity of a data set but do not look at the instance level and, thus, cannot say anything about why certain instances are misclassified as examined in this dissertation.

2.3 Instance Filtering/Selection

Most of the previous work in analyzing data at the instance-level has been an implicit evaluation of the data with instance filtering/selection. Identifying the instances that are important has been extensively considered for instance-based (or nearest neighbor) learning algorithms [7, 32]. Identifying the most important instances in the instance-based learning algorithms is important because instance-based learning algorithms generally store all of the instances and classify a new instance by calculating the distance from the new instance to all of the other instances. Therefore, reducing the number of instances stored reduces the storage and computational requirements of the algorithm. Initially, the goal of prior work for instance-based algorithms was to reduce the storage and computational requirements

without loss in classification accuracy. This often included determining which instances are detrimental and which ones are the most important. Smyth and McKenna [27] extended this notion in case-based reasoning by defining for each instance i a coverage set (the set of instances that i contributes to being correctly classified) and a reachability set (the set of instances that contribute to i being correctly classified). In this manner, they identify which instances are beneficial and detrimental in instance-based learning. Support vector machines explicitly focus on identifying the instances that define the classification boundary [9]. Once identified, only the border points are used to define the model. This dissertation builds on this idea of determining how the instances in a data set affect each other in the general context of machine learning rather than for a specific learning algorithm.

Prior work observed that outliers and noisy instances can be detrimental to inducing a model of the data. While the work in instance-based and case-based reasoning focused on competence preservation (same accuracy with less instances), other work has examined competence enhancement (increase the accuracy with less instances). Some previous work has examined how class noise and attribute noise affects the performance of various learning algorithms [23, 33] and found that class noise is generally more harmful than attribute noise and that noise in the training set is more harmful than noise in the test set. Generally, outlier instances in the dataset are first identified and then filtered out of the data set. Next, the filtered data set is used to train the learning algorithm. A popular approach to filtering has been to filter out instances that are misclassified by a learning algorithm [15, 30] or an ensemble of learning algorithms [8, 31]. The assumption is that if an instance is misclassified, then it is an outlier. In general, there is an increase in classification accuracy by filtering out the outlier instances [12, 25, 34]. However, the instances that are identified as being noisy or outliers are dependent on the learning algorithm(s). Filtering can also be detrimental if too many instances are removed from the data set or there are not a lot of instances to begin with.

2.4 Parameter Tuning/Modification

Rather than filtering the outlier/noisy instances, another approach focuses on making the learning algorithm more tolerant to noise. For example, there are a number of approaches to make perceptrons and support vector machines more noise tolerant by introducing a slack variable that allows for noisy instances [16]. However, in many cases a learning algorithm is still affected by the presence of outliers since the outliers are used to induce the model. This is because learning algorithms optimize the classification accuracy on the training instances, so often outliers receive the most attention. This is especially apparent for boosting algorithms [10, 26] that ensemble a group of trained models. Each model is trained on a subset of the data sampled with more weight being placed on the misclassified instances (outliers would receive more weight). This approach is susceptible to overfitting the outlier and noisy instances. Long and Servedio [21] showed the severity of this problem by showing that boosting algorithms are incapable of learning a convex potential function in the presence of random noise. There have been several proposed modifications to boosting algorithms to decrease focus on learning outliers [11].

Rather than modifying a learning algorithm, the hyperparameters can be optimized to be robust to noise. The grid search and manual search are the most common types of hyperparameter optimization techniques in machine learning and a combination of the two approaches is commonly used [18]. Bergstra and Bengio [3] proposed to use a random search of the hyperparameter space. The premise of random hyperparameter optimization is that most machine learning algorithms have very few hyperparameters that considerably affect the final model while the other hyperparameters have little to no effect. Random search provides a greater variety of the hyperparameters that considerably affect the model. Given the same amount of time constraints, random hyperparameter optimization has been shown to outperform a grid search. Bayesian optimization has also been used to search the hyperparameter space [28]. Bayesian optimization techniques model the dependence of an

error function $\mathcal{E}$ on the hyperparameters λ as $p(\mathcal{E}, \lambda)$ using, for example, a tree-structured Parzen estimator [4] or Gaussian processes [14].

References

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

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

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[5] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[6] Pavel B. Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer, 2009.

[7] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, 2002. ISSN 1384-5810.

[8] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[9] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[10] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

[11] Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

[12] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. *Applied Artificial Intelligence*, 14(2):205–223, 2000.

[13] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:289–300, March 2002.

[14] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Learning and Intelligent Optimization Conference*, pages 507–523, 2011.

[15] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179, 1995.

[16] Roni Khardon and Gabriel Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, May 2007. ISSN 1532-4435.

[17] Andrey Nikolaevich Kolmogorov. Three aproaches to the qualitative definition of information. *Problems of Information Transmission*, 1:4–7, 1965.

[18] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation.

In *Proceedings of the 24th International Conference on Machine Learning*, pages 473–480, 2007.

[19] Ling Li and Yaser S. Abu-Mostafa. Data complexity in machine learning. Computer Science Technical Report Caltech CSTR:2006.004, May 2006. URL `http://resolver.caltech.edu/CaltechCSTR:2006.004`.

[20] Ming Li and Paul M.B. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 edition, 2008.

[21] Philip M. Long and Rocco A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78:287–304, March 2010.

[22] Ester Bernadó Mansilla and Tin Kam Ho. On classifier domains of competence. In *ICPR (1)*, pages 136–139, 2004.

[23] David F. Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.

[24] Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, pages 743–750, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[25] J. S. Sánchez, R. Barandela, A. I. Marqués, R. Alejo, and J. Badenas. Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters*, 24:1015–1022, April 2003. ISSN 0167-8655.

[26] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[27] Barry Smyth and Elizabeth McKenna. Modelling the competence of case-bases. In *Advances in Case-Based Reasoning, 4th European Workshop on Case-Based Reasoning*, pages 208–220. Springer-Verlag, 1998.

[28] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. 2012.

[29] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.

[30] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

[31] Sofie Verbaeten and Anneleen Van Assche. Ensemble methods for noise elimination in classification problems. In *Proceedings of the 4th international conference on multiple classifier systems*, pages 317–325, 2003.

[32] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[33] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artificial Intelligence Review*, 22:177–210, November 2004.

[34] Xingquan Zhu, Xindong Wu, and Qijun Chen. Eliminating class noise in large datasets. In *In Proceeding of International Conference on Machine Learning (ICML2003)*, pages 920–927, 2003.

Chapter 3

An Instance Level Analysis of Data Complexity

Machine Learning, vol. 95, no. 2, pp. 225-256 2014.

Abstract

Most data complexity studies have focused on characterizing the complexity of the entire data set and do not provide information about individual instances. Knowing which instances are misclassified and understanding why they are misclassified and how they contribute to data set complexity can improve the learning process and could guide the future development of learning algorithms and data analysis methods. The goal of this paper is to better understand the data used in machine learning problems by identifying and analyzing the instances that are frequently misclassified by learning algorithms that have shown utility to date and are commonly used in practice. We identify instances that are hard to classify correctly (*instance hardness*) by classifying over 190,000 instances from 64 data sets with 9 learning algorithms. We then use a set of hardness measures to understand why some instances are harder to classify correctly than others. We find that class overlap is a principal contributor to instance hardness. We seek to integrate this information into the training process to alleviate the effects of class overlap and present ways that instance hardness can be used to improve learning.

3.1 Introduction

It is widely acknowledged in machine learning that the performance of a learning algorithm is dependent on both its parameters and the training data. Yet, the bulk of algorithmic de-

velopment has focused on adjusting model parameters without fully understanding the data that the learning algorithm is modeling. As such, algorithmic development for classification problems has largely been measured by classification accuracy, precision, or a similar metric on benchmark data sets. These metrics, however, only provide aggregate information about the learning algorithm and the task upon which it operates. They fail to offer any information about which instances are misclassified, let alone why they are misclassified. There is some speculation as to why some instances are misclassified, but, to our knowledge, no thorough investigation (such as the one presented here) has taken place.

Previous work on instance misclassification has focused mainly on isolated causes. For example, it has been observed that outliers are often misclassified and can affect the classification of other instances [2]. Border points and instances that belong to a minority class have also been found to be more difficult to classify correctly [9, 43]. As these studies have had a narrow focus on trying to identify and handle outliers, border points, or minority classes, they have not generally produced an agreed-upon definition of what characterizes these instances. At the data set level, previous work has presented measures to characterize the overall complexity of a data set [18]. Data set measures have been used in meta learning [6] as well as to understand under what circumstances a particular learning algorithm will perform well [25]. As with the performance metrics, the data complexity measures characterize the overall complexity of a data set but do not look at the instance level and thus cannot say anything about why certain instances are misclassified. It is our contention that identifying which instances are misclassified and understanding why they are misclassified can lead to improvements in machine learning algorithm design and application.

The misclassification of an instance depends on the learning algorithm used to model the task it belongs to and its relationship to other instances in the training set. Hence, any notion of *instance hardness*, i.e., the likelihood of an instance being misclassified, must be a relative one. However, generalization beyond a single learning algorithm can be achieved by aggregating the results from multiple learning algorithms. We use this fact to propose

an empirical definition of instance hardness based on the classification behavior of a set of learning algorithms that have been selected because of 1) their diversity, 2) their utility, and 3) their wide practical applicability. We then present a thorough analysis of instance hardness, and provide insight as to why hard instances are frequently misclassified. To the best of our knowledge our research is the first at reporting on a systematic and extensive investigation of the issue.

We analyze instance hardness in over 190,000 instances from 64 classification tasks classified by nine learning algorithms. We find that a considerable amount of instances are hard to classify correctly–17.5% of the investigated instances are misclassified by at least half of the considered learning algorithms and 2.3% are misclassified by all of the considered learning algorithms. Seeking to improve our understanding of why these instances are misclassified becomes a justifiable quest. To discover why these instance are hard to classify, we introduce a set of measurements, (*hardness measures*). The results suggest that class overlap has the strongest influence on instance hardness and that there may be other features that affect the hardness of an instance. Although we focus on hardness at the instance level, the measures can also be used at the data set level by averaging the values of the instances in the data set. Further, we incorporate instance hardness into the learning process by modifying the error function of a multilayer perceptron and by filtering instances. These methods place more emphasis on the non-overlapping instances, alleviating the effects of class overlap. We demonstrate that incorporating instance hardness into the learning process can significantly increase classification accuracy.

The remainder of the paper is organized as follows. In Section 3.2, we introduce and define instance hardness as an effective means of identifying instances that are frequently misclassified. The hardness measures are presented in Section 3.3 as a means of providing insight into why an instance is hard to classify correctly. Section 3.4 presents the experimental methodology. An analysis of hardness at the instance level is provided in Section 3.5 followed by Section 3.6 which demonstrates that improved accuracy can follow from inte-

grating instance hardness into the learning process. Section 3.7 compares instance hardness at the data set level with previous data set complexity studies. Section 3.8 provides related works and Section 3.9 concludes the paper.

3.2 Instance Hardness

Our work posits that each instance in a data set has a hardness property that indicates the likelihood that it will be misclassified. For example, outliers and mislabeled instances are expected to have high instance hardness since a learning algorithm will have to overfit to classify them correctly. Instance hardness seeks to answer the important question of what is the probability that an instance in a particular data set will be misclassified.

As most machine learning research is focused on the data set level, one is concerned with maximizing $p(h|t)$, where $h : X \rightarrow Y$ is a hypothesis or function mapping input feature vectors X to their corresponding label vectors Y, and $t = \{(x_i, y_i) : x_i \in X \wedge y_i \in Y\}$ is a training set. With the assumption that the pairs in t are drawn i.i.d., the notion of instance hardness is found through a decomposition of $p(h|t)$ using Bayes' theorem:

$$\begin{aligned} p(h|t) &= \frac{p(t|h)\ p(h)}{p(t)} \\ &= \frac{\prod_{i=1}^{|t|} p(x_i, y_i|h)\ p(h)}{p(t)} \\ &= \frac{\prod_{i=1}^{|t|} p(y_i|x_i, h)\ p(x_i|h)\ p(h)}{p(t)}. \end{aligned}$$

For a training instance $\langle x_i, y_i \rangle$, the quantity $p(y_i|x_i, h)$ measures the probability that h assigns the label y_i to the input feature vector x_i. The larger $p(y_i|x_i, h)$ is, the more likely h is to assign the correct label to x_i, and the smaller it is, the less likely h is to produce the correct label for x_i. Hence, we obtain the following definition of instance hardness, with respect to h:

$$IH_h(\langle x_i, y_i \rangle) = 1 - p(y_i|x_i, h).$$

In practice, h is induced by a learning algorithm g trained on t with hyperparameters α, i.e., $h = g(t, \alpha)$. Thus, the hardness of an instance is dependent on the instances in the training data and the algorithm used to produce h. There are many approaches that could be taken to calculate instance hardness (or equivalently $p(y_i|x_i, g(t, \alpha))$) such as an analysis of the distribution of instances in t according to their class. To gain a better understanding of what causes instance hardness in general, the dependence on a specific hypothesis can be lessened by integrating instance hardness over the set of hypotheses $\mathcal{H}$:

$$\begin{aligned} IH(\langle x_i, y_i \rangle) &= \int_{\mathcal{H}} 1 - p(y_i|x_i, h)dh \\ &= 1 - \int_{\mathcal{H}} p(y_i|x_i, h)dh \\ &= 1 - \int_{\mathcal{H}} p(y_i|x_i, t, h)dh \\ &= 1 - \int_{\mathcal{H}} p(y_i|x_i, h)p(h|t)dh. \end{aligned} \tag{3.1}$$

Note that t can be added in the above derivation because y_i is conditionally independent of t given h. Practically, to integrate over $\mathcal{H}$, one would have to integrate over the complete set of hypotheses, or, since $h = g(t, \alpha)$, over the complete set of learning algorithms and hyperparameters associated with each algorithm. This, of course, is not feasible. In practice, instance hardness can be estimated by restricting attention to a carefully chosen set of representative algorithms (and parameters). Also, it is important to estimate $p(h|t)$ because if all hypotheses were equally likely, then all instances would have the same instance hardness value under the no free lunch theorem [45]. A natural way to approximate the unknown distribution $p(h|t)$, or equivalently $p(g(t, \alpha))$, is to weigh a set of representative learning algorithms, and their associated parameters, $\mathcal{L}$, a priori with a non-zero probability while treating all other learning algorithms as having zero probability. Given such a set $\mathcal{L}$ of learning algorithms, we can then approximate Equation 3.1 to the following using stochastic

Table 3.1: Set $\mathcal{L}$ of ESLAs used to calculate instance hardness.

Learning Algorithms	
* RIpple DOwn Rule learner (RIDOR)	* Naïve Bayes
* Multilayer Perceptron trained with Back Propagation	* Random Forrest
* Locally Weighted Learning (LWL)	* 5-nearest neighbors (5-NN)
* Nearest Neighbor with generalization (NNge)	* Decision Tree (C4.5 [30])
* Repeated Incremental Pruning to Produce Error Reduction (RIPPER)	

integration:

$$IH_{\mathcal{L}}(\langle x_i, y_i \rangle) = 1 - \frac{1}{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} p(y_i | x_i, g_j(t, \alpha)) \tag{3.2}$$

where $p(h|t)$ is approximated as $\frac{1}{|\mathcal{L}|}$ and the distribution $p(y_i|x_i, g_j(t, \alpha))$ is estimated using the indicator function and classifier scores, as described in Section 3.4. For simplicity, we refer to $IH_{\mathcal{L}}$ as simply IH proceeding forward.

In this paper, we estimate instance hardness by biasing the selection of representative learning algorithms to those that 1) have shown utility, and 2) are widely used in practice. We call such classification learning algorithms the *empirically successful learning algorithms* (ESLAs). To get a good representation of $\mathcal{H}$, and hence a reasonable estimate of IH, we select a diverse set of ESLAs using unsupervised metalearning [23]. Unsupervised metalearning uses Classifier Output Difference (COD) [28] to measure the diversity between learning algorithms. COD measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions. Unsupervised metalearning then clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. Here, we considered 20 commonly used learning algorithms with their default parameters as set in Weka [17]. The resulting dendrogram is shown in Figure 3.1, where the height of the line connecting two clusters corresponds to the distance (COD value) between them. A cut-point of 0.18 was chosen and a representative algorithm from each cluster was used to create $\mathcal{L}$ as shown in Table 3.1.

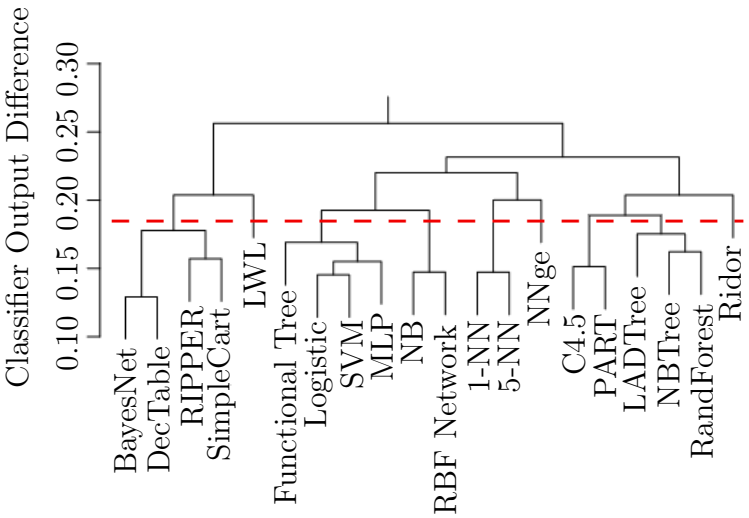

Figure 3.1: Dendrogram of the considered learning algorithms clustered using unsupervised metalearning.

We recognize that instance hardness could be calculated with either more specific or broader sets of learning algorithms, and each set would obtain somewhat different results. We also recognize that the set of ESLAs is constantly evolving and thus no exact solution is possible. As the set of ESLAs grows and evolves, instance hardness can follow this evolution by simply adjusting $\mathcal{L}$. The size and exact make up of $\mathcal{L}$ are not as critical as getting a fairly representative sample of ESLAs. While more learning algorithms may give a more accurate estimate of instance hardness, we demonstrate that both efficiency and accuracy can be achieved with a relatively small and diverse set of learning algorithms.

With this approach, the instance hardness of an instance is dependent both on the learning algorithm trying to classify it and on its relationship to the other instances in the data set as demonstrated in the hypothetical two-dimensional data set shown in Figure 3.2. Instances A, C, and D could be considered outliers, though they vary in how hard they are to classify correctly: instance A would almost always be misclassified while instances C and D would almost always be correctly classified. The instances inside of the dashed oval represent *border points*, which would have a greater degree of hardness than the non-outlier instances that lie outside the dashed oval. Obviously, some instances are harder for some learning algorithms than for others. For example, some instances (such as instance B) are

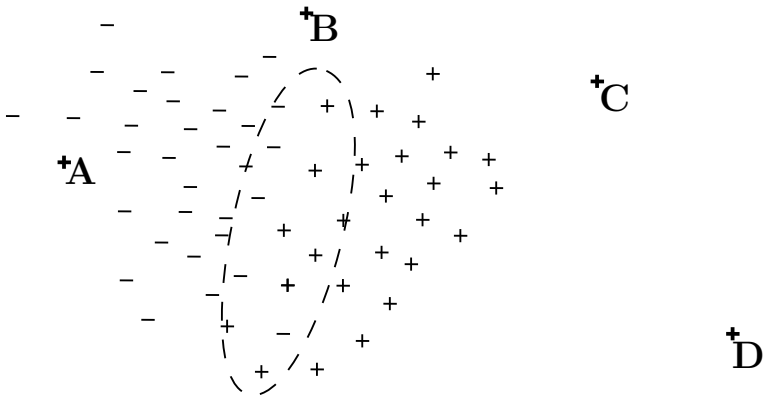

Figure 3.2: Hypothetical 2-dimensional data set.

harder for a linear classifier than for a non-linear classifier because a non-linear classifier is capable of producing more complex decision boundaries.

3.3 Hardness Measures

In this section, we present a set of measures that measure various aspects about the instance hardness level of an individual instance. Instance hardness indicates which instances *are* misclassified while the hardness measures are intended to indicate *why* they are misclassified. Each hardness measure measures an aspect of why an instance may be misclassified (class overlap, class skew, etc.) and, thus, gives key insights into: 1) why particular instances are hard to classify, 2) how we could detect them, and 3) potentially creating improved mechanisms to deal with them. In addition, a subset of the measures could be used as a less expensive alternative to estimate instance hardness, although this is not investigated in this paper.

The set of hardness measures was discovered by examining the learning mechanisms of several learning algorithms. In compiling a set of hardness measures, we chose to use those that are relatively fast to compute and are interpretable so as to provide an indication as to why an instance is misclassified.

***k*-Disagreeing Neighbors (*k*DN).** kDN measures the local overlap of an instance in the original task space in relation to its nearest neighbors. The kDN of an instance is the percentage of the k nearest neighbors (using Euclidean distance) for an instance that do not share its target class value.

$$kDN(x) = \frac{\mid \{y : y \in kNN(x) \wedge t(y) \neq t(x)\} \mid}{k}$$

where $kNN(x)$ is the set of k nearest neighbors of x and $t(x)$ is the target class for x.

Disjunct Size (DS). DS measures how tightly a learning algorithm has to divide the task space to correctly classify an instance and the complexity of the decision boundary. Some learning algorithms, such as decision trees and rule-based learning algorithms, can express the learned concept as a disjunctive description. Thus, the DS of an instance is the number of instances in a disjunct divided by the number of instances covered by the largest disjunct in a data set.

$$DS(x) = \frac{\mid disjunct(x) \mid -1}{\max_{y \in D} \mid disjunct(y) \mid -1}$$

where the function $disjunct(x)$ returns the disjunct that covers instance x, and D is the data set that contains instance x. The disjuncts are formed using a slightly modified[1] C4.5 [30] decision tree, created without pruning and setting the minimum number of instances per leaf node to 1[2].

[1]The C4.5 algorithm stops splitting the data when a sufficient increase in information gain is not achieved. The idea of the DS measure is to overfit the data. However, not splitting all the way down led to impure disjuncts. Therefore, we modified the strictness of when to stop splitting such that all instances were carried out as far as they could go. The only impure disjuncts that remained are those for instances that have the same attribute values but differ in the class value.

[2]Note that C4.5 will create fractional instances in a disjunct for instances with unknown attribute values, possibly leading to DS values less than 1. Such cases are treated as though the disjunct covered a single instance.

Disjunct Class Percentage (DCP). DCP measures the overlap of an instance on a subset of the features. Using a pruned C4.5 tree, the DCP of an instance is the number of instances in a disjunct belonging to its class divided by the total number of instances in the disjunct.

$$DCP(x) = \frac{\mid \{z : z \in disjunct(x) \wedge t(z) = t(x)\} \mid}{\mid disjunct(x) \mid}$$

Tree Depth (TD). Decision trees also provide a way to estimate the description length, or Kolmogorov complexity, of an instance. The depth of the leaf node that classifies an instance can give an intuition of the description length required for an instance. For example, an instance that requires 15 attribute splits before arriving at a leaf node is more complex than an instance that only requires 1 attribute split. Therefore, *tree depth* measures the depth of the leaf node for an instance in an induced C4.5 decision tree (both pruned (TD_P) and unpruned (TD_U)) as an estimate of the minimum description length for an instance.

Class Likelihood (CL). CL provides a global measure of overlap and the likelihood of an instance belonging to a class. The CL of an instance belonging to a certain class is defined as:

$$CL(x) = \prod_{i}^{|x|} P(x_i|t(x))$$

where $|x|$ is the number of attributes of instance x and x_i is the value of instance x's ith attribute[3]. The prior term is excluded in order to avoid bias against instances that belong to minority classes. CL assumes independence between the data attributes.

[3]Continuous variables are assigned a probability using a kernel density estimation [19].

Class Likelihood Difference (CLD). CLD captures the difference in likelihoods and global overlap. It is the difference between the class likelihood of an instance and the maximum likelihood for all of the other classes.

$$CLD(x) = CL(x) - \underset{y \in Y - t(x)}{\operatorname{argmax}} CL(x, y)$$

where Y represents set of possible labels in the data set.

Minority Value (MV). MV measures the skewness of the class that an instance belongs to. For each instance, its MV is the ratio of the number of instances sharing its target class value to the number of instances in the majority class.

$$MV(x) = 1 - \frac{\mid \{z : z \in D \wedge t(z) = t(x)\} \mid}{\max_{y \in Y} \mid \{z : z \in D \wedge t(z) = y\} \mid}.$$

Class Balance (CB). CB also measures the skewness of the class that an instance belongs to and offers an alternative to MV. If there is no class skew, then there is an equal number of instances for all classes. Hence, the CB of an instance is:

$$CB(x) = \frac{\mid \{z : z \in D \wedge t(z) = t(x)\} \mid}{\mid D \mid} - \frac{1}{\mid Y \mid}.$$

If the data set is completely balanced the class balance value will be 0.

For convenience, Table 3.2 summarizes the hardness measures and what they measure. Although all of the hardness measures are intended to understand why an instance is hard to classify, some of the measures indicate how easy an instance is to classify (they have a negative correlation with instance hardness). For example, the class likelihood (CL) measures how likely an instance belongs to a certain class. High values for CL would represent easier

Table 3.2: List of hardness measures and what they measure.

Abbr	+/-	Measure	Insight
kDN	+	k-Disagreeing Neighbors	Overlap of an instance using all of the data set features on a subset of the instances.
DS	-	Disjunct Size	Complexity of the decision boundary for an instance.
DCP	-	Disjunct Class Percentage	Overlap of an instance using a subset of the features and a subset of the instances.
TD	+	Tree Depth	The description length of an instance in an induced C4.5 decision tree.
CL	-	Class Likelihood	Overlap of an instance using all of the features and all of the instances.
CLD	-	Class Likelihood Difference	Relative overlap of an instance using all of the features and all of the instances.
MV	+	Minority Value	Class skew.
CB	-	Class Balance	Class skew.

instances. In Table 3.2, the "+" and "-" symbols distinguish which instances are positively and negatively correlated with instance hardness.

Class overlap and class skew are two commonly assumed and observed causes of instance hardness that are measured with the hardness measures. Mathematically, the class overlap of an instance for a binary task can be expressed as:

$$classOverlap(\langle x_i, y_i \rangle) = p(\bar{y}_i|x_i, t) - p(y_i|x_i, t). \tag{3.3}$$

where $\bar{y}_i$ represents an incorrect class for the input feature vector x_i. The class skew of the class of an instance can be expressed as:

$$classSkew(\langle x_i, y_i \rangle) = \frac{p(y_i|t)}{p(\bar{y}_i|t)}. \tag{3.4}$$

There is no known method to measure class overlap or to determine when class skew affects instance hardness. The hardness measures allow a user to estimate class overlap and class

skew as well as other uncharacterized sources of hardness. Equations 3.3 and 3.4 could be extended to multi-class problems with a 1 vs. 1, or a 1 vs. all approach.

3.4 Experimental Methodology

In this section we provide our experimental methodology. Recall that to compute the instance hardness of an instance x, we must compute the probability that x is misclassified when the learner is trained on the other points from the dataset. Since this type of leave-one-out procedure is computationally prohibitive, the learning algorithms are evaluated using 5 by 10-fold cross-validation[4]. We use five repetitions to better measure the instance hardness of each instance and to protect against the dependency on the data used in each fold. We then compare the hardness measures with instance hardness.

We examine instance hardness on a large and varied set of data sets chosen with the intent of being representative of those commonly encountered in machine learning problems. We analyze the instances from 57 UCI data sets [14] and 7 non-UCI data sets [31, 32, 39, 40]. Table 3.3 shows the data sets used in this study organized according to the number of instances, number of attributes, and attribute type. The non-UCI data sets are in bold.

We compare calculating instance hardness using all of the learning algorithms in $\mathcal{L}$ with calculating instance hardness using a single learning algorithm. In addition, $p(y_i|x_i, g(t, \alpha))$ is estimated using two methods: 1) the indicator function (IH_ind) and 2) the classifier scores (IH_class). IH_ind and IH_class are calculated using 5 by 10-fold cross-validation. Generally, classification learning algorithms classify an instance into nominal classes. To produce a real-valued score, we calculate classifier scores for the nine investigated learning algorithms. Obviously, the indicator function and the classifier scores do not produce true probabilities. However, the classifier scores can provide the confidence of an inferred model for the class label of an instance. Below, we present how we calculate the classifier scores for the investigated learning algorithms.

[4]5 by 10-fold cross-validation runs 10-fold cross-validation 5 times, each time with a different random seed for selecting the 10 partitions of the data.

Table 3.3: Datasets used organized by number of instances, number of attributes, and attribute type.

# Instances	# Attributes	Attribute Type		
		Categorical	Numerical	Mixed
$M < 100$	$k < 10$	Balloons Contact Lenses		Post-Operative **cm1_req**
	$10 < k < 100$	Lung Cancer	**desharnais**	Labor Trains **Pasture**
$100 < M < 1000$	$k < 10$	Breast-w Breast Cancer	Iris Ecoli Pima Indians Glass Bupa Balance Scale	Badges 2 Teaching- Assistant
	$10 < k < 100$	Audiology Soybean(large) Lymphography Congressional- Voting Records Vowel Primary-Tumor Zoo	Ionosphere Wine Sonar Heart-Statlog **ar1**	Annealing Dermatology Credit-A Credit-G Horse Colic Heart-c Hepatitis Autos Heart-h **eucalyptus**
	$k > 100$		**AP_Breast_Uterus**	Arrhythmia
$1000 < M < 10000$	$k < 10$	Car Evaluation Chess Titanic	Yeast	Abalone
	$k < 100$	Mushroom Splice	Waveform-5000 Segment Spambase Ozone level- Detection	Thyroid- (sick & hypothyroid)
	$k > 100$		Musk (version 2)	
$M > 10000$	$k < 10$	Nursery	MAGIC Gamma- Telescope	
	$k < 100$	Chess- (King-Rook vs. King-Pawn) Letter		Adult-Census- Income (KDD) **Eye movements**

Multilayer Perceptron: For multiple classes, each class from a data set is represented with an output node. The classifier score is the largest value of the output nodes normalized between zero and one (the *softmax* [8]):

$$\hat{p}(y|x) = \frac{o_i(x)}{\sum_i^{|Y|} o_i(x)}$$

where y is a class from the set of possible classes Y and o_i is the value from the output node corresponding to class y_i

Decision Tree: To calculate a classifier score, an instance first follows the induced set of rules until it reaches a leaf node. The classifier score is number of training instances that have the same class as the examined instance divided by all of the training instances that also reach the same leaf node.

5-NN: 5-NN returns the percentage of the nearest-neighbors that agree with the class label of an instance as the classifier score.

LWL: LWL finds the k-nearest neighbors for an instance from the training data and weights them by their distance from the test instance. The weighted k-nearest neighbors are then used to train a base classifier. Weka uses a decision stump as the base classifier. A decision stump is a decision tree that makes a singe binary split on the most informative attribute. A test instance is propagated to a leaf node. The sum of weights of the training instances in the leaf node that have the same class value as the test instance is divided by the sum of the weights of all of the training instances in the leaf node.

Naïve Bayes: Returns the probability of the most probable class by multiplying the probability of the class by the probabilities of the attribute values for an instance given the class:

$$\max_{y_j \in Y} p(y_j) \prod_i^{|x|} p(x_i|y_j).$$

NNge: Since NNge only keeps exemplars of the training data, a class score of 1 is returned if an instance agrees with the class of the nearest exemplar, otherwise a 0 is returned.

Random Forest: Random forests return the class counts from the leaf nodes of each tree in the forest. The counts for each class are summed together and then normalized between 0 and 1.

RIDOR: RIDOR creates a set of rules, but does not keep track of the number of training instances covered by a rule. A classifier score of 1 is returned if RIDOR predicts the correct class for an instance, otherwise a 0 is returned (same as the indicator function).

RIPPER: RIPPER returns the percentage of training instances that are covered by a rule and share the same class as the examined instance.

To our knowledge, instance hardness is the only measurement that seeks to identify instances that are hard to classify. However, there are other methods that could be used to identify hard instances that have not been examined for identifying hard instances. One such method that we compare against is active learning. Active learning is a semi-supervised technique that uses a mode inferred from the labeled instances to choose which unlabeled instances are the most informative to be labeled by an external oracle. The informative scores assigned by active learning techniques can be used as a hardness measure. This assumes that the most informative instances are those that the model is least certain about, which would include the border points. We implemented two active learning techniques: uncertainty sampling (US) [24] and query-by-committee (QBC) [36]. For uncertainty sampling, we use margin sampling [33]:

$$x^* = \underset{x}{\operatorname{argmin}}\, p(\hat{y}_1|x) - p(\hat{y}_2|x)$$

where $\hat{y}_1$ and $\hat{y}_2$ are the first and second most probable class labels for the instance x. We use naïve Bayes to calculate the probability of the classes for an instance. For query-by-committee, we use a committee of five learning algorithms using query by bagging [1]. The

level of disagreement is determined using vote entropy [12]:

$$x^* = \underset{x}{\text{argmax}} - \sum_i \frac{V(y_i)}{C} log \frac{V(y_i)}{C}$$

where y_i ranges over all possible class labels, $V(y_i)$ is the number of votes that a class label received from the committee, and C is the size of the committee. We examine QBC using naïve Bayes and decision trees. Active learning requires that some labeled instances are available to the models to produce the scores for the other instances. We divide the data set in half, using one half of the instances to calculate the scores for the other half.

We emphasize the extensiveness of our analysis. We examine over 190,000 instances individually. A total of 28,750 models are produced from 9 learning algorithms trained with 64 data sets using 5 by 10-fold cross-validation [5]. With this volume and diversity, our results can provide more useful insight about the extent to which hard instances exist and what contributes to instance hardness.

3.5 Instance-level Analysis

In this section we examine the hardness measures to identify hard instances and the hardness measures to discover what causes an instance to be misclassified. We use instance hardness with the indicator function (IH_ind) to establish the frequency of an instance being misclassified. Figure 3.3 shows the cumulative percentage of instances that are misclassified a specified percentage of times by the learning algorithms in $\mathcal{L}$ (Table 3.1). The first pair of columns shows that all of the instances were classified correctly by zero or more of the considered learning algorithms. The second pair of columns shows the percentage of instances that were misclassified by at least one of the considered learning algorithms. Overall, 2.4% of the instances from the UCI data sets are misclassified by all of the considered learning algorithms and 16.8% are misclassified by at least half. For the instances from the non-UCI

[5]Ridor was not used on the Letter data set as it ran out of memory with 4 Gb of RAM. The remaining 8 learning algorithms still give a good indication of how difficult each instance is to correctly classify.

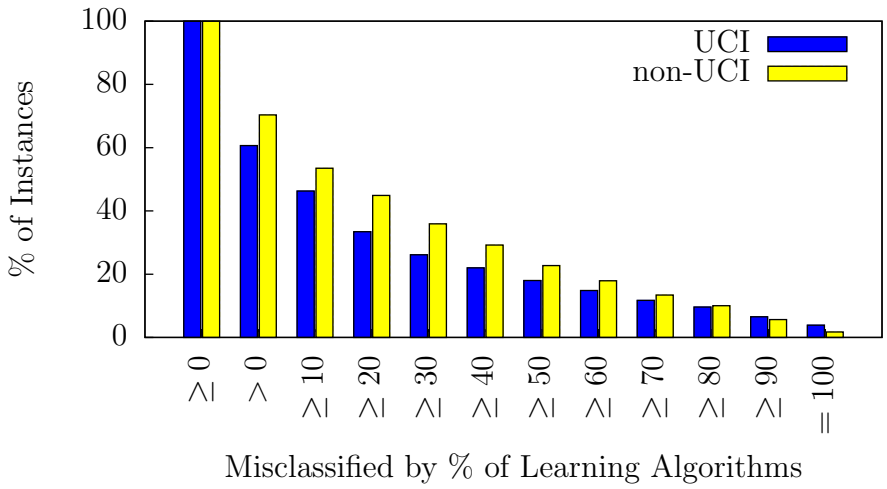

Figure 3.3: Percentage of instances that are misclassified by at least a percentage of the learning algorithms.

data sets, 1.7% are misclassified by all of the considered learning algorithms and 22.7% are misclassified by at least half. The trend of hardness is similar for the UCI and non-UCI data sets. For the set of instances from the UCI and non-USI data sets, only 38.3% of the instances are classified correctly 100% of the time by the examined learning algorithms. These results show that a considerable amount of instances are hard to classify correctly. Seeking to improve our understanding of why these instances are misclassified is the goal of the hardness measures.

We calculate the hardness measures for all of the instances regardless of their instance hardness. We first examine the relationship between the hardness measures. This will provide insight into how similar the measures are with each other and detect possible overlap in what they measure (see Table 3.2 for the hardness measures and what they measure). Next, we examine the relationship of the hardness measures with the hardness measures. We first normalize the measures by subtracting the mean and dividing by the standard deviation for each measure before analyzing the results.

We first examine the correlation between the hardness measures. Table 3.4 shows a pairwise comparison of the hardness measures using the Spearman correlation. Only (CL)

Table 3.4: Spearman correlation matrix for the hardness measures. The magnitude of only one pair of measures is stronger than 0.95, showing that the measures measure different aspects of instance hardness.

	kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB
kDN	1.0	-0.519	-0.420	0.189	0.301	-0.715	-0.703	0.387	0.240
DS		1.0	0.570	-0.405	-0.348	0.571	0.559	-0.303	-0.139
DCP			1.0	-0.340	-0.202	0.452	0.432	-0.235	-0.051
TD_P				1.0	0.859	-0.276	-0.312	0.030	0.113
TD_U					1.0	-0.414	-0.441	0.162	0.293
CL						1.0	**0.989**	-0.386	-0.225
CLD							1.0	-0.359	-0.224
MV								1.0	0.783
CB									1.0

and class likelihood difference (CLD) are strongly correlated with a correlation coefficient of 0.989. This suggests that, besides CL and CLD, the hardness measures measure different properties of the hardness of an instance.

The more interesting question to consider is how does instance hardness relate to the considered hardness measures. Table 3.5 shows the Spearman correlation coefficients relating instance hardness to the other considered hardness measures for the UCI and non-UCI data sets. The hardness measure with the strongest correlation with each instance hardness method is in bold. The first section of the table uses the indicator function to calculate instance hardness, the second section uses the classifier scores to calculate instance hardness, and the third section shows the results for active learning. IH_ind and IH_class use the indicator function and classifier scores respectively from all of the learning algorithms in $\mathcal{L}$ to calculate instance hardness. The following rows use a single learning algorithm to calculate instance hardness. For all of the hardness measures, kDN, DCP, CL, and CLD have the strongest correlation with all of the hardness measures. Using PCA on the hardness measures, kDN, CL, and CLD have the largest coefficients for the first principal component (thus accounting for more variance than the other measures). kDN, CL, and CLD measure class overlap using all of the features from the data set. The other measures (which measure

Table 3.5: The Spearman correlation coefficients for the hardness measures relating to the examined methods for identifying hard instances.

		kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB	Lin
Indicator Function	IH_ind	**0.830**	-0.547	-0.475	0.324	0.475	-0.670	-0.660	0.522	0.436	0.885
	MLP	**0.668**	-0.420	-0.397	0.270	0.354	-0.484	-0.476	0.367	0.287	0.725
	C.5	**0.625**	-0.459	-0.453	0.262	0.353	-0.469	-0.458	0.361	0.299	0.801
	5-NN	**0.648**	-0.307	-0.295	0.254	0.304	-0.319	-0.318	0.293	0.230	0.738
	LWL	**0.549**	-0.363	-0.302	0.149	0.288	-0.484	-0.470	0.506	0.447	0.671
	NB	**0.545**	-0.339	-0.302	0.162	0.284	-0.506	-0.499	0.405	0.328	0.626
	NNge	**0.716**	-0.464	-0.432	0.304	0.394	-0.552	-0.543	0.359	0.277	0.753
	RandFor	**0.669**	-0.490	-0.448	0.276	0.349	-0.487	-0.479	0.393	0.312	0.760
	Ridor	**0.711**	-0.437	-0.391	0.216	0.355	-0.575	-0.558	0.473	0.398	0.761
	RIPPER	**0.675**	-0.420	-0.387	0.172	0.306	-0.555	-0.538	0.496	0.350	0.747
Classifier Score	IH_class	**0.875**	-0.615	-0.540	0.341	0.513	-0.782	-0.767	0.542	0.425	0.938
	MLP	**0.764**	-0.515	-0.528	0.399	0.516	-0.679	-0.667	0.419	0.324	0.809
	C4.5	0.680	-0.629	**-0.711**	0.452	0.483	-0.644	-0.627	0.391	0.234	0.874
	5-NN	***0.771***	-0.316	-0.309	0.000	0.187	-0.574	-0.556	0.384	0.232	0.818
	LWL	**0.736**	-0.538	-0.432	0.198	0.426	**-0.745**	-0.718	0.615	0.521	0.874
	NB	0.698	-0.471	-0.411	0.205	0.382	**-0.775**	**-0.762**	0.411	0.273	0.779
	NNge	**0.716**	-0.464	-0.432	0.304	0.394	-0.552	-0.543	0.359	0.277	0.753
	RandFor	**0.852**	-0.611	-0.539	0.322	0.463	-0.724	-0.708	0.475	0.343	0.901
	Ridor	**0.711**	-0.437	-0.391	0.216	0.355	-0.575	-0.558	0.473	0.398	0.761
	RIPPER	**0.717**	-0.542	-0.583	0.417	0.486	-0.673	-0.649	0.398	0.263	0.854
	US_NB	-0.656	0.451	0.374	-0.097	-0.310	**0.889**	**0.881**	-0.373	-0.187	0.859
	QBC_NB	0.440	-0.229	-0.169	0.083	0.222	**-0.521**	**-0.534**	0.265	0.201	0.500
	QBC_C4.5	**0.672**	-0.486	-0.358	0.362	0.542	-0.605	-0.597	0.357	0.322	0.726

overlap on a subset of the features, class skew, and the description length) are not as indicative of an instance being hard to classify. We can infer that, in general, class overlap is a principal contributor to instance hardness for the considered data sets whether considering ESLAs in general (IH_ind and IH_class) or for a specific learning algorithm. This can be seen by examining individual instances and their corresponding hardness measures. The hardness measures and instance hardness values for a sample of instances are provided in Table 3.6. The first instance is a clear example that exhibits class overlap and should be misclassified as indicated by the values of the hardness measures (i.e. high value for kDN, CLD is negative, etc.).

Table 3.6: The hardness measures and instance hardness values for an example set of instances.

#	data	id	kDN	DS	DCP	TDP	TDU	CL	CLD	MV	CB	IH_ind	IH_cla	US_NB	QBC_NB	QBC_C4.5
1	yeast	1470	0.98	0	0.25	12	9	0.28	-0.32	0.47	0.06	1	0.84	0.20	0.09	0.60
2	colon	56	0.46	1	0.98	4	3	1	1	0	0.15	0.84	0.69	1	0	1
3	ar1	84	0.37	0	0.33	5	2	0.92	0.84	0	0.59	0.91	0.71	1	0.74	0.55

One of the difficulties of identifying hard instances is that hardness may arise from several sources. For example, instances 2 and 3 in Table 3.6 have multiple possible reasons for why they are misclassified, but no hardness measure strongly indicates that it should be misclassified (i.e. the kDN values are less than 0.5, meaning that the instances agree with the majority of their neighbors). The last column "Lin" in Table 3.5 shows the correlation coefficients of a linear model of the hardness measures predicting the hardness measures. The instance hardness and hardness measures from the UCI and non-UCI data sets for each instance were compiled and linear regression was used to predict the hardness measures. Apart from US_NB and QBC_NB, a linear combination of the hardness measures results in a stronger correlation with instance hardness than any of the individual measures suggesting that there is no one measure that sufficiently captures the hardness of an instance.

Comparing the hardness measures, IH_class has the strongest correlation with the linear combination of the hardness measures and kDN. IH_class also has a strong correlation with CL and CLD. Only US_NB has a stronger correlation with CL and CLD than IH_class. These strong correlations with IH_class suggest that IH_class may be a good candidate for determining the hardness of an instance. The QBC methods are not as strongly correlated with any of the hardness measures as the other hardness measures. The active learning approaches select border points as the hardest instances, but do not indicate that the outlier instances are hard. We also observe that using classifier scores has a stronger correlation with the hardness measures than using the indicator function to calculate instance hardness. For all of the considered learning algorithms, calculating instance hardness with the classifier scores provide a stronger or equal correlation with the hardness measures than the

indicator function, suggesting that the classifier scores may provide a better indication of which instances are hard to classify. Also, for our examination of when ESLAs misclassify an instance, using an ensemble of learning algorithms to determine hardness has a stronger correlation with the hardness measures than a single learning algorithm.

The previous results suggest that, in general, class overlap causes instance hardness. However, in making this point, we realize that all data sets have different levels and causes of hardness. Table 3.7 shows the correlation between IH_class and the hardness measures for the instances in each data set. The column "DSH" refers to the *data set hardness* and is the average IH_class value for the instances in the data set. The harder data sets have a higher DSH value. The values in bold represent the hardness measures that have the strongest correlation with IH_class for the instances in the data set. The underlined values are the hardness measures with a correlation magnitude greater than 0.75. The values in Table 3.7 indicate that the hardness of the majority of the data sets is strongly correlated with the hardness measures that measure class overlap. There are a few data sets that have a strong correlation between IH_class and the measures that measure class skew (MV and CB). The most notable are the post-opPatient and zoo data sets. For those data sets, in addition to having a strong correlation with MV and CB, instance hardness is also strongly correlated with other hardness measures that measure class overlap.

It is not surprising that class overlap is observed as a principal contributor to instance hardness since outliers and border points, which exhibit class overlap, have been observed to be more difficult to classify correctly. However, instances that belong to a minority class have also been observed to be more difficult to classify correctly. This is confirmed as the coefficients for the class imbalance measures (MV and CB) in the linear regression models are statistically significant. Also, removing MV and CB from the linear model results in a weaker correlation. To what extent does class skew affect instance hardness? One of the core problems seen with class skew is that of data ambiguity, when multiple instances have the same feature values but different classes. In these cases, the instances that belong to

Table 3.7: The correlation of the hardness measures with IH_class for the instances in each data set. DSH is the average IH_class value of the instances in the data set

Dataset	DSH	kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB
abalone	0.815	**0.859**	-0.485	-0.287	-0.203	-0.085	-0.194	-0.141	0.323	0.323
adult-census	0.208	**0.898**	-0.722	-0.743	0.515	0.599	-0.737	-0.737	0.569	0.569
anneal.ORIG	0.108	0.658	-0.600	-0.349	0.326	0.416	**-0.689**	-0.687	0.425	0.425
AP_BreastUterus	0.056	0.563	**-0.615**	-0.279	0.094	0.323	-0.168	-0.168	0.535	0.535
ar1	0.126	0.684	**-0.814**	-0.388	0.726	0.395	0.450	0.450	0.450	0.450
arrhythmia	0.416	**0.845**	-0.769	-0.334	-0.655	-0.478	-0.404	-0.407	0.687	0.687
audiology	0.339	**0.836**	-0.783	-0.262	-0.011	-0.010	-0.681	-0.668	0.653	0.653
autos	0.337	**0.752**	-0.405	-0.082	0.064	0.033	-0.450	-0.447	0.086	0.086
badges2	0.003	0.216	**-0.563**	NA	NA	NA	-0.377	-0.377	**0.563**	**0.563**
balance-scale	0.259	**0.935**	-0.851	-0.578	0.792	0.749	-0.775	-0.797	0.466	0.466
balloons	0.072	0.746	-0.390	NA	0.035	0.035	**-0.931**	**-0.931**	-0.283	-0.283
breast-cancer	0.339	**0.877**	-0.632	-0.764	0.256	0.265	-0.490	-0.490	0.645	0.645
breast-w	0.059	0.627	**-0.809**	-0.610	0.746	0.804	-0.525	-0.533	0.598	0.598
bupa	0.396	**0.715**	-0.512	-0.358	0.395	0.404	0.191	0.191	0.389	0.389
carEval	0.140	0.924	-0.883	-0.561	0.886	0.873	**-0.937**	-0.912	0.705	0.705
chess	0.614	**0.606**	-0.245	-0.498	0.226	0.073	-0.310	-0.242	0.190	0.190
chess-KRVKP	0.087	0.608	-0.348	-0.317	0.725	0.726	**-0.860**	**-0.860**	0.126	0.126
cm1_req	0.324	0.628	-0.166	**-0.710**	0.548	NA	0.236	0.236	**0.710**	**0.710**
colic	0.223	**0.796**	-0.660	-0.644	0.325	0.296	-0.444	-0.443	0.282	0.282
colon	0.286	**0.620**	-0.528	0.391	-0.316	-0.342	-0.173	-0.173	0.495	0.495
contact-lenses	0.281	0.859	-0.880	-0.744	**0.907**	0.868	-0.877	-0.871	0.551	0.551
credit-a	0.197	**0.755**	-0.581	-0.743	0.367	0.574	-0.511	-0.511	0.360	0.360
credit-g	0.321	**0.887**	-0.595	-0.420	0.288	0.556	-0.620	-0.620	0.675	0.675
dermatology	0.099	**0.757**	-0.689	-0.444	0.494	0.457	-0.738	-0.744	0.526	0.526
desharnais	0.386	**0.877**	-0.714	-0.199	-0.024	-0.381	0.053	0.136	0.562	0.562
ecoli	0.229	**0.870**	-0.741	-0.234	-0.137	0.213	-0.831	-0.829	0.712	0.712
eucalyptus	0.467	**0.845**	-0.632	-0.506	0.380	0.352	-0.390	-0.381	0.298	0.298
eye_movements	0.492	**0.618**	-0.459	-0.115	0.198	0.254	-0.289	-0.265	-0.159	-0.159
glass	0.399	**0.816**	-0.606	0.045	0.136	0.144	-0.477	-0.474	-0.055	-0.055
heart-c	0.244	**0.816**	-0.756	-0.314	0.413	0.368	-0.740	-0.743	0.046	0.046
heart-h	0.237	**0.803**	-0.751	-0.563	0.297	-0.181	-0.675	-0.681	0.394	0.394
heart-statlog	0.248	**0.793**	-0.672	-0.187	0.496	0.497	-0.719	-0.719	0.095	0.095
hepatitis	0.222	0.825	**-0.888**	-0.011	0.550	0.635	-0.715	-0.715	0.615	0.615
hypothyroid	0.039	**0.655**	-0.281	-0.144	0.284	0.272	-0.617	-0.619	0.449	0.449
ionosphere	0.138	0.350	**-0.556**	0.044	0.182	0.183	-0.265	-0.262	0.119	0.119
iris	0.071	0.598	-0.579	-0.463	**0.870**	0.846	-0.626	-0.626	NA	NA
labor	0.177	**0.667**	-0.455	-0.514	0.336	-0.248	-0.553	-0.553	0.389	0.389
letter	0.347	0.752	**-0.790**	-0.244	0.518	0.588	-0.785	-0.769	0.040	0.040
lungCancer	0.537	**0.736**	-0.393	-0.251	0.283	0.203	-0.052	-0.057	-0.200	-0.200
lymphography	0.251	**0.776**	-0.759	-0.113	0.291	0.299	-0.546	-0.538	0.246	0.246
MagicTelescope	0.223	**0.821**	-0.624	-0.387	-0.116	0.088	-0.611	-0.612	0.435	0.435

Continued on next page

Table 3.7: **(cont.)** The correlation of the hardness measures with IH_class for the instances in each data set. DSH is the average IH_class value of the instances in the data set

Dataset	DSH	kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB
mushroom	0.016	0.085	-0.342	NA	**-0.565**	**-0.565**	-0.147	-0.140	0.451	0.451
nursery	0.110	0.569	-0.879	-0.384	0.896	**0.904**	-0.897	-0.892	0.717	0.717
ozone	0.071	0.550	-0.547	-0.217	0.224	**0.587**	-0.499	-0.503	0.290	0.290
pasture	0.295	**0.745**	-0.672	-0.133	0.569	0.667	-0.671	-0.673	NA	NA
pimaDiabetes	0.305	**0.895**	-0.696	-0.625	0.510	0.659	-0.622	-0.622	0.481	0.481
post-opPatient	0.425	0.785	-0.573	**-0.788**	0.145	NA	-0.079	-0.071	0.775	0.775
primary-tumor	0.678	**0.887**	-0.486	-0.754	0.260	0.311	-0.539	-0.489	0.476	0.476
segment	0.115	0.616	**-0.911**	-0.476	0.783	0.719	-0.636	-0.637	NA	NA
sick	0.037	0.591	0.002	0.407	0.137	0.331	**-0.772**	**-0.772**	0.390	0.390
sonar	0.274	0.652	-0.568	-0.330	0.303	0.264	**-0.715**	**-0.715**	0.027	0.027
soybean	0.181	**0.820**	-0.718	-0.221	0.275	0.259	-0.586	-0.591	0.138	0.138
spambase	0.133	0.583	**-0.655**	-0.273	0.343	0.261	-0.603	-0.616	0.037	0.037
splice	0.158	0.373	-0.549	-0.402	0.520	0.504	**-0.674**	-0.673	0.283	0.283
teachingAssist	0.495	**0.790**	-0.629	-0.388	0.055	-0.161	-0.219	-0.218	0.066	0.066
titanic	0.305	0.356	0.272	**-0.888**	0.030	0.030	-0.140	-0.140	0.256	0.256
trains	0.411	**0.756**	-0.362	-0.241	0.071	NA	-0.200	-0.200	NA	NA
vote	0.070	0.674	**-0.825**	-0.622	0.811	0.691	-0.645	-0.654	0.549	0.549
vowel	0.287	0.364	**-0.521**	-0.190	0.343	0.320	-0.443	-0.403	NA	NA
waveform-5000	0.268	**0.805**	-0.595	0.116	0.419	0.417	-0.651	-0.651	-0.184	-0.184
wine	0.079	**0.711**	-0.519	-0.242	-0.093	-0.368	-0.538	-0.539	-0.102	-0.102
yeast	0.523	**0.893**	-0.674	-0.342	0.081	0.285	-0.285	-0.222	-0.026	-0.026
zoo	0.134	**0.900**	-0.836	-0.392	0.690	0.684	-0.828	-0.821	0.830	0.830

the minority class will be misclassified. There are only 204 such instances, about 0.1% of all of the instances used in this study. We removed all of the ambiguous instances and then divided the instances into those that have a MV value of 0 (they belong to the majority class) and those that have a value greater than 0. This considers any instance that does not belong to the majority class as belonging to a minority class. There are 97,469 instances that belong to the majority class and 92,669 instances that do not. We observe that instances that belong to a minority class are harder to classify correctly than those that do not. The average IH_class value for the instances that belong to a majority class is 0.16 while the average instance hardness value for the instances not belonging to the majority class is 0.41. Table 3.8 compares the hardness measures for the instances that belong to a minority class and those that belong to the majority class. The last column (easy) gives the value for the

Table 3.8: Various statistics for the hardness measures for instances that belong to the majority class and those that do not. For the instances that belong to the minority class, the values for the measures indicate higher levels of class overlap.

	Minority Class				Majority Class				
	Min.	Mean	Max.	Std Dev	Min.	Mean	Max.	Std Dev	easy
DN	0	0.348	1	0.307	0	0.172	1	0.236	0
DS	0	0.179	1	0.286	0	0.363	1	0.410	1
DCP	0	0.786	1	0.300	0.002	0.909	1	0.170	1
TD_P	1	9.530	59.88	6.232	1	9.593	136.265	7.355	1
TD_U	0	7.315	29	4.809	0	5.555	29	4.526	1
CL	0	0.563	1	0.375	0	0.886	1	0.188	1
CLD	-1	0.297	1	0.602	-1	0.803	1	0.306	1
MV	1	0.308	0.910	0.278	0	0	0	0	1
CB	-0.471	-0.026	0.189	0.111	0	0.213	0.213	0.155	1
IH_class	0	0.410	0.999	0.269	0	0.163	0.994	0.200	0

hardness measures for the easiest instances (the instances that are always correctly classified). Not including MV and CB (which are biased since all of the instances that belong to the non-majority classes are separated from the majority class instances), all of the hardness measures except for pruned tree depth (TD_P) indicate that the instances that belong to a minority class are harder to classify correctly as well. Thus, we observe that class skew exacerbates the effects of the underlying causes for instance hardness. This coincides with Batista's conclusion that class skew alone does not hinder learning algorithm performance, but rather class skew magnifies the hardness already present in the instances [4]. For example, Table 3.9 gives the hardness measure values for two instances from the chess data set. The hardness measures are similar for each measure except the first instance (id 22037) belongs to the majority class while instance 26549 does not. The difference in the IH_ind value for the instances is considerable. The difference in IH_class values does not vary considerably since many of the class scores are similar to the hardness measures. This supports the fact that class skew exacerbates the effects of class overlap and also shows that IH_ind may be better able to incorporate the effects of class skew than IH_class. Given that class skew exacerbates the effects of class overlap on instance hardness, the expected instance hardness

Table 3.9: The hardness measures and instance hardness values for an example set of instances from the chess data set.

id	kDN	DS	DCP	TDP	TDU	CL	CLD	MV	CB	IH_ind	IH_cla	US_NB	QBC_NB	QBC_C4.5
22037	0.64	0.29	1	5	5	0.09	-0.36	0	0.11	0.24	0.41	0.33	0.00	0.40
26549	0.64	0.29	1	5	5	0.12	-0.36	0.52	0.02	0.49	0.39	0.39	0.00	0.32

for an instance is related to the class overlap (Equation 3.3) and class skew (Equation 3.4) of the instance:

$$\mathbb{E}[IH(\langle x_i, y_i \rangle)] \sim f(classOverlap(\langle x_i, y_i \rangle),\ classSkew(\langle x_i, y_i \rangle)).$$

The exact form of f is unknown at this stage. Additionally, other factors not discussed here may affect the hardness of an instance. Discovering the relationship between class overlap, class skew, and instance hardness, as well as identifying other sources of hardness, is left for future work.

3.6 Integrating Instance Hardness into the Learning Process

In this section we examine how to exploit instance hardness during the learning process to alleviate the effects of class overlap and instance hardness. Incorporating instance hardness into the learning process provides significant improvements in accuracy. Note that the improvement requires computing instance hardness for each instance. In the experiments, we opt to use IH_class instead of IH_ind as they are strongly correlated and IH_class produces slightly better results. We also ran the experiments calculating instance hardness with the same single learning algorithm that is inferring the model. This provides the opportunity to compare whether it is more appropriate to use a specific measure of instance hardness rather than a more general one. In addition, we ran the experiments using the active learning hardness measures. The active learning techniques are not designed to identify hard instances and using them as a hardness measure often resulted in poor results. In order to avoid a deluge of data, we do not show their results.

3.6.1 Informative Error

Informative error (IE) is based on the premise of knowing if an instance *should* be misclassified. We implement IE in multilayer perceptrons (MLPs) trained with backpropagation using instance hardness computed using 1) all of the learning algorithms in $\mathcal{L}$ (IE_{ESLA}) and 2) only using a MLP (IE_{MLP}). We use instance hardness to estimate if an instance should be misclassified. A common approach for classification problems with MLPs is to create one output node for every class value. If the data set has a class with three possible values, then three output nodes are created. The target output value for each node is either 1 if the instance belongs to that class or 0 if it does not. The error function of $target - output$ for each of the k output nodes can then be formulated as:

$$error(x) = \begin{cases} 1 - o_k & \text{if } t(x) = k_{class} \\ 0 - o_k & \text{otherwise} \end{cases}$$

where o_k is the output value for node k, $t(x)$ is the target value for instance x and k_{class} is the class represented by node k.

We modify the error function such that it subtracts the instance hardness value of an instance from the target value for the output node only.

$$error(x) = \begin{cases} 1 - IH(x, t(x)) - o_k & \text{if } t(x) = k_{class} \\ 0 - o_k & \text{otherwise} \end{cases}$$

The instance hardness value is only subtracted from the output node that corresponds with the target class value of an instance. If the instance hardness value were added to the output value for the output nodes that do not correspond with the target class value of an instance then this could potentially confuse the network as an instance is incorrect for one class value yet correct for all of the others. For example, if an instance has an instance hardness value of 1, then the errors would essentially tell the network that the target value is wrong whereas

all of the other classes are correct. Also, if an instance had an instance hardness value of 0.5, all output nodes would have the same target value and no information is gained. IE places more emphasis on the non-overlapping instances by reducing the weight of the error from instances with high instance hardness values.

Table 3.10 shows the results of using IE to train a MLP on 52 data sets (the data sets that did not have instance hardness greater than 0.5 were not used) compared against two filtering techniques (repeated edited nearest neighbor (RENN) [41] and fast local kernel noise reduction (FaLKNR) [34]) and two boosting methods (AdaBoost [15] and MultiBoost [44] using a MLP as the base algorithm). RENN repeatedly removes the instances that are misclassified by a 3-nearest neighbor classifier and has produced good results. FaLKNR removes any instances that disagree with the predicted class from a support vector machine trained on the neighborhood of the selected instance. The average accuracy, the number of times that the accuracy using IE_{MLP} is better, the same, or worse than the other methods, and the p-value calculated using the Wilcoxon signed-rank test are provided in the bottom three rows as a summary of the table. There are 14 data sets on which IE_{MLP} increases accuracy by more than 5%, indicated by an asterisk. On the lung cancer data set, accuracy increases by 21.9% and is 3 percentage points higher than the next best algorithm (FaLKNR). On the labor data set, IE_{MLP} increases accuracy by 10.5% and is 5 percentage points greater than the next best algorithm. On average, IE_{MLP} increases more than 3% in accuracy over the original and 2% over RENN. The increases in accuracy are statistically significant. In this case, IE_{MLP} is significantly better than IE_{ESLA}. Thus, in this case, using a specific bias from a learning algorithm is preferred. This is examined in more detail in the next section.

Although IE is described in the context of MLPs, it can also be applied to other learning algorithms that are incrementally updated based on an error value such as the class of non-closed form regression models (i.e., logistic regression and isotonic regression). Similar to informative error, instance hardness could be used to weight the instances prior to training

Table 3.10: Pairwise comparison of informative error with standard backpropagation, RENN, FaLKNR, AdaBoost, and MultiBoost. An asterisk indicates data sets on which IE_{MLP} improves accuracy more than 5%.

Dataset	Orig	RENN	FaLKNR	AdaBoost	MultiBoost	IE_{ESLA}	IE_{MLP}
abalone	26.24	27.80	28.78	26.24	26.24	27.84	**29.12**
adult-census	82.91	83.82	83.45	82.91	82.91	**85.22**	84.46
anneal.ORIG	98.78	98.33	97.55	98.89	98.89	**99.33**	96.82
arrhythmia	67.70	61.50	67.92	67.70	67.70	**71.68**	71.06
audiology	**83.19**	74.78	77.88	**83.19**	**83.19**	79.65	81.95
autos	80.00	76.10	68.29	79.51	78.05	78.05	**80.39**
balance-scale	90.72	89.45	90.72	92.64	**92.80**	90.40	91.71
breast-cancer*	64.69	74.48	73.08	70.28	69.93	**75.87**	74.34
breast-w	95.28	96.14	96.28	94.99	95.85	96.57	**96.62**
bupa	71.59	71.88	71.01	71.88	71.59	**72.75**	71.59
carEval	**99.54**	92.53	99.25	**99.54**	**99.54**	98.61	99.46
chess-KRvsKP	99.41	99.31	**99.47**	99.41	99.41	99.41	99.39
chess	62.25	56.94	62.18	65.75	**67.60**	51.49	61.31
colic*	80.43	83.70	85.33	80.98	82.07	85.33	**86.25**
contact-lenses*	70.83	**91.67**	83.33	70.83	70.83	**91.67**	87.50
credit-a	84.20	**87.97**	84.78	84.20	84.35	86.52	87.62
credit-g*	71.60	76.00	72.80	71.50	73.00	76.90	**78.22**
dermatology	96.17	96.45	97.81	96.17	96.17	97.54	**98.69**
ecoli	86.01	87.20	**87.50**	84.23	85.12	86.90	87.20
glass	67.76	70.09	68.22	71.50	67.29	70.56	**71.96**
heart-c*	80.86	82.51	79.54	77.56	79.87	84.16	**86.20**
heart-h	**85.03**	84.35	84.01	80.27	81.63	82.99	84.42
heart-statlog*	78.15	82.96	83.33	78.15	80.37	**85.93**	84.81
hepatitis*	80.00	86.45	82.58	79.35	79.35	87.10	**89.68**
hypothyroid	94.04	94.35	94.57	94.62	95.10	**95.52**	94.90
ionosphere	91.17	86.61	86.61	91.17	**91.74**	**91.74**	89.23
iris	**97.33**	96.67	96.67	96.67	96.67	96.67	96.80
labor*	85.96	85.97	91.23	85.96	85.96	**96.49**	91.93
letter	82.08	82.56	82.60	**88.35**	87.30	80.48	81.86
lungCancer*	37.50	50.00	56.25	37.50	37.50	59.38	**60.00**
lymphography	84.46	83.78	83.11	84.46	85.14	85.14	**85.95**
MagicTelescope	85.87	85.42	85.19	85.90	86.25	85.53	**86.36**
nursery	99.73	97.45	98.89	**99.97**	**99.97**	99.87	99.92
ozone	96.41	96.81	97.12	96.41	**99.73**	96.96	97.41
pimaDiabetes	75.39	76.69	75.91	75.26	75.13	77.08	**78.07**
post-opPatient*	55.56	71.11	71.11	52.22	54.44	66.67	**72.22**
primary-tumor*	38.35	47.20	46.02	43.07	43.07	49.56	**51.27**
segment	96.06	95.97	96.41	96.06	95.93	96.10	**96.84**
sick	97.27	97.27	96.85	97.27	97.11	**97.51**	97.42

Continued on next page

Table 3.10: **(cont.)**Pairwise comparison of informative error with standard backpropagation, RENN, FaLKNR, AdaBoost, and MultiBoost. An asterisk indicates data sets on which IE_{MLP} improves accuracy more than 5%.

Dataset	Orig	RENN	FaLKNR	AdaBoost	MultiBoost	IE_{ESLA}	IE_{MLP}
sonar*	82.21	84.13	85.10	83.65	83.17	87.98	**88.17**
soybean	93.41	92.97	**95.17**	93.41	93.41	**95.17**	94.73
spambase	91.44	91.05	92.18	91.44	91.05	92.24	**92.65**
splice	95.96	95.24	95.36	95.96	95.96	**96.80**	96.78
teachingAssistant*	58.94	61.59	63.58	58.94	58.94	64.90	**65.56**
titanic	78.46	**79.06**	**79.06**	78.60	78.96	78.87	**79.06**
trains*	70.00	80.00	50.00	70.00	70.00	**90.00**	**90.00**
vote	94.71	94.71	**96.55**	94.48	94.48	95.17	95.95
vowel	92.73	93.84	93.64	96.26	**96.67**	91.62	91.94
waveform-5000	83.56	84.93	86.30	83.36	83.50	85.66	**86.60**
wine	97.19	96.63	96.63	97.19	97.19	97.75	**98.65**
yeast	59.43	59.03	60.31	59.43	59.10	59.97	**60.77**
zoo	**96.04**	94.06	94.06	**96.04**	**96.04**	**96.04**	95.84
Average	81.05	82.45	82.14	81.37	81.60	84.03	**84.57**
better-same-worse	40-1-11	43-0-9	43-1-8	40-0-12	38-1-13	36-1-15	
p-Value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	0.003	

a model. This weight could then be used in a number of learning algorithms such as nearest-neighbor or naïve Bayes algorithms.

3.6.2 Filtering the data set

A simple idea to handle hard instances and reduce overlap is to filter or remove them from a data set prior to training. The idea of filtering is to remove the instances that are suspected outliers or noise and thus increase class separation [37]. We use the IH_class values to determine which instances to filter from the data sets. We compare the results to those by RENN and the majority and consensus filters proposed by Brodley and Friedl [10]. The majority and consensus filters remove an instance if it is misclassified respectively by the majority of, or all, three learning algorithms (C4.5, IB1, and thermal linear machine [11]). When using the instance hardness values, we use the classifier scores from the five folds of the nine learning algorithms as our ensemble and remove any instances with an IH_class value greater than a set threshold. We set the threshold at 0.5 (IH_0.5), 0.7 (IH_0.7) and

0.9 (IH_0.9). We also compare using each learning algorithm to filter the instances and as the learning algorithm (IH_LA). For example, IH_LA for MLP uses a MLP to identify which instances to filter prior to training a MLP. Each filtering technique was used on a set of 52 data sets evaluated using five by ten-fold cross-validation on the nine learning algorithms. Testing is done on all of the data, including the instances that were removed.

For the nine learning algorithms, Table 3.11 shows the average accuracy, pairwise comparison of the accuracies, and p-values from the Wilcoxon signed-rank statistical significance test comparing the filtering method to the original accuracy. Only the averages are displayed to avoid the overload of tables and much of the aggregate information is present in the pairwise comparison of the algorithms (number of times that a learning algorithm increases-stays the same-decreases the accuracy) and the p-value from the Wilcoxon signed rank significance test. Filtering significantly increases classification accuracy for most of the filtering techniques and learning algorithms. IH 0.7 achieves the greatest increase in accuracy, being slightly better than the majority filter. One of the advantages of using instance hardness is that various thresholds can be used to filter the instances. However, we note that there is not one filtering approach that is best for all learning algorithms and data sets (as indicated by the counts). For filtering, using the same learning algorithm to infer the model and to determine which instances to filter is only better than using all of the learning algorithms in $\mathcal{L}$ for C4.5 and 5-NN.

To examine the variability of each data set and learning algorithm combination, we examine an adaptive filtering approach that generates a set of learning algorithms to calculate instance hardness for a specific data set/learning algorithm combination [38]. We call the set of learning algorithms used to calculate instance hardness a *filter set*. The adaptive approach discovers the filter set through a greedy search of $\mathcal{L}$. The adaptive approach iteratively adds a learning algorithm from $\mathcal{L}$ to a filter set by selecting the learning algorithm that produces the highest classification accuracy when added to the filter set, as shown in Algorithm 1. A constant threshold value is set to filter instances in $runLA(F)$ for all iterations. We

Table 3.11: The average accuracy values for the nine learning algorithms comparing filtering techniques against not filtering the data (Orig). "count" gives the number of times that a filtering algorithm improves, maintains, or reduces classification accuracy. On average, filtering the data sets significantly improves the classification accuracy.

Algorithm	Orig	IH_0.5	IH_0.7	IH_0.9	RENN	Majority	Consen	IH_LA
MLP	81.05	83.12	**83.58**	81.86	82.45	82.52	81.92	82.95
count		35-1-16	37-0-15	30-2-20	28-3-21	25-0-27	24-0-28	36-0-16
p-value		**0.001**	**< 0.001**	**0.025**	**0.047**	0.151	0.246	**0.002**
C4dot5	80.11	80.23	81.46	80.53	80.51	81.48	81.11	**82.06**
count		32-0-20	41-2-9	25-2-25	25-4-23	32-3-17	36-3-13	38-2-12
p-value		0.054	**< 0.001**	0.226	0.122	**0.002**	**0.001**	**< 0.001**5
5-NN	79.03	81.42	82.14	80.21	82.28	81.62	81.05	**82.34**
count		39-1-12	38-3-11	37-4-11	39-2-11	36-5-11	32-9-11	41-2-9
p-value		**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**
LWL	69.36	71.05	69.65	69.80	69.75	**70.80**	70.32	67.69
count		32-11-9	26-12-14	23-13-16	24-14-14	33-8-11	22-18-12	21-13-18
p-value		**0.002**	0.127	0.103	0.091	**0.002**	**0.009**	0.634
NB	75.68	77.79	77.22	**76.50**	76.17	77.52	77.05	75.04
count		37-1-14	36-0-16	32-3-17	27-7-18	35-4-13	31-8-13	21-1-30
p-value		**< 0.001**	**< 0.001**	**0.008**	0.083	**0.001**	**< 0.001**	0.871
NNge	79.45	81.69	**82.16**	80.05	81.16	81.40	81.10	81.57
count		34-0-18	41-0-11	29-2-21	30-3-19	31-4-17	29-7-16	36-0-16
p-value		**< 0.001**	**< 0.001**	0.070	**0.040**	**0.006**	**0.003**	**0.001**
RandFor	81.59	82.52	**83.07**	81.83	82.44	82.37	81.97	82.80
count		28-3-21	36-0-16	29-1-22	26-6-20	24-5-23	25-8-19	27-4-21
p-value		**0.00**9	**0.001**	0.081	**0.045**	0.051	**0.026**	**0.031**
Ridor	78.09	**79.29**	79.22	78.45	78.16	78.87	78.94	78.65
count		36-3-13	36-2-14	25-1-26	27-2-23	34-3-15	29-7-16	32-3-17
p-value		**< 0.001**	**0.001**	0.173	0.419	**0.003**	**0.021**	**0.013**
RIPPER	77.83	79.21	79.16	78.44	77.52	**79.79**	78.89	78.83
count		37-1-14	38-0-14	32-1-19	26-4-22	36-1-15	32-7-13	37-2-13
p-value		**< 0.001**	**< 0.001**5	**0.019**	0.464	**0.001**	**0.001**	**0.003**
Average	78.02	79.59	**79.74**	78.63	78.94	79.60	79.15	79.10
count		42-0-10	45-0-7	38-0-14	33-1-18	38-0-14	39-0-13	38-0-14
p-value		**< 0.001**	**< 0.001**	**< 0.001**	**0.003**	**< 0.001**	**< 0.001**	**0.001**

Algorithm 1 Adaptively constructing a filter set.

Let F be the filter set used for filtering and $\mathcal{L}$ be the set of candidate learning algorithms for F.
Initialize F to the empty set: $F \leftarrow \{\}$
Initialize the current accuracy to the accuracy from an empty filter set: $currAcc \leftarrow runLA(\{\})$. $runLA(F)$ returns the accuracy from a learning algorithm trained on a data set filtered with F.
while $\mathcal{L} \neq \{\}$ **do**
  $bestAcc \leftarrow currAcc$;
  $bestLA \leftarrow null$;
  for all $g \in \mathcal{L}$ **do**
    $tempF \leftarrow F + g$;
    $acc \leftarrow runLA(tempF)$;
    if $acc > bestAcc$ **then**
      $bestAcc \leftarrow acc$;
      $bestLA \leftarrow g$;
    end if
  end for
  if $bestAcc > currAcc$ **then**
    $\mathcal{L} \leftarrow \mathcal{L} - bestLA$;
    $F \leftarrow F + bestLA$;
    $currAcc \leftarrow bestAcc$;
  else
    break;
  end if
end while

examine thresholds of 0.5, 0.7, and 0.9. The baseline accuracy for the greedy approach is the accuracy of the learning algorithm without filtering. The search stops once adding one of the remaining learning algorithms to the filter set does not increase accuracy. The running time for the adaptive approach is $O(N^2)$ where N is the number of learning algorithms to search over. The significant improvement in accuracy makes the increase in computational time reasonable in most cases.

Table 3.12 gives the results for adaptively filtering for a specific data set/learning algorithm combination. The adaptive approach significantly increases the classification accuracy over IH_0.7 for all of the learning algorithms and thresholds. The accuracy increases for at least 85% of the data sets regardless of which learning algorithm is being used for classifica-

tion. A_0.9 achieves the highest classification accuracy for the adaptive approach. Interestingly, there is no one particular learning algorithm that is always included in a filter set for a particular learning algorithm. The frequency for how often a learning algorithm is included in a filter set for each learning algorithm and an aggregate count (overall) is given in Table 3.13. MLPs and random forests are included in more than 50% of the constructed filter sets while RIPPER and NB are included in less than 2% of the filter sets. The remaining learning algorithms are used in a filter set between 13% and 23% of the time. It is interesting that some of the learning algorithms include a particular learning algorithm in the filter set for most of the data sets while other learning algorithms never or rarely include it. For example, MLP is always included in the filter set for NB, yet never for 5-NN. Also, only the MLP and 5-NN learning algorithms frequently include themselves in the filter set. Thus, hardness for a learning algorithm is often better detected using a different learning algorithm.

3.7 Data Set-level Analysis

Our work has focused on hardness at the instance-level. However, prior work has been done that examines what causes hardness at the data set level. The hardness measures and hardness measures can be averaged together to measure hardness at the data set level. The averaged hardness measures can provide insight into a data set's characteristics and possibly provide direction into which methods are the most appropriate for the data set. Previous studies have primarily looked at only binary classification problems. We compare instance hardness at the data set level with other data set complexity measures. We use a set of complexity measures by Ho and Basu [18] (implemented with DCoL [27]). In this study we do not limit our examination to two-class problems. Hence, we do not use the measurements from Ho and Basu that are only for two-class problems. Ho and Basu's complexity measures that were used are shown in Table 3.14. Some of the original measures were adapted to handle multi-class problems [27].

Table 3.12: The average accuracy values for nine learning algorithms comparing the adaptive filtering approach against IH 0.7. "count" gives the number of times that a filtering algorithm improves, maintains, or reduces classification accuracy. The adaptive filtering approach significantly increases classification accuracy.

Algorithm	IH_0.7	A_0.5	A_0.7	A_0.9
MLP	83.583	86.302	86.863	87.997
counts		51-1-0	52-0-0	52-0-0
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
C4.5	81.459	84.854	86.023	86.875
counts		49-1-2	51-1-0	52-0-0
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
5-NN	82.135	85.953	87.189	89.162
counts		49-3-0	51-1-0	52-0-0
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
LWL	69.649	74.382	74.043	74.048
counts		46-4-2	43-7-2	43-7-2
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
NB	77.220	80.368	80.637	80.345
counts		49-2-1	50-1-1	50-1-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
NNge	82.158	85.876	87.145	88.892
counts		49-2-1	50-1-1	50-1-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
RandForest	83.065	86.306	87.353	89.506
counts		49-2-1	51-0-1	51-0-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
Ridor	77.699	81.802	82.509	83.494
counts		50-1-1	51-0-1	51-0-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
RIPPER	79.163	84.418	85.197	86.197
counts		49-1-2	50-1-1	51-0-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
Average	79.742	83.535	84.280	85.342
counts		51-0-1	51-0-1	51-0-1
p-value		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**

Table 3.13: The frequency of selecting a learning algorithm when adaptively constructing a filter set. Each row gives the percentage of cases that the learning algorithm was included in the filter set for the learning algorithm in the column.

	Overall	MLP	C4.5	5-NN	LWL	NB	Nnge	RandF	Ridor	RIP
MLP	51.59	86.67	16.67	0	53.33	100	13.33	26.67	80.00	93.33
C4.5	17.46	13.33	16.67	6.67	26.67	0	13.33	20.00	26.67	33.33
5-NN	23.81	6.67	0	86.67	6.67	0	26.67	26.67	20.00	26.67
LWL	15.87	0	0	40.00	0	66.67	0	6.67	20.00	0
NB	1.59	0	0	0	13.33	0	0	0	0	0
NNge	18.25	26.67	16.67	20.00	13.33	0	46.67	93.33	13.33	0
RandF	55.56	26.67	100	80.00	6.67	0	80.00	0	86.67	53.33
Ridor	13.49	13.33	50.00	0	6.67	53.33	6.67	6.67	6.67	0
RIP	0.79	0	0	0	0	6.67	0	0	0	0

We first compare our measures to those used by Ho and Basu [18]. The matrix of Spearman correlation coefficients comparing the hardness measures against those measures used by Ho and Basu are shown in Table 3.15. The measures were normalized by subtracting the mean and dividing by the standard deviation. The values in bold represent correlations with a magnitude greater than 0.75. Ony N1 and N3 are strongly correlated with kDN, CL, and CLD. N1 is the percentage of instances with at least one nearest neighbor of a different class. N3 is the leave-one-out error of the one-nearest neighbor classifier. Both N1 and N3 are similar and can be categorized as measuring class separability. N1, N3, kDN, CL, and CLD measure class overlap using all of the features in the data set.

We examined each hardness measure and complexity measure individually to determine how well it predicts data set hardness (the average instance hardness of the instances in the data set). The Spearman correlation coefficient for the hardness measures and the measures from Ho and Basu with data set hardness are shown in Table 3.16. kDN, CL, CLD, N1, and N3 all have a correlation coefficient greater than 0.8. Recall that kDN, CL, CLD are strongly correlated with N1 and N3. Despite diversity in the measures, only these few are strongly correlated with data set hardness and they measure class overlap.

We also apply linear regression to evaluate data set hardness as a combination of the hardness measures and the measures from Ho and Basu. The correlation coefficients are

Table 3.14: List of complexity measures from Ho and Basu [18].

F2:	**Volume of overlap region**:The overlap of the per-class bounding boxes calculated for each attribute by normalizing the difference of the maximum and minimum values from each class.
F3:	**Max individual feature efficiency**: For all of the features, the maximum ratio of the number of instances not in the overlapping region to the total number of instances.
F4:	**Collective feature efficiency**: F3 only return the ratio for the attribute that maximizes the ratio. F4 is a measure for all of the attributes.
N1:	**Fraction of points on class boundary**: The fraction of instances in a data set that are connected to their nearest neighbors that have a different class in a spanning tree.
N2:	**Ratio of ave intra/inter class NN dist**: The average distance to the nearest intra-class neighbors divided by the average distance to the nearest inter-class neighbors.
N3:	**Error rate of 1NN classifier**: Leave-one-out error estimate of 1NN.
T1:	**Fraction of maximum covering spheres**: The normalized count of the number of clusters of instances containing a single class
T2:	**Ave number of points per dimension**: Compares the number of instances to the number of features.

shown in the column "Lin" in Table 3.16. For the linear model of the hardness measures, only kDN is statistically significant for the hardness measures. For Ho and Basu's complexity measures, only N1 is statistically significant. The correlation of data set hardness with the linear models of the hardness measures and the measures from Ho and Basu are weaker than the correlation of data set hardness with an individual measure. When using both sets of measures, the resulting correlation coefficient is 0.896 with none of the measures being statistically significant. The linear model also has a weaker correlation coefficient than only using kDN.

Based on correlation from a linear regression model, our aggregate hardness measures are competitive with those from Ho and Basu. When the hardness measures are used in combination with those from Ho and Basu, a slightly stronger correlation is achieved. This is somewhat expected as there are many underlying and misunderstood factors that affect

Table 3.15: Spearman correlation matrix comparing the hardness measures against the complexity measures from Ho and Basu. The strong correlation (bolded values) indicate that there is some overlap between our measures and those by Ho and Basu.

	F2	F3	F4	N1	N2	N3	T1	T2
DN	0.433	0.112	0.237	**0.908**	0.696	**0.867**	0.298	-0.142
DS	-0.550	0.063	0.011	-0.523	-0.427	-0.464	-0.123	-0.445
DCP	-0.542	0.086	0.107	-0.661	-0.456	-0.676	-0.147	-0.033
TD_P	0.403	0.014	0.079	0.283	0.235	0.233	-0.040	0.340
TD_U	0.306	0.121	0.233	0.336	0.221	0.240	-0.071	0.338
CL	-0.490	0.008	-0.186	**-0.797**	-0.644	**-0.763**	-0.246	-0.078
CLD	-0.463	-0.035	-0.162	**-0.805**	-0.649	**-0.775**	-0.272	-0.063
MV	0.424	0.336	0.162	0.307	0.304	0.318	0.163	-0.148
CB	0.148	0.201	0.008	-0.034	0.062	0.015	0.065	-0.027

Table 3.16: The Spearman correlation coefficients for each hardness measure and Ho and Basu's complexity measures relating to data set hardness. The measures that measure class overlap have a strong correlation with data set hardness.

kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB	Lin
0.901	-0.561	-0.758	0.427	0.354	**-0.864**	**-0.868**	0.313	0.088	**0.882**

F2	F3	F4	N1	N2	N3	T1	T2	Lin
0.455	0.078	0.190	**0.860**	0.675	**0.828**	0.222	-0.127	**0.844**

complexity. By measuring the complexity from many different angles, more perspective can be found.

The averaged hardness measures at the data set level provide an indication of the source of hardness and could further indicate which learning algorithms and/or methods for integrating instance hardness into the learning process are the most appropriate to use for a particular data set. A cursory examination of the correlation of the hardness measures with instance hardness at the data set level (Table 3.16) does not reveal an obvious connection. Further in depth analysis is left for future work.

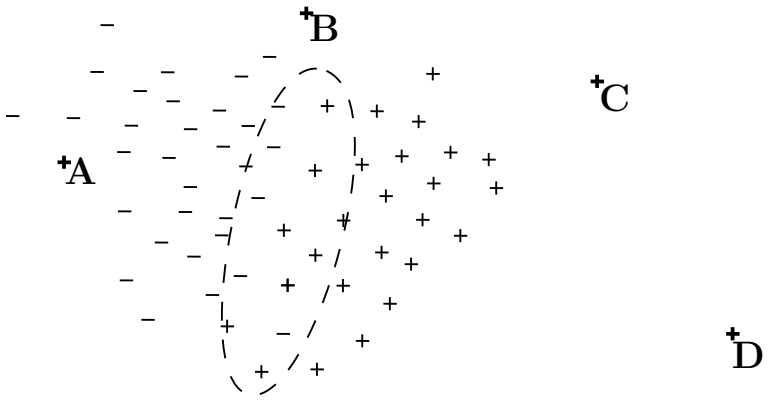

Figure 3.4: Hypothetical 2-dimensional data set.

3.8 Related Work

There are a number of methods and approaches that can be used to identify instances that are hard to classify correctly. In this section we review some previous work for identifying hard instances. Fundamentally, instances that are hard to classify correctly are those for which a learning algorithm has a low probability of predicting the correct class label after having been trained on a training set. To compare the related works with instance hardness we reference the hypothetical data set in Figure 3.2. For convenience, we reproduce Figure 3.2 in Figure 3.4. We also compare instance hardness (IH_ind and IH_class) with related works in Table 3.17 on a subset of the examined instances. We will refer to Table 3.17 throughout this section.

Machine learning research has observed that data sets are often noisy and contain outliers, and that noise and outliers are harder to classify correctly. Although we do not explicitly search for outliers, outliers and noisy instances will constitute a subset of the hard instances. Much work has been put forth to identify outliers and noise. Discovering outliers is important in anomaly detection where an outlier may represent an important instance. For example, an outlier in a database of credit card transactions may represent a fraudulent transaction. Anomaly detection is generally unsupervised and looks at the data set as a whole. One of the difficulties with outlier detection is that there is no agreed-upon definition of what

Table 3.17: Comparison summary of the methods that identify hard instances.

data		IH		Classifier Scores									Active Learn			Outlier Detection			
set	id	ind	class	MLP	C4.5	IB5	LWL	NB	NNg	Rand	Rid	RIP	USN	QN	QC	LOP	Maj	Con	REN
ecoli	263	1	0.85	1.00	0.78	1	0.91	1.00	1	0.92	1	0.96	0.99	0	0.72	0.26	yes	no	yes
ta	128	1	0.82	0.76	0.90	0.99	0.78	0.95	1	0.98	1	0.85	0.63	0.14	0.52	0	yes	no	yes
abal	4014	1	0.86	0.94	1	1	0.91	1.00	1	1	1	0.87	0.04	0.46	0.47	0.04	yes	yes	no
yeast	327	1	0.86	0.99	0.99	1	0.87	0.95	1	0.88	1	0.88	0.63	0.17	0.42	0.22	yes	yes	no
abal	720	1	0.89	1.00	1	1	1	1	1	1	1	1	0.99	0.08	0.32	1	yes	yes	no
spam	3913	0.07	0.09	0.59	0	0	0.20	0	0	0.02	0	0.05	1	0	0.15	1	no	no	no
yeast	6	0.76	0.75	0.91	0.86	1	0.61	0.95	1	0.88	0.40	0.89	0.80	0.09	0.47	1	yes	yes	yes
adult	30676	0.04	0.05	0.05	0.02	0	0.06	0.00	0.40	0.11	0	0.13	1	0	0	1	no	no	no
adult	30833	0	0.00	0.11	0.08	0	0.07	0.03	0	0	0	0.13	0.94	0	0	1	no	no	no
wave	3524	0.58	0.53	0.66	0.98	1	0.81	0.19	0	0.58	0.60	0.64	0.60	0	0.54	0.34	yes	yes	yes
chess	8691	0.53	0.53	0.82	0.72	0.58	0.87	0.81	0	0.46	0.40	0.48	0.04	0.57	0.38	0	yes	no	no
annea	715	0.53	0.56	0.02	0.64	1	0.56	0.67	0.80	0.76	0.40	0.16	0.18	0.68	0.73	0.01	yes	yes	yes
arrhy	69	0.51	0.57	0.92	0.45	1	0.43	0	1	0.76	0	0.22	1	0.95	0.42	0.18	yes	yes	yes

constitutes an outlier or noise. Thus, a variety of different outlier detection methods exist, such as statistical methods [3], distance-based methods [20], and density-based methods [7]. Anomaly detection methods identify anomalous instances as those that lie outside the group(s) of the majority of the other instances in the data set. In the hypothetical two-dimensional data set shown in Figure 3.4, instances C and D would be identified as anomalous but not instances A and B.

Most anomaly detection methods do not have a continuous output and are not supervised. One anomaly detection method that outputs continuous values is local outlier factor. Local outlier factor (LOF) [7] suggests that each instance has a degree of "outlierness" rather than a binary labeling. LOF seeks to overcome the problem facing most anomaly detection methods–that the sub-spaces within many data sets have different densities. LOF considers relative density rather than the global density of the data set. Instances with a LOF value of 1 or less are not outliers. The values produced by LOF are somewhat hard to interpret as there is no upper bound or any value that indicates when a LOF value represents an outlier. For one data set, an LOF value of 1.1 may represent an outlier while a value of 2 could represent an outlier in another data set.

There are a number of approaches that aim at overcoming the uninterpretability of LOF [22]. One approach is local outlier probability (LoOP) [21]. LoOP builds on LOF with statistical methods to produce a probability that an instance is a local density outlier. This allows the values to be compared across data sets. There are two major assumptions that LoOP makes: 1) that the k-nearest neighbors of an instance p are centered around p and 2) that the distances behave like the positive leg of a normal distribution. Despite being more interpretable, LoOP often does not identify hard instances as outliers and identifies easy instances as outliers as shown in Table 3.17 (LOP).

Filtering, or removing instances prior to training, is another approach that seeks to identify mislabeled and/or noisy instances with the intent of improving an inferred model of the data. Unlike anomaly detection, filtering is often supervised, removing instances that are misclassified by a learning algorithm. In Figure 3.4, filtering would likely identify instances A, B, and some of the border points as hard to classify. A popular approach to outlier detection is repeated-edited nearest-neighbor (RENN) [41] which repeatedly removes the instances that are misclassified by a 3-nearest neighbor classifier and has produced good results. Brodley and Friedl [10] expanded this idea by removing the instances that were misclassified by all or the majority of the learning algorithms in an ensemble of three learning algorithms. These methods do not output a continuous value but they do take into account the class label. As shown in Table 3.17, these methods (maj, con, and REN) do not often identify easy instances as outliers, but they may not identify hard instances as outliers.

Some learning algorithms produce probabilistic output, such as naïve Bayes, Bayes nets, and Gaussian processes. The output from probabilistic algorithms could naturally answer the question of which instances are hard to classify correctly. However, there are often assumptions that are made that are not true of the data distribution (i.e. the attributes are independent or the data is normally distributed). Many other machine learning algorithms do not produce a probabilistic output. In those cases, the probabilities can be approximated by normalizing the output values or using some heuristic to produce pseudo probabilistic

values. The posterior classifier probabilities from the learning algorithms in $\mathcal{L}$ for a subset of the instances are provided in Table 3.17. The posterior classifier probabilities provide a good approximation for instance hardness, but, as discovered in Section 3.5, they have a lower correlation with the hardness measures. This is apparent when examining instances that have an instance hardness measure around 0.5 (the last four instances in Table 3.17).

Probabilistic outputs from a classifier are important when the outputs are combined with other sources of information for making decisions, such as the outputs from other classifiers. Probabilistic outputs are often not well calibrated, such as the output from naïve Bayes [13]. As such, a number of methods have been proposed to calibrate classifier scores [5, 29, 46, 47]. For binary classification problems, the calibration is usually done by training the learning algorithm to get the classifier scores $s(x)$ and then mapping these scores into a probability estimate $\hat{P}(y|x)$ by learning a mapping function. Platt [29] suggests finding the parameters A and B for a sigmoid function of the form $\hat{P}(y|x) = \frac{1}{1+e^{As(x)+B}}$ to map the classifier scores $s(x)$ to the probability estimates minimizing the log-likelihood of the data. Multi-class classification problems are broken down into binary classification problems such as 1 vs 1 or 1 vs all. 1 vs 1 creates a classifier for each pair of classes. 1 vs all creates a classifier that discriminates between the instances of a particular class and all the instances that have a different class. The calibrated probabilities from the binary classification problems are then recombined back together. Classifier scores are supervised and produce continuous outputs for identifying hard instances. In Figure 3.4, instances A, B, and the border points would be identified as being hard to classify.

Active learning [35] seeks to find the most informative instances in a data set. Active learning assumes that there is a set of labeled instances and an abundance of unlabeled training data and that labeling the data is expensive, thus, the most informative instances should be labeled first. In active learning, a learning algorithm chooses which instances to use for training. Active learning assigns unlabeled instances a degree of how informative they may be to a learning algorithm by optimizing a given criterion. This informative measure

could be used as a means of identifying hard instances. For example, uncertainty sampling [24] selects an unlabeled instance x^* whose labeling the learning algorithm is least certain about:

$$x^* = \underset{x}{\operatorname{argmax}}\, 1 - p(\hat{y}|x)$$

where $\hat{y}$ is the class label with the highest posterior probability for the learning algorithm. Other methods, such a query-by-committee [16, 36] and a Support Vector Machine method by Tong and Koller [42], seek to reduce the size of the version space[6] [26]. Query-by-committee uses a committee of models trained on the labeled instances and selects the instances that the committee disagrees about most. Thus, active learning identifies the border points as being hard to classify. Table 3.17 shows that active learning scores vary widely for the same instances. Active learning scores do not have a high correlation with instance hardness.

Clearly, none of the previous work was designed to better understand why instances are misclassified as is the case with instance hardness. For example, filtering aims at removing mislabeled instances from a data set, and the classifier scores are for applications where a confidence on a prediction is required. Incorporating the ideas of previous work, instance hardness provides a framework for identifying which instances are hard to classify and understanding why they are hard to classify.

3.9 Conclusions and Future Work

In this paper we examined why instances are misclassified and how to characterize them. We presented instance hardness to empirically discover which instances are hard to classify correctly. We also presented a set of hardness measures to characterize why some instances are difficult to classify correctly. We used a broad set of data sets and learning algorithms and examined hardness at the instance level. We found that class overlap is a principal contributor to instance hardness and data set hardness. Class skew has been observed to increase instance hardness. We found that class skew alone does not cause instances to be

[6] Version space is the subset of parameters that correctly classifies the labeled examples.

misclassified. Rather, class skew exacerbates the other characteristics, such as class overlap, that cause an instance to be misclassified. Continued study of instance hardness should lead to additional insights regarding data complexity.

Being able to measure instance hardness and complexity has important ramifications for future machine learning and meta-learning research. We briefly examined integrating instance hardness into the learning process by filtering the data sets prior to training and using informative error. In each case, integrating into the learning process the knowledge of which instances are hard to classify correctly resulted in a significant increase in classification accuracy. These techniques show that integrating into the learning process the knowledge about which instances are hard can increase generalization accuracy. Future work includes understanding the circumstances and situations which are most appropriate for each technique. There is no one technique for identifying hard instances that is best for all data sets as demonstrated with the adaptive filter sets.

Calculating instance hardness and the hardness measures can be a computationally expensive procedure requiring the computation of N learning algorithms. However, the instance hardness values only need to be computed once and they can be used in a wide variety or applications as was shown in Section 3.6. The hardness measures need to be calculated only once as well. For many data sets, this additional computation complexity is acceptable. For massive data sets, though, the additional computational complexity can be a significant concern. In this case, the set of learning algorithms used to calculate instance and the hardness measures can be altered to those that better handle massive data sets. Also, we showed that there is no specific set of learning algorithms that is best for all data sets and learning algorithms. Using the same learning algorithm to calculate instance hardness and to infer the model of the data does not always result in the most accurate model.

Being able to better analyze data would allow a practitioner to select an algorithm more suited to their purposes. Also, the evaluation of a learning algorithm could be enhanced by knowing which instances are hard and, with a high likelihood, should be misclassified. This

could lead to a better stopping criterion. We expect that the exploration of instance hardness and data complexity may lead to more in depth investigation and applications in new areas of machine learning and data mining. Instance hardness and the hardness measures could be used in combination with techniques from active learning to determine a subset of the most important instances from a data set. Future work also includes work in meta-learning. For example, the hardness measures could be used to estimate the performance of a learning algorithm on a data set.

References

[1] Naoki Abe and Hiroshi Mamitsuka. Query learning strategies using boosting and bagging. In *Proceedings of the 15th International Conference on Machine Learning*, pages 1–9, 1998. ISBN 1-55860-556-8.

[2] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th international conference on Knowledge discovery and data mining*, pages 504–509, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

[3] Vic Barnett and Toby Lewis. *Outliers in statistical data.* John Wiley & Sons Ltd., 2nd edition edition, 1978.

[4] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations Newsletter*, 6(1):20–29, 2004. ISSN 1931-0145.

[5] Paul N. Bennett. Assessing the calibration of naive bayes posterior estimates. Technical Report CMU-CS-00-155, Carnegie Mellon University, 2000.

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

[7] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. *SIGMOD Record*, 29(2):93–104, June 2000. ISSN 0163-5808.

[8] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neuro-computing: Algorithms, Architectures and Applications*, pages 227–236. Springer, 1989.

[9] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, 2002. ISSN 1384-5810.

[10] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[11] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[12] Ido Dagan and Sean P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the 12th International Conference on Machine Learning*, pages 150–157, 1995.

[13] Pedro Domingos and Michael J. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In Lorenza Saitta, editor, *ICML*, pages 105–112. Morgan Kaufmann, 1996. ISBN 1-55860-419-7.

[14] A. Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

[15] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[16] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Information, prediction, and query by committee. In *Advances in Neural Information Processing Systems (NIPS)*, pages 483–490, 1992.

[17] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[18] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:289–300, March 2002.

[19] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179, 1995.

[20] Edwin M. Knorr and Raymond T. Ng. Finding intensional knowledge of distance-based outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 211–222, 1999.

[21] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1649–1652, 2009.

[22] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *Proceedings of the 11th SIAM International Conference on Data Mining*, pages 13–24, 2011.

[23] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[24] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12, 1994. ISBN 0-387-19889-X.

[25] Ester Bernadó Mansilla and Tin Kam Ho. On classifier domains of competence. In *ICPR (1)*, pages 136–139, 2004.

[26] Tom M. Mitchell. Generalization as search. *Artifical Intelligence*, 18(2):203–226, 1982.

[27] Albert Orriols-Puig, Núria Macià, Ester Bernadó-Mansilla, and Tin Kam Ho. Documentation for the data complexity library in c++. Technical Report 2009001, La Salle - Universitat Ramon Llull, April 2009.

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

[29] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Advances in Large Margin Classifiers*, 2000.

[30] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[31] Jarkko Salojärvi, Kai Puolamäki, Jaana Simola, Lauri Kovanen, Ilpo Kojo, and Samuel Kaski. Inferring relevance from eye movements: Feature extraction. Technical Report A82, Helsinki University of Technology, March 2005.

[32] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL `http://promise.site.uottawa.ca/SERepository/`.

[33] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In *Proceedings of the 4th International Conference on*

Advances in Intelligent Data Analysis, IDA '01, pages 309–318, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42581-0.

[34] Nicola Segata, Enrico Blanzieri, and Pádraig Cunningham. A scalable noise reduction technique for large case-based systems. In *Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, pages 328–342, 2009. ISBN 978-3-642-02997-4.

[35] Burr Settles. Active learning literature survey. Technical Report Computer Sciences Technical Report 1648, University of Wisconsin-Madison, January 2010.

[36] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992. ISBN 0-89791-497-X.

[37] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[38] Michael R. Smith and Tony Martinez. Increasing task accuracy with adaptive filter sets. In *In Submission*, 2012.

[39] Gregor Stiglic and Peter Kokol. GEMLer: Gene expression machine learning repository. University of Maribor, Faculty of Health Sciences, 2009. URL `http://gemler.fzv.uni-mb.si/`.

[40] Kirsten Thomson and Robert J. McQueen. Machine learning applied to fourteen agricultural datasets. Technical Report 96/18, The University of Waikato, September 1996.

[41] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

[42] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, November 2001.

[43] Jason van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning*, pages 935–942. ACM, 2007.

[44] Geoffrey I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[45] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[46] Bianca Zadrozny and Charles Elkan. Learning and making decisions when costs and probabilities are both unknown. In *KDD*, pages 204–213, 2001.

[47] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th international conference on Knowledge discovery and data mining*, pages 694–699. ACM, 2002.

Part II

Improving Machine Learning by Integrating Meta-information about Individual Training Examples into the Learning Process

Chapters 4 – 7 introduce several methods for integrating information about each instance into the learning process. Chapter 4 introduces a filtering method that removes instances that *should* be misclassified based on the values for their hardness measures. Chapter 5 introduces a method for weighting the instances in a training set. Chapter 6 introduces a method for ordering the training instances from least complex to most complex and then using this ordering in the learning process (curriculum learning). Curriculum learning is then compared with boosting and filtering. Chapter 7 examines the use of classifier diversity for being robust to class noise in filtering and ensembles. These chapters were published under the following references.

Michael R. Smith, and Tony Martinez. "Improving Classification Accuracy by Identifying and Removing Instances that Should Be Misclassified", In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2011)*, pages 2690–2697, August 2011.

Michael R. Smith, and Tony Martinez. "Reducing the Effects of Detrimental Instances", In *Proceedings of the 13th International Conference on Machine Learning and Applications*, pages 183–188, 2014.

Michael R. Smith, and Tony Martinez. "A Comparative Evaluation of Curriculum Learning with Filtering and Boosting in Supervised Classification Problems", Computational Intelligence, to appear, 2014.

Michael R. Smith, and Tony Martinez. "Becoming More Robust to Label Noise with Classifier Diversity", In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2015)*, to appear, July 2015.

Chapter 4

Improving Classification Accuracy by Identifying and Removing Instances that Should Be Misclassified

In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp 2690-2697, 2011.

Abstract

Appropriately handling noise and outliers is an important issue in data mining. In this paper we examine how noise and outliers are handled by learning algorithms. We introduce a filtering method called PRISM that identifies and removes instances that *should* be misclassified. We refer to the set of removed instances as *ISMs (instances that should be misclassified)*. We examine PRISM and compare it against 3 existing outlier detection methods and 1 noise reduction technique on 48 data sets using 9 learning algorithms. Using PRISM, the classification accuracy increases from 78.5% to 79.8% on a set of 53 data sets and is statistically significant. In addition, the accuracy on the non-outlier instances increases from 82.8% to 84.7%. PRISM achieves a higher classification accuracy than the outlier detection methods and compares favorably with the noise reduction method.

4.1 Introduction

It is common that noise and outliers exist in real world data sets due to errors such as typographical errors or measurement errors. When the data is modeled using machine learning algorithms, the presence of noise and outliers can affect the model that is generated. Improving how learning algorithms handle noise and outliers can produce better models.

Handling noise and outliers has been addressed in a number of different ways, beginning with preventing overfit. A common approach to prevent overfit is adhering to Occam's razor which states that the simplest hypothesis that fits the data tends to be the best one. Using Occam's razor, a trade-off is made between accuracy on the training set and the complexity of the model, preferring a simpler model that will not overfit the training set. Another technique to prevent overfit is to use a validation set during training to ensure that noise and outliers are not learned. Some learning algorithms have a built in method to remove suspected outliers, such as C4.5 which prunes leaves that are thought to be insignificant [16].

Other approaches explicitly address noise and outliers by trying to identify them. One difficulty, however, is that there is no agreed upon definition of what constitutes an outlier or how to distinguish noise from an exception. Outlier detection aims a finding anomalies in the data and has been done using a variety of approaches such as statistical methods [11], rule creation [10], and clustering techniques [3]. Noise reduction methods attempt to identify and remove mislabeled instances such as repeated edited nearest neighbor that removes any instance that is misclassified by a 3-nearest neighbor classifier [24] or removing instances misclassified by a voting ensemble [4].

In this paper, rather than attempting to identify anomalies or mislabeled instances, we identify instances that *should* be misclassified and examine how they affect classification accuracy. We introduce PRISM (*PReprocessing Instances that Should be Misclassified*), a novel filtering method that removes instances that *should* be misclassified using heuristics that predict how likely it is that an instance will be misclassified [21]. By "instances that should be misclassified," we mean that in the absence of additional information other than what the data set provides, the label assigned by the learning algorithm to the instance is the most appropriate one, even if it happens to be different from the instance's target value.

We call the instances that are removed *Instances that Should be Misclassified* or ISMs to distinguish them from outliers and class noise. ISMs exhibit a high degree of class overlap where class overlap refers to how similar an instance is to other instances of different classes

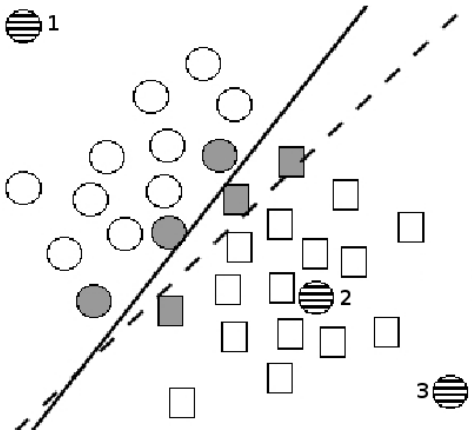

Figure 4.1: A hypothetical 2-dimensional data set with that illustrates how outliers (circles with striped fill) affect the model generated by a learning algorithm. The solid line represents the true classification border and the dashed line represents the classification from a learning algorithm that is affected by the outlier. The filled in instances represent border points.

in an area of the task space. This idea is portrayed in Figure 4.1 that shows a hypothetical 2-dimensional dataset with three outliers (instances with striped fill). In this paper, outlier instances 2 and 3 are removed by PRISM since they should be misclassified. Traditional outlier approaches that do not consider the class label would deem instances 1 and 3 as outliers but would not consider instance 2 as an outlier since class is not taken into account. Noise reduction techniques may not remove instance 3 since it is sufficiently different from the square instances.

We also classify each instance as a ISM, border point, or other based on a set of heuristics that predict how likely an instance is to be misclassified [21]. The presence of noise and outliers affects the classification border, effectively pulling the classification border as the the learning algorithm optimizes the squared error, or some other objective function. Removing the noise and outliers during training allows the learning algorithm to learn a more accurate classification boundary. This idea is also portrayed in Figure 4.1. The instances with a solid fill represent border points. The solid line represents the true classification border and the dashed line represents the classification from a learning algorithm that is affected by the outlier.

Removing the ISMs prior to training increases the overall average accuracy from 78.5% to 79.8%. In addition, the accuracy for instances that are not ISMs increases from 82.8% to 84.7% while the accuracy on ISMs decreases. In this manner, the learning algorithm models the data more precisely. Removing the ISMs during training improves the classification accuracy on the border points from 69.0% to 71.6% on all of the data sets and from 63.6% to 69.0% on data sets that had more than 10% of the instances identified as ISMs.

We also broadly investigate how outliers (as defined using different outlier detection methods) and noise affect the classification accuracy. In addition to PRISM, we use three outlier detection methods and 1 noise reduction algorithm; a distance-based approach [17], local outlier factor (LOF) [3], the enhanced class outlier distance based algorithm (ECODB) [18], and repeated edited nearest neighbor (RENN) [24]. The distance approach and LOF identify the traditional outliers and slightly decrease the classification accuracy. The other methods take the class label into account and improve classification accuracy.

Section 4.2 describes the experimental methodology. Section 4.3 describes how PRISM detects ISMs and, as an extension, how we identify instances as border points. The results are then presented in Section 4.4. We review related works in Section 4.5 and conclude in Section 4.6.

4.2 Experimental Methodology

We examine how filtering affects the classification accuracy of 53 data sets and 9 learning algorithms trained with and without filtering. The learning algorithms were chosen with the intent of being representative of a diverse set of learning algorithms commonly used in practice. The algorithms that were used are shown in Table 4.1. No parameter optimization was performed on any of the algorithms. They were used as implemented in Weka with their default parameters since we are interested in the effect that filtering has rather than parameter tuning [25]. The set of 53 data sets was selected from the UCI Machine Learning Repository [2]. This set was built to include data sets that vary significantly along impor-

Table 4.1: List of learning algorithms.

Learning Algorithms
Decision Tree (C4.5 [16]) {C4.5}
Naïve Bayes {NB}
Multi-layer Perceptron trained with Back Propagation {MLP}
Perceptron {Percep}
Support Vector Machine {SVM}
1-NN (1-nearest neighbor) {IB1}
5-NN (5-nearest neighbors) {IB5}
Repeated Incremental Pruning to Produce Error Reduction) {RIPPER}
RBF Network {RBF}

tant dimensions such as the number of attributes, the types of the attributes, the number of instances and the application domain. All of the data sets have a percentage of instances that are identified as outliers ranging from 0.1% to 52%. Each data set and algorithm is evaluated using 10-fold cross-validation on the filtered data set. The instances that are filtered (outliers/noise) are evaluated on a model trained using all of the non-filtered instances.

To examine the effect of filtering on classification accuracy, we first determine which instances to filter and remove them from the training set. We use the following outlier detection and noise reduction methods to filter instances:

- A distance-based approach as implemented by Rama-swamy et al [17] that ranks each instance based on its distance to its k nearest neighbors. The instances with the top n rankings are identified as outliers. This method partitions the data set to accommodate large data sets.
- LOF (Local Outlier Factor) [3] is an approach loosely related to density-based clustering that assigns each instance a value representing its potential of being an outlier with respect to the instances in its neighborhood. In this work, we identified the instances with the top n LOF values as outliers.

- ECODB (Enhanced Class Outlier Distance Based) [18] is an outlier detection approach that takes the class label into account. ECODB chooses as outliers the top n instances that have the smallest distance to their k-nearest neighbors, the greatest deviation, and a different class label from its k-nearest neighbors.
- RENN (Repeated Edited Nearest Neighbor) [24] repeatedly removes instances that are misclassified by a 3-NN classifier until no instances are misclassified.
- PRISM (PReprocessing Instances that Should be Misclassified) removes instances that should be misclassified by a learning algorithm. PRISM is described in Section 4.3.

We used RapidMiner [13] to implement the first three outlier detection methods. Once the datasets are filtered, we evaluate each learning algorithm using 10-fold cross-validation using the filtered dataset for training and the whole data set for testing. We then compare these results to those obtained by training the learning algorithm using all of the instances. In addition to filtering instances, we also identify instances as ISMs, border points or others as described in Section 4.3.

4.3 PRISM and Instance Types

PReprocessing Instances that Should be Misclassified or PRISM filters instances that *should* be misclassified. By "should be misclassified," we mean that based on the information in the dataset, the label assigned by the learning algorithm is the most appropriate even though it is incorrect. PRISM uses heuristics from Smith et al [21] to identify instances that should be misclassified. They conducted a comprehensive empirical analysis of what causes instances to be misclassified and found that class overlap is the primary contributor to misclassification. The work is summarized here to provide context for the heuristics that are used to identify ISMs. First, each instance is assigned an instance hardness value to determine which instances are intrinsically hard to correctly classify. The instance hardness values for each instance in a set of 57 data sets was collected on 9 learning algorithms using

the following equation:

$$instance\ hardness(x) = \frac{\sum_i^N incorrect(LA_i, x)}{N}$$

where x is the data instance, N is the number of learning algorithms, and $incorrect(LA, x)$ is a function returning 1 if an instance x was misclassified by the learning algorithm LA, and 0 otherwise. The hardest instances are those which no learning algorithm correctly classifies and are what PRISM attempts to filter. Their hardness value is 1. To avoid having to run all nine learning algorithms over each novel instance and to generalize the instances that are hard to correctly classify, we use five heuristics to predict instance hardness and to filter the instances.

The first heuristic, *k-Disagreeing Neighbors* (kDN), measures the local overlap of an instance in the original task space. The kDN of an instance is the percentage of that instance's k nearest neighbors (using Euclidean distance) that do not share its target class value.

$$kDN(x) = \frac{\mid \{y : y \in kNN(x) \wedge t(y) \neq t(x)\} \mid}{k}$$

where $kNN(x)$ is the set of k nearest neighbors of x and $t(x)$ is the target class value associated with x.

The next heuristic examines the disjunct size and measures how tightly the learning algorithm has to divide the task space to correctly classify an instance and the complexity of the decision boundary. The *Disjunct Size* (DS) of an instance is the number of instances in a disjunct divided by the number of instances covered by the largest disjunct in a data set.

$$DS(x) = \frac{\mid disjunct(x) \mid -1}{\max_{y \in D} \mid disjunct(y) \mid -1}$$

where the function $disjunct(x)$ returns the disjunct that covers instance x, and D is the data set that contains instance x. The disjuncts are formed using a C4.5 decision tree, created

without pruning and setting the minimum number of instances per leaf node to 1 [16].[1] In a decision tree, the disjuncts are the leaf nodes.

The third heuristic measures an instance's overlap on a subset of the features. C4.5 forms disjuncts but uses pruning. Using a pruned tree, the *Disjunct Class Percentage* (DCP) of an instance is the number of instances in a disjunct belonging to its class divided by the total number of instances in the disjunct.

$$DCP(x) = \frac{\mid \{z : z \in disjunct(x) \wedge t(z) = t(x)\} \mid}{\mid disjunct(x) \mid}$$

The fourth heuristic provides a global measure of overlap using all of the instances and attributes and a measure of the likelihood of an instance belonging to a class. The *Class Likelihood* (CL) of the attribute values for an instance belonging to a certain class is defined as

$$CL(x, t(x)) = \prod_{i}^{|x|} P(x_i | t(x))$$

where x_i is the value of instance x on its ith attribute. Continuous variables are assigned a probability using a kernel density estimation [9].

The fifth heuristic captures the difference in likelihoods and global overlap. The *Class Likelihood Difference* (CLD) is the difference between the class likelihood of an instance and the maximum likelihood for all of the other classes.

$$CLD(x, t(x)) = CL(x, t(x)) - \underset{y \in Y - t(x)}{\operatorname{argmax}} CL(x, y)$$

Using these heuristics we can identify instances that should be misclassified (ISMs) by assuming that ISMs have high instance hardness values. We can also identify instances as border points and other by assuming that the instance hardness values for border points range

[1] Note that C4.5 will create fractional instances in a disjunct for instances with unknown attribute values, possibly leading to DS values less than 1. Such cases are treated as though the disjunct covered a single instance.

between 0.11 and 1, and that other instances have a low instance hardness values. Based on these observations and the correlation between instance hardness and the heuristics, we identify an instance as a ISM or a border point using the following equation.

$$type(x) = \begin{cases} ISM & \text{if} \quad CLD(x, t(x)) < 0 \;\&\& \\ & \quad ((DS(x) == 0 \;\&\& \\ & \quad DCP(x) < 0.5)|| \\ & \quad DN(x) > 0.8) \\ border & \text{else if} \quad (DS(x) == 0 \;\&\& \\ & \quad DCP(x) < 1) \;|| \\ & \quad DN(x) > 0.2 \\ other & \text{otherwise} \end{cases}$$

That is, an instance is first identified as a ISM if the wrong class has the highest class likelihood for the instance and it meets one of two conditions: 1) the average DN value[2] is greater than 0.8; or 2) the instance is the only instance covered by the unpruned disjunct and the instance belongs to the minority class of the instances covered by a pruned disjunct. Therefore, a ISM disagrees with at least 80% of its neighbors or it is the only instance belonging to a disjunct and after pruning it is a minority class in the disjunct. If an instance is not a ISM, it is evaluated to determine if it is a border point. An instance is a border point if it satisfies one of the following two conditions: 1) the average DN value is greater than 0.2; or 2) the instance is the only instance covered by an unpruned disjunct and all of the instances covered by the pruned disjunct do not belong to the same class. If an instance is neither a ISM nor a border point, it is identified as *other* signifying that there is no or little class overlap. The values chosen in the heuristics were chosen based on empirical data

[2]To factor out the effect of neighborhood size, we use $DN(x)$ rather than $kDN(x)$, where $DN(x)$ is the average of $kDN(x)$ over all values of k between 1 and 17. Setting DN above 0.8 implies that on average, for every 5 instances in the neighborhood, at least 4 disagree with the instance under consideration.

to correlate with instance hardness [21]. In the following experiments, the other outlier detection methods are set to identify the same number of outliers as PRISM detected.

4.4 Results

This section presents the results of the experiments. Removing ISMs by PRISM for training increases the classification accuracy significantly. Only accuracy for the perceptron learning algorithm is not significantly changed by training without the ISMs as shown in the top part of Table 4.2. This may be due to the perceptron's inability to overfit the data and causing it to naturally ignore ISMs and hard instances. The change is most significant for the nearest neighbor learning algorithms.

To further determine the effectiveness of training without the ISMs, we examine the accuracy of the learning algorithms in the context of instance types. Table 4.2 shows the average accuracy for each of the nine considered learning algorithms and for a voting ensemble according to instance type on the datasets, where ">= 10%" averages the data sets with at least 10% of the instances being ISMs, "< 10%" averages those data sets with less than 10% of the instances being ISMs and "Overall" averages all of the data sets. "Orig" uses all of the data to train the learning algorithms and "NoISM" refers to training the learning algorithms without ISMs. The p-values are from the Wilcoxon signed-rank test and values in bold represent those that are not statistically significant with alpha equal to 0.05. The average classification accuracy and the voting ensemble are included to provide insight at a higher level than an individual learning algorithm. The change in classification accuracy by removing the ISMs for training is statistically significant in the majority of the cases.

The classification accuracy on the ISMs decreases for all learning algorithms when filtering with PRISM, which is expected since ISMs should not be learned by the learning algorithms and ISMs should be misclassified. The change is not significant for C4.5 and naïve Bayes. When 10% or more of the instances are ISMs, the change in accuracy is not significant for C4.5, IB1, IB5, and naïve Bayes while C4.5, MLP, naïve Bayes, perceptron, and RIPPER

Table 4.2: The average accuracy for the nine considered learning algorithms and voting ensemble on the data sets and the p-values using the Wilcoxon Signed-Ranks test comparing training with the original dataset and training without ISMs. >= 10% averages the data sets with at least 10% of the instances being ISMs according to instance type (ISMs, border points, and other). < 10% averages the data sets with less than 10% of the instances being ISMs and "Overall" averages all of the data sets. "Orig" uses all of the data to train the learning algorithms and "NoISM" refers to training the learning algorithms without ISMs.

	Classifier	>= 10%		p-value	<10%		p-value	Overall		p-value
		Orig	NoISM		Orig	NoISM		Orig	NoISM	
All	C4.5	0.601	0.617	**0.0681**	0.860	0.864	**0.1170**	0.802	0.808	0.0244
	IB1	0.529	0.587	0.0032	0.834	0.851	<0.0001	0.765	0.792	<0.0001
	IB5	0.577	0.617	0.0436	0.852	0.869	<0.0001	0.790	0.812	<0.0001
	MLP	0.595	0.626	**0.0571**	0.871	0.883	0.0495	0.808	0.825	0.0113
	NB	0.578	0.581	**0.2451**	0.804	0.815	0.0015	0.753	0.762	0.0017
	Perceptron	0.574	0.597	0.0217	0.851	0.857	**0.2358**	0.788	0.798	**0.0548**
	RBFNet	0.538	0.566	0.0392	0.850	0.856	0.0066	0.779	0.791	0.0021
	RIPPER	0.545	0.584	0.0087	0.846	0.851	0.0436	0.778	0.790	0.0027
	SVM	0.604	0.619	**0.0571**	0.856	0.862	0.0351	0.799	0.807	0.0089
	Average	0.571	0.599	0.0040	0.847	0.856	<0.0001	0.785	0.798	<0.0001
	Voting	0.573	0.599	0.0089	0.885	0.891	**0.0778**	0.814	0.825	0.0041
ISMs	C4.5	0.073	0.061	>**0.05**	0.186	0.158	**0.4090**	0.160	0.136	**0.2843**
	IB1	0.104	0.086	>**0.05**	0.174	0.149	0.0003	0.158	0.135	0.0006
	IB5	0.117	0.078	>**0.05**	0.150	0.133	0.0008	0.143	0.121	0.0033
	MLP	0.184	0.082	0.0197	0.203	0.193	**0.0582**	0.198	0.168	0.0069
	NB	0.051	0.044	>**0.05**	0.133	0.125	**0.2005**	0.115	0.107	**0.1492**
	Perceptron	0.133	0.076	0.0037	0.141	0.128	**0.2912**	0.139	0.117	0.0329
	RBFNet	0.103	0.066	0.0089	0.233	0.174	0.0011	0.204	0.149	0.0001
	RIPPER	0.135	0.108	<0.01	0.216	0.157	**0.0618**	0.198	0.146	0.0179
	SVM	0.192	0.149	0.0392	0.128	0.118	0.0287	0.143	0.125	0.0044
	Average	0.121	0.083	0.0052	0.174	0.148	0.0016	0.162	0.134	0.0001
	Voting	0.090	0.064	>**0.05**	0.183	0.128	0.0146	0.162	0.114	0.0045
Border points	C4.5	0.672	0.709	0.0485	0.755	0.751	**0.4920**	0.737	0.741	**0.0853**
	IB1	0.567	0.664	0.0024	0.632	0.679	0.0001	0.617	0.675	<0.0001
	IB5	0.641	0.719	0.0032	0.660	0.708	0.0001	0.656	0.710	<0.0001
	MLP	0.659	0.729	0.0052	0.752	0.769	**0.2005**	0.731	0.760	0.0060
	NB	0.676	0.683	**0.3707**	0.680	0.696	0.0197	0.679	0.693	0.0036
	Perceptron	0.639	0.682	0.0068	0.720	0.724	**0.3409**	0.702	0.714	**0.1038**
	RBFNet	0.603	0.654	0.0087	0.716	0.731	0.0485	0.691	0.713	0.0009
	RIPPER	0.592	0.663	0.0051	0.722	0.735	**0.2266**	0.693	0.719	0.0071
	SVM	0.676	0.708	0.0150	0.713	0.719	**0.2451**	0.705	0.717	0.0239
	Average	0.636	0.690	0.0015	0.706	0.723	0.0008	0.690	0.716	<0.0001
	Voting	0.662	0.711	<0.01	0.774	0.784	**0.1190**	0.749	0.767	0.0060

Table 4.2: **cont.** The average accuracy for the nine considered learning algorithms and voting ensemble on the data sets and the p-values using the Wilcoxon Signed-Ranks test comparing training with the original dataset and training without ISMs. >= 10% averages the data sets with at least 10% of the instances being ISMs according to instance type (ISMs, border points, and other). < 10% averages the data sets with less than 10% of the instances being ISMs and "Overall" averages all of the data sets. "Orig" uses all of the data to train the learning algorithms and "NoISM" refers to training the learning algorithms without ISMs.

	Classifier	>= 10%		p-value	<10%		p-value	Overall		p-value
		Orig	NoISM		Orig	NoISM		Orig	NoISM	
Other	C4.5	0.878	0.991	0.0500	0.962	0.969	0.0028	0.945	0.973	0.0003
	IB1	0.892	0.985	0.0050	0.988	0.993	0.0150	0.969	0.992	0.0002
	IB5	0.943	1.000	**NEI**	0.997	0.999	0.0018	0.986	0.999	0.0002
	MLP	0.922	0.982	0.0500	0.977	0.987	0.0006	0.966	0.986	0.0001
	NB	0.882	0.990	**NEI**	0.912	0.918	0.0222	0.906	0.932	0.0052
	Perceptron	0.959	0.876	**>0.05**	0.971	0.975	0.0307	0.969	0.955	0.0170
	RBFNet	0.939	0.973	**>0.05**	0.961	0.968	0.0080	0.957	0.969	0.0024
	RIPPER	0.914	0.928	**NEI**	0.956	0.958	**0.1711**	0.948	0.952	**0.0668**
	SVM	0.990	0.992	**NEI**	0.977	0.981	0.0060	0.979	0.983	0.0044
	Average	0.924	0.968	0.0027	0.967	0.972	0.0001	0.958	0.971	<0.0001
	Voting	0.995	0.998	**NEI**	0.995	0.996	**0.2420**	0.995	0.997	**0.1131**

are not statistically significant when less than 10% of the instances are ISMs. The non-significance is important as it shows that the learning algorithms correctly misclassify ISMs. RBFNet and SVM appear to be most affected by ISMs as the change in classification accuracy is always significant.

By removing the ISMs for training, there is significant improvement for all of the learning algorithms on the border points except for C4.5 and perceptron. The increase is by as much as 10% for the data sets where 10% or more of the instances are ISMs. Only the naïve Bayes learning algorithm does not have a significant increase in classification on the border points on datasets with 10% or more ISMs. The increase is less or not significant on the border points on datasets with less than 10% ISMs and actually decreases slightly for C4.5.

For the other instances, there was not enough information (NEI in the table) to determine if the change in classification accuracy was significant for some of the learning algorithms. This is due in part to both methods being equivalent on some data sets. The change is significant for all of the learning algorithms except for RIPPER and the voting ensemble, although removing the outliers does not decrease the classification either. For all of the

Table 4.3: The average classification accuracy for each learning algorithm trained with and without filtering.

	Orig	Dist	LOF	ECODB	RENN	PRISM
C4.5	0.803	0.794	0.802	**0.807**	**0.805**	**0.809**
IB1	0.771	0.773	0.773	0.784	**0.809**	0.797
IB5	0.791	0.789	0.793	0.802	**0.822**	0.814
MLP	0.813	0.814	0.814	0.822	**0.829**	**0.831**
NB	0.765	**0.773**	0.767	**0.772**	**0.774**	**0.776**
Percept	0.801	0.803	0.798	**0.808**	**0.811**	**0.812**
RBFNet	0.796	0.791	0.792	0.797	**0.807**	**0.806**
RIPPER	0.787	0.787	0.788	0.792	0.790	**0.798**
SVM	0.805	0.803	0.801	**0.810**	0.808	**0.814**
Overall	0.792	0.792	0.792	0.799	**0.806**	**0.806**

other learning algorithms, there is a significant increase in classification accuracy. Thus, the presence of ISMs does affect non-ISM instances.

We also compared PRISM with other filtering techniques. When comparing the other filtering methods, only 48 of the 53 data sets are used. Five of the data sets were omitted because the distance approach, LOF, and/or ECODB ran out of memory (15 GB) when running on them. To determine statistical significance, we used the Friedman test and post-hoc tests as well as the Wilcoxon Signed-Ranks test as suggested by Demšar [5].

Table 4.3 shows the average classification accuracy on the test sets with and without filtering for each learning algorithm trained on the different subsets of data used for training. "Orig" refers to using the whole data set for training. The "Dist," "LOF," "ECODB," "RENN," and "PRISM" columns remove instances that were identified as outliers for training using the distance approach, LOF, ECODB, RENN, and PRISM methods respectively. The values in bold are the highest classification accuracy or within 0.5% of the highest classification accuracy for each learning algorithm. RENN and PRISM achieve the highest overall classification accuracy and a higher classification accuracy than Orig for all of the learning algorithms.

The effectiveness of each method is further shown by ranking the classification accuracy for each filtering method as shown in Table 4.4. The lowest rank is desired. The values in

Table 4.4: The average rank for each learning algorithm on 48 data sets trained with and without filtering.

Algorithm	Orig	Dist	LOF	ECODB	RENN	PRISM
C4.5	3.38	4.33	3.58	3.44	3.46	**2.81**
IB1	4.30	4.29	4.10	3.38	**2.24**	2.69
IB5	4.08	4.48	4.08	3.51	2.43	**2.42**
MLP	3.47	4.10	3.88	3.65	3.08	**2.82**
NB	4.06	3.53	3.67	3.5	3.33	**2.91**
Percept	3.44	3.75	3.79	3.76	3.30	**2.96**
RBFNet	3.63	3.94	3.84	3.74	3.01	**2.84**
RIPPER	3.98	3.83	3.77	3.02	3.72	**2.68**
SVM	3.54	3.73	3.70	3.5	3.64	**2.90**
Overall	3.76	4.00	3.82	3.50	3.13	**2.78**

Table 4.5: The average classification accuracy for each learning algorithm trained with various subsets of the data set.

Data Set	% ISMs	Orig	Dist	p-value	LOF	p-value	ECODB	p-value	RENN	p-value	PRISM	p-value
Breast Uterus	0.009	0.961	0.959	>0.05	0.962	>0.05	0.961	>0.05	0.966	>0.05	**0.972**	**<0.025**
ar1	0.05	0.901	0.917	**0.005**	0.881	**0.025**	0.907	>0.05	**0.926**	**<0.05**	0.915	**0.05**
cm1 req	0.056	0.725	0.734	>0.05	0.742	>0.05	0.74	>0.05	**0.775**	**0.01**	0.774	**0.025**
desharnais	0.148	0.641	0.664	>0.05	0.636	>0.05	0.641	>0.05	**0.665**	>0.05	0.661	>0.05
eucalyptus	0.179	0.578	0.56	>0.05	0.553	**<0.025**	0.576	>0.05	**0.612**	>0.05	0.600	**<0.05**
pasture	0.056	0.713	0.698	>0.05	0.688	>0.05	0.71	>0.05	0.772	**0.01**	**0.744**	**<0.05**
Average	0.083	0.753	0.755	>0.05	0.744	>0.05	0.756	>0.05	**0.786**	**0.05**	0.778	**0.01**

bold indicate the best average rank. Removing outliers using PRISM has the lowest rank overall and for each learning algorithm except for IB1. PRISM is the only outlier detection method that is ranked lower than using the original dataset for every learning algorithm. PRISM's overall ranking is the furthest away from 3.5 and only PRISM and RENN have a ranking lower than 3.5. The Friedman test rejects the null hypothesis that each subset of the original dataset used for training is equivalent with a p-value less than 0.01. Thus, classification accuracy is affected by removing outliers. The Friedman test only rejects the null hypothesis that every method should have the same rank (3.5 in this case). To compare the different methods for identifying outliers, we perform post-hoc tests as described by Demšar [5].

The Bonferroni-Dunn significance test was used to compare training the learning algorithms trained with filtered data sets against using the original data set for training. The test indicated that the distance approach (decrease in classification accuracy) and RENN and PRISM (increase in classification accuracy) are statistically different from training with the original data set with an alpha value of 0.05. We also compared PRISM against the distance approach, LOF and ECODB. The difference in classification accuracy between using PRISM to remove outliers and the other approaches is statistically significant with an alpha value of 0.05. These results were confirmed using Holm's and Hochenberg's procedures.

This shows that preprocessing a dataset before training affects classification accuracy. Filtering by PRISM and RENN do the best at increasing the classification accuracy overall. To a lesser extent, ECODB in general also increases classification accuracy and is statistically significant using the Wilcoxon signed-ranks test. Also, the increase in classification accuracy by PRISM and RENN is statistically significant and the decrease in accuracy by the distance approach and LOF are statistically insignificant with alpha equal to 0.05.

We also examined removing outliers during training on a test set of six non-UCI datasets drawn from a number of different fields: bioinformatics [22], agriculture [23], and software engineering [19] to demonstrate that PRISM is effective on novel data sets. Because the heuristics were discovered on UCI datasets, this set of data sets was used as a test set to ensure that the heuristics generalize well. The percentage of ISMs in the data sets range from 0.9% to 17.9%. The results are summarized in Table 4.5. Each row gives the percentage of ISMs that each data set contains, the average accuracy for each training set and the p-value for the change in accuracy using the Wilcoxon signed-rank test. The highest classification accuracies and the p-values that are statistically significant are in bold with alpha equal to 0.05. PRISM and RENN achieve the highest classification accuracy on each data set and provide an average increase of 2.5%. The other methods often result in lower classification accuracy. The change in accuracy by PRISM is also statistically significant for all of the data sets except for desharnais where as RENN is statistically significant for only three of

Table 4.6: The average classification accuracy for each learning algorithm trained with various subsets of the additional data sets.

	Orig	Dist	LOF	ECODB	RENN	PRISM
C4.5	0.780	0.769	0.751	0.775	0.781	**0.808**
IB1	0.733	0.734	0.718	0.736	**0.792**	0.769
IB5	0.737	0.753	0.729	0.748	**0.792**	0.771
MLP	0.745	0.758	0.771	0.766	**0.806**	0.796
NB	0.727	0.721	0.720	0.744	0.760	**0.763**
Percept	0.746	0.761	0.751	0.752	**0.787**	0.769
RBFNet	0.751	0.733	0.725	0.736	**0.767**	0.747
RIPPER	0.785	0.789	0.749	0.772	0.787	**0.805**
SVM	0.774	0.780	0.779	0.775	**0.801**	0.772
Overall	0.753	0.755	0.744	0.756	0.786	0.778

the datasets. Table 4.6 shows how each learning algorithm performed on the additional set of data sets. PRISM and RENN provide the highest classification accuracy for all of the learning algorithms.

Finally, we examine the effect of traditional outliers. We have shown that ISMs affect the classification accuracy of a data set by pulling the classification boundary toward the wrong class. Outliers could also pull the classification boundary in the other direction. Could the accuracy be increased even more by handling the outliers as well as the ISMs? To determine the effect of removing both ISM and outliers for training, we combine the set of ISMs from PRISM with the outliers using the distance method, LOF, and ECODB. 32.7%, 36.7%, and 37.8% of the outliers identified by the distance approach, LOF, and ECODB overlap with the set of ISMs found by PRISM.

Combining the instances identified by PRISM and an outlier detection method results in a loss of accuracy compared to using PRISM by itself. The loss, however, is only 0.8% on average and is *not* statistically significant with an alpha of 0.05 using the Wilcoxon signed-rank test. Outliers are typically considered to be instances that are different than other instances and/or are in an underrepresented area of the task space. By removing the outliers, there is no information for the learning algorithm to generalize well on them. Thus, removing them can be detrimental to the classification accuracy. However, removing the combination

of outliers identified by PRISM and another outlier detection method resulted in a higher classification accuracy than just using the other outlier detection method. For each outlier detection method, the increase in accuracy is statistically significant with a p-value less than or equal to 0.0001. Thus, ISMs most directly affect the classification boundaries produced by the learning algorithms.

4.5 Related Work

Outlier detection has received growing attention, especially from the data mining community where outliers may represent anomalies or points of focus [1, 11, 15]. One difficulty in outlier detection is that there is no agreed upon definition of what constitutes an outlier. As such, outlier detection methods have used synthetic data sets or have injected noisy instances into a data set to establish which instances are outliers, thereby making assumptions about the characteristics of outliers. Also, there are many outlier detection algorithms from a variety of fields using different approaches; a few techniques are reviewed here. Khoshgoftaar et al [10] use a rule-based outlier detection method to remove outliers. They analyzed their approach by artificially injecting noise into clean data from software measurement data of a NASA software project. Liu et al [12] present an ensemble method for detecting outliers similar to boosting. In boosting, each training instance is assigned a weight. This method is augmented by adding a weight to each attribute (information gain) and outliers are detected by comparing the weights of the training instances. Finally, rules are generated that result in the largest attribute weight information gain. An approach loosely related to density-based clustering is Local Outlier Factor (LOF) [3]. LOF assigns each instance a value representing its potential of being an outlier with respect to the instances in its neighborhood. A thorough survey of outlier detection methodologies is provided by Hodge and Austin [8].

The concept of class outlier mining has also been examined [7, 14]. The goal of class outlier mining is to detect outliers taking into account the class label. For example, Semantic Outlier Factor (SOF) [6] is a class outlier mining approach based on applying a clustering

technique that takes the class label into account. ECODB [18], used in this work, is another example of class outlier mining. PRISM is similar to these approaches in that it takes the class label into account. PRISM differs from the other methods in that it also takes into account the expected classification of the instance.

Closely related to class outlier mining is noise reduction [20, 24] that attempts to identify and remove mislabeled instances. For example, Brodley and Friedl [4] attempt to identify mislabeled instances using an ensemble of classifiers. Rather than determining if an instance is mislabeled, PRISM filters instances that should be misclassified. The sets of removed instances from the PRISM and other noise reduction techniques will expectedly be similar. A different approach by Zeng and Martinez [26] uses multi-layer perceptrons that changes the class label on suspected outliers assuming that the wrong label was assigned to that instance. Our work does not focus on a single learning algorithm, but rather examines the effects of instances that should be misclassified in a broader context.

4.6 Conclusions

In this paper we introduced PRISM, a novel filtering method that identifies instances that should be misclassified (ISMs). We used a composite heuristic to identify ISMs that combines ideas from multiple learning algorithms. We have shown that noise and outliers do affect how learning algorithms model the data. However, noise and outlier detection and removal is difficult because there is no universal definition of what an outlier actually is or if an instance is noisy. In addition to PRISM, we used 3 outlier detection approaches and 1 noise reduction method to train 9 learning algorithms with filtering and compared the results to those from the learning algorithms trained using the original data set. RENN and PRISM both resulted in higher classification accuracy and consistently ranked better than the other approaches. PRISM consistently ranked the best among all of the filtering approaches. The distance-based approach and LOF did not show an improvement in classification accuracy

but did allow a speed up in training by having less instances to train. The distance-based approach and LOF both ranked worse than training with the original data set.

Removing instances identified by RENN and PRISM for training achieved the highest overall classification accuracy compared with the learning algorithms trained on the original data sets as well as with outliers removed by the other methods. With PRISM, we were able to achieve improvements in classification accuracy regardless of the learning algorithm being evaluated. On average, the increase in accuracy was about 1.3%. However, on data sets where more than 10% of the instances are ISMs, the increase on average is 2.8% compared to 1.2% for data sets with less than 10% ISMs. Rather than focusing on correctly classifying the instances that should be misclassified and arbitrarily adjusting the classification boundary, removing the ISMs for training allows the learning algorithms to focus on the instances that can be correctly classified. Removing the ISMs allows a more appropriate decision surface to be discovered since the ISMs do not arbitrarily pull the decision surface from its more optimal position. This leads to higher classification accuracy.

The presence of noise and outliers affects the learned model as the accuracy on border points and other instances increases when the model is trained on filtered data. Learning algorithms such as C4.5 and multi-layer perceptrons are more robust to outliers in the training data than other models but removing the outliers for training improved their classification accuracy as well as the less robust learning algorithms. Examining each instance type, the accuracy for the ISMs decreased as would be expected, but the accuracy of the border points and other instances increased sufficiently to provide an overall increase in accuracy despite the decrease on the ISMs.

Another advantage to identifying ISMs is for evaluation. The instances that should be misclassified can be handled differently. For example, all of the outliers can be ignored when calculating classification accuracy since the outliers should be misclassified. The accuracy would then give more insight as to how closely the learning algorithms models the data. Using this approach for evaluation, ignoring the ISMs during training increases the classification

accuracy on average by 1.94% as compared to when training using all of the instances. Ignoring outliers during training is most effective with a high percentage of instances being outliers. When 10% or more of the instances are outliers, the average increase in classification accuracy is 5.0% compared to 1.1% for data sets with less than 10% outliers.

Outliers and noise affect how learning algorithms model a data set. By filtering noise and outliers for training, the classification accuracy can be improved and the model will more effectively model the data.

References

[1] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th international conference on Knowledge discovery and data mining*, pages 504–509, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

[2] Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/~mlearn/`.

[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. *SIGMOD Record*, 29(2):93–104, June 2000. ISSN 0163-5808.

[4] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

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

[6] Zengyou He, Shengchun Deng, and Xiaofei Xu. Outlier detection integrating semantic knowledge. In *Proceedings of the 3rd International Conference on Advances in Web-Age Information Management*, pages 126–131, 2002.

[7] Zengyou He, Joshua Zhexue Huang, Xiaofei Xu, and Shengchun Deng. Mining class outliers: Concepts, algorithms and applications. In *Proceedings of the 5th International Conference on Advances in Web-Age Information Management*, pages 589–599, 2004.

[8] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[9] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179, 1995.

[10] Taghi M. Khoshgoftaar, Naeem Seliya, and Kehan Gao. Rule-based noise detection for software measurement data. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 302–307, 2004.

[11] Jeremy Kubica and Andrew Moore. Probabilistic noise identification and data cleaning. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 131–138, 2003.

[12] Xiao-Dong Liu, Chun yi Shi, and Xue-Dao Gu. A boosting method to detect noisy data. In *Proceedings of the 4th International Conference on Machine Learning and Cybernetics*, volume 4, pages 2015–2020, 2005.

[13] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–940, New York, NY, USA, 2006. ACM.

[14] Spiros Papadimitriou and Christos Faloutsos. Cross-outlier detection. In *Proceedings of the 8th International Symposium on Advances in Spatial and Temporal Databases*, pages 199–213, 2003.

[15] Mikhail I. Petrovskiy. Outlier detection algorithms in data mining systems. *Programming and Computer Software*, 29(4):228–237, 2003.

[16] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[17] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *Int. Conf. on Management of Data (SIGMOD)*, 29(2): 427–438, 2000.

[18] Motaz K. Saad and Nabil M. Hewahi. A comparative study of outlier mining and class outlier mining. *CS Letters*, 1(1), 2009.

[19] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL `http://promise.site.uottawa.ca/SERepository/`.

[20] Nicola Segata and Enrico Blanzieri. Fast and scalable local kernel machines. *Journal of Machine Learning Research*, 11:1883–1926, August 2010. ISSN 1532-4435.

[21] Michael R. Smith. An empirical study of instance hardness. Master's thesis, Brigham Young University, April 2010.

[22] Gregor Stiglic and Peter Kokol. GEMLer: Gene expression machine learning repository. University of Maribor, Faculty of Health Sciences, 2009. URL `http://gemler.fzv.uni-mb.si/`.

[23] Kirsten Thomson and Robert J. McQueen. Machine learning applied to fourteen agricultural datasets. Technical Report 96/18, The University of Waikato, September 1996.

[24] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

[25] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Fransisco, 2nd edition, 2005.

[26] Xinchuan Zeng and Tony R. Martinez. An algorithm for correcting mislabeled data. *Intelligent Data Analysis*, 5:491–502, 2001.

Chapter 5

Reducing the Effects of Detrimental Instances

In *Proceedings of the 13th International Conference on Machine Learning and Applications*, pp 183-188, 2014.

Abstract

Not all instances in a data set are equally beneficial for inducing a model of the data. Some instances (such as outliers or noise) can be detrimental. However, at least initially, the instances in a data set are generally considered equally in machine learning algorithms. Many current approaches for handling noisy and detrimental instances make a binary decision about whether an instance is detrimental or not. In this paper, we 1) extend this paradigm by weighting the instances on a continuous scale and 2) present a methodology for measuring how detrimental an instance may be for inducing a model of the data. We call our method of identifying and weighting detrimental instances *reduced detrimental instance learning* (*RDIL*). We examine RDIL on a set of 54 data sets and 5 learning algorithms and compare RDIL with other weighting and filtering approaches. RDIL is especially useful for learning algorithms where every instance can affect the classification boundary and the training instances are considered individually, such as multilayer perceptrons trained with backpropagation (MLPs). Our results also suggest that a more accurate estimate of which instances are detrimental can have a significant positive impact for handling them.

5.1 Introduction

The goal of supervised machine learning is to induce an accurate generalizing function from a set of labeled training instances. Given that in most cases, all that is known about a task is contained in the set of training instances, at least initially, the instances in a data set are generally considered equally. However, some instances are more detrimental than others for inducing a model of the data. For example, outliers or mislabeled instances are not as beneficial as border instances and are often detrimental in many cases. In addition, other instances can be detrimental for inducing a model of the data even if they are labeled correctly and are not outliers.

A possible effect of considering all instances equally, including the detrimental instances, when inducing a model of the data is shown in the hypothetical two-dimensional data set in Figure 5.1a. The solid line represents the "actual" classification boundary and the dashed line represents a potential induced classification boundary. Instances A and B are detrimental instances that "pull" the decision boundary away from the true boundary and cause the instances in the space between the true boundary and induced boundary to be misclassified. A learning algorithm can more precisely model the data by considering instances differently during training to suppress the effects of detrimental training instances.

This is especially true for learning algorithms such as backpropagation for training multi-layer perceptrons (MLP). Detrimental instances (e.g. instance A) have the greatest effect on the classification boundary since they can have the largest error value. As shown by Elman [5], this is particularly important during the early stages of training a MLP when the initial gradient is calculated. Elman proposes a method to initially train the MLP with simpler instances then gradually increase to more complex instances. This procedure has developed into curriculum learning and has had success in deep learning [1]. However, it is not as successful for shallow MLPs [22]. The impact of detrimental instances is lessened in other learning algorithms since the influence of a particular instance is localized. For example, k-NN only considers the k nearest neighbors of an instance.

Assuming that all that is known about a task is contained in the training set, how detrimental an instance is for inducing a model of the data can be estimated based on its relationship with the other instances in the data set. For example, instance A from Figure 5.1a represents a detrimental instance as an outlier–being in a region with instances of a differing class. In contrast, instance A in the data set shown in Figure 5.1b is not as detrimental given additional instances of the same class in the same region. As determining if an instance is detrimental exhibits a degree of uncertainty, we examine weighting the instances in a data set by their likelihood of being misclassified. Instance A from Figure 5.1a, for example, has a high likelihood of being misclassified while instance B may have a lower likelihood of being misclassified. Weighting the instances limits the influence of an instance proportionate to its detrimentality measure. We present a theoretically-motivated methodology for estimating the likelihood that an instance will be misclassified that lessens the dependence on any one model. We call this approach of weighting the instances by their probability of being misclassified *reduced detrimental instance learning* (*RDIL*). Filtering or removing detrimental instances prior to training can be viewed as a special case of instance weighting. We show that both filtering and weighting are viable solutions and examine when each is most beneficial.

We examine RDIL on a set of 5 learning algorithms and 54 data sets. We compare multiple versions of RDIL with another weighting scheme, pair-wise expectation maximization (PWEM) [17], as well as several filtering algorithms: misclassification filters and an ensemble filter [3], and repeated-edited nearest-neighbor (RENN) [24]. We find that some learning algorithms benefit more from filtering and others from instance weighting. Specifically, filtering is more beneficial for decision trees, rule-based learners, and nearest neighbor algorithms. Weighting the instances has a more significant impact on multilayer perceptrons. In cases with high amounts of noise, weighting the instances is generally preferable to filtering as it generally achieves higher accuracy and does not require a threshold to be set if filtering is based on a continuous value.

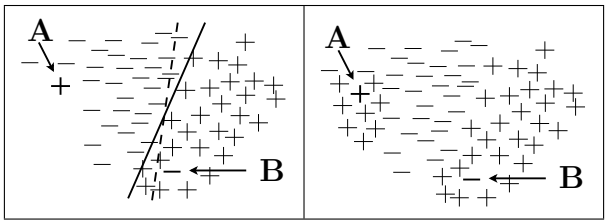

Figure 5.1: Hypothetical, 2-dimensional data set with two detrimental instances (instances A and B) that shows a) that treating all instances equally in a data set with detrimental instances can adversely affect the classification boundary and b) that how detrimental an instance may be is dependent on the other instances in a data set.

The remainder of the paper is organized as follows. Section 5.2 reviews related work in handling noise. Section 5.3 motivates weighting the instances. The detrimentality measure is presented in Section 5.4. Our experimental methodology is presented in Section 5.5. The results of RDIL are provided in Section 5.6. Section 5.7 concludes the paper.

5.2 Related Work

Many real-wold data sets contain detrimental instances that arise from noise (typos, measurement errors, etc.) or from the stochastic nature of the task. Noise is a subset of detrimental instances. Previous work has examined how class noise and attribute noise affects the performance of various learning algorithms [14]. They found that class noise is generally more harmful than attribute noise. The consequences of class noise, as summarized by Frénay and Verleysen [6], include 1) a deterioration of classification performance, 2) increased learning requirements and model complexity, and 3) a distortion of observed frequencies.

Most learning algorithms are designed to tolerate a certain degree of detrimental instances by making a trade-off between the complexity of the induced model and minimizing error on the training data to prevent overfit. For example, to avoid overfit many algorithms use a validation set for early stopping and/or regularization by adding a complexity penalty to the loss function [2]. Some learning algorithms have been adapted specifically to better handle noise. Boosting algorithms [18], for example, assign more weight to misclassified instances–

which often include mislabeled and noisy instances. To address this, Servedio [20] presented a boosting algorithm that does not place too much weight on any training instance.

Preprocessing the data set explicitly handles detrimental instances by removing, weighting, or correcting them. Filtering detrimental instances has received much attention and has generally been shown to result in an increase in classification accuracy, especially when there are large amounts of noise [7, 21]. One frequently used filtering technique removes any instance that is misclassified by a learning algorithm [9] or set of learning algorithms [3]. Other approaches use information theoretic or machine learning heuristics to remove noisy instances. Segata et al. [19], for example, remove instances that are too close or on the wrong side of the decision surface generated by a support vector machine. However, filtering has the potential downside of discarding useful instances and/or too many instances.

Rather than making a binary decision about the detrimentality of in instance, weighting allows a continuous scale. Filtering can be considered a special case of weighting where each instance is assigned a weight of 0 or 1. Rebbapragada and Brodley [17] weight the instances using expectation maximization to cluster instances that belong to a pair of the classes. The probabilities between classes for each instance is compiled and used to weight the influence of each instance.

Data cleaning does not discard any instances, but rather strives to correct the noise in the instances. As in filtering, the output from a learning algorithm has been used to clean the data. Polishing [23] trains a learning algorithm (in this case a decision tree) to predict the value for each attribute (including the class). The predicted (i.e. corrected) attribute values for the instances that increase generalization accuracy on a validation set are used instead of the uncleaned attribute values.

5.3 Modeling Detrimentality

Lawrence and Schölkopf [11] model a data set using a generative model that also models the noise. Let T be a training set composed of instances $\langle x_i, \hat{y}_i \rangle$ drawn i.i.d. from the underlying

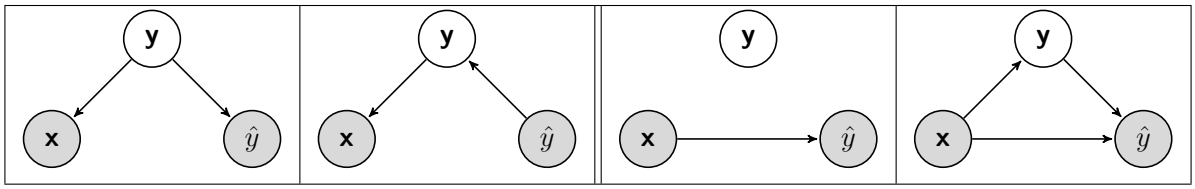

Figure 5.2: Graphical model of a a) generative probabilistic model, b) the generative model proposed by Lawrence and Schölkopf [11] and a discriminative probabilistic model for c) $p(\hat{y}|x)p(x)$ and d) $p(\hat{y}|x,y)p(y|x)p(x)$.

data distribution $\mathcal{D}$. Each instance has an associated latent random variable/feature y_i. Thus, x_i is the set of input features, $\hat{y}_i$ is the possibly noisy class label given in the training set, and y_i is the true unknown class label. Lawrence and Schölkopf assume that the joint distribution $p(x_i, y_i, \hat{y}_i)$ is factorized as $p(\hat{y}_i|y_i)p(x_i|y_i)p(y_i)$ as shown in Figure 5.2a. Since modeling the prior distribution of the unobserved random variable y_i is not feasible, they estimate the prior distribution of $p(\hat{y}_i)$ with some assumptions about the noise as shown in Figure 5.2b.

Following the premise of Lawrence and Schölkopf, we explicitly model the possibility that an instance is mislabeled (i.e. $y \neq \hat{y}$). Rather than using a generative model, though, we use a discriminative model since we are focusing on classification tasks and do not require the full joint distribution. Also, discriminative models have been shown to yield better performance on classification tasks [15]. A generative model, which models the full joint distribution of $p(x, y, \hat{y})$, differs from a discriminative model which is mostly concerned with modeling the likelihood of the class: $p(\hat{y}|x)$. Given a training set T, a discriminative model generally seeks to find the most probable hypothesis h that maps each $x_i \mapsto \hat{y}_i$–ignoring the fact that $\hat{y}_i$ may not equal y_i. This is shown graphically in Figure 5.2c where $p(\hat{y}_i|x_i)p(x_i)$ is estimated using a discriminative approach such as a neural network or a decision tree to induce a hypothesis of the data. The possibility of label noise is not explicitly modeled in this form (i.e. $p(y_i)$ is ignored). Label noise is generally handled by avoiding overfit such that more probable, simpler hypotheses are preferred.

Label noise can be more explicitly modeled by considering that $\hat{y}_i$ may be noisy by associating a latent random variable y_i with each instance. In this context, a supervised learning algorithm seeks to maximize $p(\hat{y}_i|x_i, y_i)p(y_i|x_i)p(x_i)$–modeled graphically in Figure 5.2d. This factorization of the likelihood for the observed class label for an instance suggests that it should be weighted by the probability that y_i is the actual class. Thus, when considering the possibility that $\hat{y}_i \neq y_i$, it is natural to weight the instances by $p(y_i|x_i)$. This provides the motivation for *reduced detrimental instance learning* (RDIL). RDIL takes two passes through the data set. In the first pass, $p(y_i|x_i)$ is calculated. Next, a learning algorithm is trained with the training instances weighted by $p(y_i|x_i)$. However, calculating $p(y_i|x_i)$ is not trivial. A method to estimate $p(y_i|x_i)$ is described in the following section.

5.4 Estimating $p(y_i|x_i)$

In this paper, $p(y_i|x_i)$ is used as the detrimentality measure for each training instance. In general, $p(y_i|x_i)$ does not make sense outside the context of an induced hypothesis. Thus, using an induced hypothesis h from a learning algorithm trained on T, the quantity $p(y_i|x_i, h)$ can be approximated as $p(\hat{y}_i|x_i, h)$ assuming that $p(y_i|\hat{y}_i)$ is represented in h. In other words, the induced discriminative model is able to model if one class label is more likely than another class label given an observed noisy label. After training a learning algorithm on T, the class distribution for an instance x_i can be calculated based on the output from the learning algorithm.

The dependence of $p(y_i|x_i)$ on a particular hypothesis h can be removed by summing over all possible hypotheses h in $\mathcal{H}$ and multiplying each $p(\hat{y}_i|x_i, h)$ by $p(h)$:

$$p(y_i|x_i) \approx p(\hat{y}_i|x_i) = \sum_{h \in \mathcal{H}} p(\hat{y}_i|x_i, h)p(h). \tag{5.1}$$

This formulation is infeasible, though, because 1) it is not practical (or possible) to sum over the set of all hypotheses, 2) calculating $p(h)$ is non-trivial, and 3) not all learning algorithms

Table 5.1: The diverse set of algorithms (as determined by unsupervised meta-learning) used to estimate $p(y_i|x_i)$.

Learning Algorithms
* Multilayer Perceptron trained with Back Propagation (MLP)
* Decision Tree (C4.5)
* Locally Weighted Learning (LWL)
* 5-Nearest Neighbors (5-NN)
* Nearest Neighbor with generalization (NNge)
* Naïve Bayes (NB)
* RIpple DOwn Rule learner (RIDOR)
* Random Forest (Random Forest)
* Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

produce a probability distribution. These limitations make probabilistic generative models attractive, such as the kernel Fisher discriminant algorithm [11]. However, for classification tasks, discriminative models generally have a lower asymptotic error than generative models [15].

In this paper we approximate $p(\hat{y}_i|x_i)$ using a diverse subset of $\mathcal{H}$. The diversity of the subset of $\mathcal{H}$ refers to a set of hypotheses that differ in their classification of novel instances. The diverse subset of $\mathcal{H}$ is created using unsupervised meta-learning (UML) [12]. UML first uses classifier output difference (COD) [16] to measure the diversity between learning algorithms. COD measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions. UML then clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. We considered 20 learning algorithms from Weka with their default parameters [8]. The resulting dendrogram is shown in Figure 5.3, where the height of the line connecting two clusters corresponds to the distance (COD value) between them. A cut-point of 0.18 was chosen to create 9 clusters and a representative algorithm from each cluster was chosen to create a diverse set of $\mathcal{H}$. Other numbers of clusters could have been used. The learning algorithms that are used to estimate $p(\hat{y}_i|x_i)$ are listed in Table 5.1.

$p(\hat{y}_i|x_i)$ is estimated for each training instance using 10-fold cross-validation (the instance $\langle x,\hat{y}_i\rangle$ is *not* used to induce the hypothesis h). Using a set of diverse hypotheses induced by

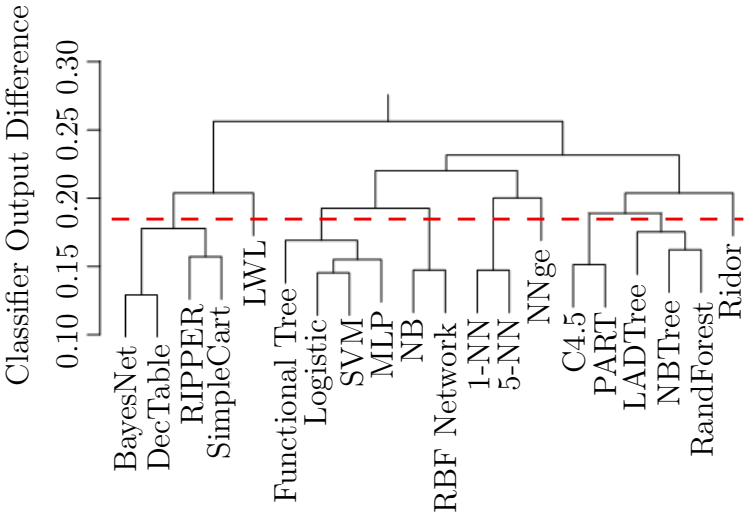

Figure 5.3: Dendrogram of the considered learning algorithms clustered using unsupervised metalearning based on their classifier output difference.

the learning algorithms in $\mathcal{L}$, we approximate $p(\hat{y}_i|x_i)$ as:

$$p(\hat{y}_i|x_i) \approx p(\hat{y}_i|x_i, \mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} p(\hat{y}_i|x_i, l_j(T)) \tag{5.2}$$

where $l_j(T)$ is the hypothesis induced by the j^{th} learning algorithm trained on T. From Equation 5.1, $p(h)$ is estimated as $\frac{1}{|\mathcal{L}|}$ for the hypotheses induced by the learning algorithms in $\mathcal{L}$ and as zero for all of the other hypotheses in $\mathcal{H}$.

5.5 Methodology

We investigate the effects of filtering and weighting on the C4.5, 5NN, MLP, Random Forest, and RIPPER learning algorithms (abbreviated in Table 5.1). Table 5.2 summarizes how an instance is weighted by $p(y_i|x_i)$ for the examined learning algorithms. For MLPs trained with backpropagation, the error $((t-o)f'(net))$ is scaled by $p(y_i|x_i)$ where $(t-o)$ is the difference between the target value and the output of the network, $f'(net)$ is the derivative of the activation function f and net is the sum of the product of each input i_j and its corresponding weight w_j: $net = \sum_j w_j i_j$. For Random Forests, the distribution for selecting

Table 5.2: How instance weighting is integrated into the considered learning algorithms.

LA	Orig	RDIL
MLP	$(t-o)f'(net)$	$p(y_i\|x_i)(t-o)f'(net)$
Random Forest	Uniform dist	Weighted by $p(y_i\|x_i)$
C4.5, 5-NN, RIPPER	Count number of instances, i.e. $\frac{\sum_{c_i} 1}{\sum_T 1}$	Sum $p(y_i\|x_i)$ $\frac{\sum_{c_i} p(y_i\|x_i)}{\sum_T p(y_i\|x_i)}$

instances in the random trees is weighted by $p(y_i|x_i)$ rather than being uniformly weighted. For the other learning algorithms that keep track of counts, each instance is weighted by $p(y_i|x_i)$. $\sum_{c_i}$ represents summing over instances that meet some criterion c_i and $\sum_T$ sums over all of the instances in the data set.

We consider three weighting schemes: RDIL-$\mathcal{L}$, RDIL-Biased, and PWEM described below. 1) **RDIL-$\mathcal{L}$** uses $p(\hat{y}_i|x_i, \mathcal{L})$. Since not all learning algorithms in Table 5.1 produce a probability distribution, the Kronecker delta function $\delta(g(x_i), y_i)$ is used in this paper instead of $p(\hat{y}_i|x_i, h)$ where $h(x_i)$ returns the predicted class from the induced hypothesis h given input features x_i. 2) **RDIL-Biased** approximates $p(y_i|x_i)$ as $p(\hat{y}_i|x_i, h)$ where the hypothesis h is induced by the same learning algorithm that is used to induce a model of the data. To get a real-value from a single hypothesis, we compute a classifier score for each instance from the learning algorithm. Below, we present how we calculate the classifier scores for the investigated learning algorithms.

MLP: For multiple classes, each class from a data set is represented with an output node. After training a MLP with backpropagation, the classifier score is the largest value of the output nodes normalized between zero and one: $\hat{p}(y|x) = \frac{o_i(x)}{\sum_i^{|Y|} o_i(x)}$ where y is a class from the set of possible classes Y and o_i is the value from the output node corresponding to class y_i.

C4.5: To calculate a classifier score, an instance first follows the induced set of rules until it reaches a leaf node. The classifier score is the number of training instances that have the

same class as the examined instance divided by all of the training instances that also reach the same leaf node.

5-NN: The percentage of the nearest-neighbors that agree with the class label of an instance as the classifier score.

Random Forest: For each tree, an instance follows the induced set of rules until it reaches a leaf node. The counts from the reached leaf nodes for each class are summed together and then normalized between 0 and 1.

RIPPER: The percentage of training instances that are covered by a rule and share the same class as the examined instance.

Obviously, a classifier score does not produce a true probability. However, the classifier scores approximate the confidence of $p(y_i|x_i)$. 3) Pair-wise expectation maximization (**PWEM**) [17] weights each instance using the EM algorithm. For each pair of classes, the instances that belong to the two classes are clustered using EM where the number of clusters is determined using the Bayesian Information Criterion [10]. Given the $Y-1$ clusterings (Y is the number of classes in the data set), $p(y|x)$ is calculated as:

$$p(y_i|x_i) = \sum_{\theta} p(\theta)p(y_i|x_i,\theta) = \sum_{\theta} p(\theta) \sum_{c=1}^{k} p(y_i|c,\theta)p(c|x_i,\theta)$$

where θ is a clustering model induced using the EM algorithm, c is a cluster in θ, and k is the number of clusters in θ.

We also compare weighting with three filtering techniques. 1) **Filter-$\mathcal{L}$** uses $p(\hat{y}_i|x,\mathcal{L})$ for filtering, similar to the three learning algorithm ensemble filter examined by Brodley and Friedl [3]. Instances that are misclassified by 50% of the learning algorithms in the ensemble are filtered from the training set. Note that other percentages could also be used. We found that 50% generally produces good results compared to values of 70% and 90%. In practice a validation set could be used to determine the percentage that would be used. 2) **Filter-Biased** removes any instance that is misclassified by the same learning algorithm that is

being used to induce a model from the training set. 3) Repeated-edited nearest-neighbor (**RENN**) [24] repeatedly removes the instances that are misclassified by a 3-nearest neighbor classifier.

Each noise handling method is evaluated by averaging the results from ten runs of each experiment. For each experiment, the data is shuffled and then split into 2/3 for training and 1/3 for testing. The training and testing sets are stratified. Random noise is introduced by randomly changing $n\%$ of the training instances to a new label chosen uniformly from the possible class labels (noisy completely at random). The noise levels are examined at 0%, 10%, 20%, 30%, and 40%. We examine noise handling using the 5 chosen learning algorithms on a set of 54 data sets from the UCI data repository [13]. Statistical significance between pairs of algorithms is determined using the Wilcoxon signed-ranks test as suggested by Demšar [4].

As there is no way to determine if an instance is noisy or mislabeled without the use of a domain expert, most previous work adds artificial noise to show the impact of noise and how handling noise improves the accuracy. Generally, once there are large amounts of noise, a noise handling approach significantly increases the classification accuracy. In the following experiments, artificial noise is added to the data sets.

5.6 Results

In this section, we present the results of our experiments. For the tables in this section, the algorithm in the first row is the baseline algorithm that the algorithms in the subsequent rows are compared against. The values in the "g,e,l" rows represent the number of times that the accuracy from the baseline algorithm is greater than, equal to, or less than the compared algorithm. A ✗ represents cases where the baseline algorithm achieves significantly higher classification accuracy. A ✓ represents cases where the accuracy from the compared algorithm is significantly higher than the baseline algorithm.

Table 5.3: The average accuracy over the 54 data sets for the 5 considered learning algorithms using the investigated noise handling approaches with no artificial noise added to the data sets. A ✓ to the left represent cases where the noise handling approach significantly increases the accuracy and ✗ where the noise handling approach significantly decreases the accuracy.

	C4.5	5-NN	MLP	Rand For	RIPPER
Orig	79.31	79.37	81.67	81.18	78.35
RDIL-$\mathcal{L}$	78.19	78.72	**82.26** ✓	80.82	77.86
g,e,l	27,1,26	27,4,23	18,3,33	28,2,24	26,2,26
RDIL-Biased	79.29	78.34 ✗	81.49	80.94	77.98
g,e,l	23,7,24	32,7,15	23,5,26	26,4,24	29,4,21
PWEM	76.41	78.02 ✗	82.79	81.51	74.17
g,e,l	30,3,21	33,3,18	23,3,28	34,1,19	39,1,14
Filter-$\mathcal{L}$	79.55	79.40	81.80	81.66	78.98
g,e,l	25,11,18	23,9,22	23,4,27	28,2,24	27,5,22
Filter-Biased	79.34	76.99 ✗	81.39	81.16	77.20
g,e,l	25,7,22	35,4,15	24,10,20	21,12,21	30,7,17
RENN	76.83 ✗	76.99 ✗	78.80 ✗	78.20 ✗	76.65 ✗
g,e,l	32,3,19	35,4,15	38,1,15	35,2,17	34,2,18

Table 5.3 compares no noise handling (Orig) with the considered noise handling techniques. The only noise handling technique that significantly increases classification accuracy is a MLP using RDIL-$\mathcal{L}$. In contrast, no noise handling achieves significantly higher accuracy than using a noise handling technique in several cases. RENN achieves significantly lower classification accuracy for all of the considered learning algorithms. This highlights a point that is often overlooked in the noise handling literature–noise handling can be detrimental if used in all cases. Previous work has generally considered only a few data sets where noise handling is beneficial. The impact of filtering or weighting is also dependent on which learning algorithm is used to induce a model of the data. As expected, MLPs achieve the most significant increase in accuracy with instance weighting. On the other hand, C4.5, 5-NN, Random Forests, and RIPPER achieve the most significant increase in accuracy with filtering.

Examining the performance of the considered learning algorithms without noise handling ("orig" in Tables 5.4 and 5.5), we note that MLPs and random forests generally achieve the highest classification accuracy and may be the most tolerant to the inherent detrimental

instances in each data set. However, MLPs and Random Forests also appear to be the least robust to noise as they obtain the lowest average classification accuracy when more than 10% of the instances are corrupted with noise. With no artificial noise, MLPs and Random Forests achieve about 81% accuracy. With 20% artificial noise, the average accuracy decreases to about 72%. On the other hand, C4.5, 5-NN, and RIPPER achieve an average accuracy of about 79% with no artificial noise and an average accuracy of about 74% with 20% artificial noise. With high degrees of noise, the built-in noise handling mechanisms of learning algorithms become more beneficial.

5.6.1 Weighting Schemes

As instance weighting is not as well explored as filtering, we now examine various weighting schemes to handle class noise. Table 5.4 compares RDIL-$\mathcal{L}$ with RDIL-Biased and PWEM. The accuracies from the algorithms with no weighting are given to better measure the effectiveness of the methods. RDIL-$\mathcal{L}$ significantly outperforms the other weighting schemes in most cases (represented by bold p-values): 24 out of the 25 cases for PWEM, and 18 out of the 25 cases for RDIL-Biased. In no case does a competing weighting scheme achieve significantly higher classification accuracy than RDIL-$\mathcal{L}$. Recall that the nine learning algorithms were chosen to be diverse so as to represent more of the hypothesis space $\mathcal{H}$. This suggests that a better estimation of $p(\hat{y}_i|x_i)$ produces better results for weighting and filtering. This is shown empirically as RDIL-$\mathcal{L}$ and Filter-$\mathcal{L}$ have the most significant increase in accuracy for each learning algorithm (Table 5.3). However, there is an obvious trade-off since obtaining a more accurate estimate of $p(\hat{y}_i|x_i)$ is more computationally expensive.

5.6.2 Weighting VS Filtering

We now compare weighting against filtering. Weighting and filtering are *both* viable and significantly increase the classification accuracy when noise is added. The difference between filtering and weighting techniques depends on the estimation of $p(y_i|x_i)$. Generally,

Table 5.4: A comparison of the average accuracy from the investigated instance weighting methods on the considered learning algorithms. Bold values with a ✓ represent cases where RDIL-$\mathcal{L}$ (R-$\mathcal{L}$) acheives significantly higher accuracy than RDIL-Biased (R-B) or PWEM (PW).

	C4.5					5-NN				
	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
R-$\mathcal{L}$	78.19	77.03	76.33	74.32	71.08	78.72	77.86	77.01	75.07	70.63
R-B	79.29	77.14	**75.20✓**	**71.06✓**	**65.74✓**	**78.34✓**	**77.41✓**	**75.39✓**	**71.59✓**	**64.92✓**
g,e,l	25,4,25	32,2,19	38,0,16	44,0,10	43,1,10	34,7,13	41,3,9	43,1,10	46,1,7	44,1,9
PW	**76.41✓**	**74.50✓**	**73.34✓**	**70.94✓**	**68.28✓**	**78.02✓**	**77.54✓**	**76.12✓**	**73.56✓**	**67.18✓**
g,e,l	35,4,15	39,3,11	38,4,12	41,0,13	37,0,17	34,4,16	36,2,15	37,1,16	37,3,14	37,3,14
orig	79.31	76.92	74.32	69.709	63.08	79.37	77.63	74.42	69.96	62.54

	MLP					Random Forest				
R-$\mathcal{L}$	82.26	80.7	78.40	75.17	69.30	80.82	79.72	78.06	75.89	70.78
R-B	**81.49✓**	**78.19✓**	**73.84✓**	**69.14✓**	**62.10✓**	80.94	**78.24✓**	**74.01✓**	**68.25✓**	**60.93✓**
g,e,l	34,2,18	41,1,11	46,1,6	48,0,6	47,1,6	25,1,28	42,2,9	48,0,5	48,1,5	49,1,4
PW	**82.79✓**	**79.67✓**	**76.42✓**	**71.9✓**	**65.95✓**	81.51	**78.69✓**	**76.37✓**	**72.54✓**	**65.87✓**
g,e,l	34,4,16	37,2,14	38,1,14	42,0,12	42,0,12	33,4,17	34,4,15	40,4,9	47,1,6	48,1,5
orig	81.67	77.46	72.25	67.17	60.46	81.18	77.75	72.72	66.87	59.63

	RIPPER				
R-$\mathcal{L}$	77.86	76.54	75.54	73.46	69.63
R-B	77.98	76.28	**74.50✓**	**70.70✓**	**65.97✓**
g,e,l	27,3,24	31,2,20	36,3,15	46,2,6	45,0,9
PW	**74.17✓**	**71.94✓**	**70.68✓**	**68.57✓**	**64.82✓**
g,e,l	36,4,14	44,2,7	45,4,5	40,2,12	41,1,12
orig	78.35	76.32	73.45	69.87	65.10

estimating $p(y_i|x_i)$ with the set of learning algorithms $\mathcal{L}$ achieves greater classification accuracy than using a biased estimate. Table 5.5 compares RDIL-$\mathcal{L}$ with Filter-$\mathcal{L}$. With no noise, RDIL-$\mathcal{L}$ achieves significantly higher accuracy than Filter-$\mathcal{L}$ for the MLP. Since each instance can affect the classification boundary for MLPs (as shown in Figure 5.1), weighting the instances in the training set has a more significant impact in MLPs than the other learning algorithms which partition the input space. On the other hand, the Filter-$\mathcal{L}$ achieves a significantly higher accuracy than RDIL-$\mathcal{L}$ for the four other learning algorithms. Note that MLP with RDIL-$\mathcal{L}$ achieves the highest overall average accuracy for noise levels 0%-20% (RDIL-$\mathcal{L}$ achieves the highest accuracy for 30% and 40% noise using Random Forest and C4.5 respectively). Except for MLPs, the significance of the impact of RDIL-$\mathcal{L}$ increases as the noise level increases except for all of the examined learning algorithms. RDIL-$\mathcal{L}$ significantly increases the classification accuracy for C4.5, 5-NN, and Random Forests when there are high amounts of noise.

Over all noise levels, RDIL-$\mathcal{L}$ compared with Filter-$\mathcal{L}$ achieves significantly higher classification accuracy in 6 of the 25 cases and the filter-$\mathcal{L}$ achieves significantly higher classification accuracy in 7 cases. (In the other 12 cases, there is no significant difference). The Filter-$\mathcal{L}$ has a more significant effect than RDIL-$\mathcal{L}$ for RIPPER at noise levels 0, 0.1 and 0.2 and RDIL-$\mathcal{L}$ never achieves significantly higher classification accuracy than the Filter-$\mathcal{L}$. Therefore, instance weighting is not the best option for every learning algorithm. However, with the Filter-$\mathcal{L}$ we chose the threshold that produced the highest classification accuracy on the test set, which is not always possible to do. Instance weighting avoids the overhead of having to determine a threshold for filtering when using an ensemble filter. Instance weighting is better for learning algorithms that consider each instance individually and each instance can affect the classification boundary (e.g. MLP).

Table 5.5: A comparison of RDIL-$\mathcal{L}$ (R-$\mathcal{L}$) with the $\mathcal{L}$-filter (F-$\mathcal{L}$) on the considered learning algorithms. Bold values with a ✓ represent cases where RDIL-$\mathcal{L}$ (R-$\mathcal{L}$) acheives significantly higher accuracy than the $\mathcal{L}$-filter. The ✗ represents cases where the $\mathcal{L}$-filter acheives significantly higher accuracy.

	C4.5					5-NN				
	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
R-$\mathcal{L}$	78.19	77.03	76.33	74.32	71.08	78.72	77.86	77.01	75.07	70.63
F-$\mathcal{L}$	79.55 ✗	78.35 ✗	76.79	73.58	**69.30** ✓	79.40 ✗	78.35	76.6	**74.35** ✓	**69.64** ✓
g,e,l	19,2,33	21,1,31	30,1,23	33,0,21	36,0,18	19,4,31	28,2,23	26,4,24	31,0,23	31,0,23
orig	79.31	76.92	74.32	69.709	63.08	79.37	77.63	74.42	69.96	62.54

	MLP					Random Forest				
R-$\mathcal{L}$	82.26	80.7	78.4	75.17	69.30	80.82	79.72	78.06	75.89	70.78
F-$\mathcal{L}$	81.80	80.66	78.24	74.85	69.46	81.66 ✗	79.91	78.06	**75.29** ✓	**69.94** ✓
g,e,l	37,2,15	30,3,20	31,1,21	33,0,21	29,1,24	18,4,32	31,0,22	26,1,26	33,0,21	34,3,17
orig	81.67	77.46	72.25	67.17	60.46	81.18	77.75	72.72	66.87	59.63

	RIPPER				
R-$\mathcal{L}$	77.86	76.54	75.54	73.46	69.63
F-$\mathcal{L}$	78.98 ✗	77.82 ✗	76.40 ✗	73.92	69.84
g,e,l	15,3,36	18,3,32	18,0,36	27,1,26	25,1,28
orig	78.35	76.32	73.45	69.87	65.10

5.7 Conclusions

In this paper we examined handling detrimental instances using the hypotheses from multiple learning algorithms. We introduced *reduced detrimental instance learning* (RDIL) which weights each instance based on an approximation of $p(\hat{y}_i|x_i)$. We examined RDIL on a set of 5 learning algorithms and 54 data sets. We found that a better estimate of $p(\hat{y}_i|x_i)$ leads to better detrimentality handling in both instance weighting and filtering. Weighting the instances avoids having to spend extra computational time and having to use training instances to select a threshold for filtering when using an ensemble filter. Instance weighting has the greatest effect on learning algorithms where every instance can affect the classification boundary and the training instances are considered individually, such as multilayer perceptrons trained with backpropagation (MLPs). On the other hand, instance filtering had a more significant impact on the C4.5, 5-NN, Random Forest, and RIPPER learning algorithms with no artificial noise. However, instance weighting was shown to be preferable to filtering for the examined learning algorithms when there are high amounts of noise. An analysis of when to use a particular noise handling technique is a direction for future work.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[2] Christopher M Bishop and Nasser M Nasrabadi. *Pattern Recognition and Machine Learning*, volume 1. springer New York, 2006.

[3] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

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

[5] Jeffery L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

[6] Benoit Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[7] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. *Applied Artificial Intelligence*, 14(2):205–223, 2000.

[8] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[9] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179, 1995.

[10] Robert E. Kass and Larry Wassermann. A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion. *Journal of the American Statistical Association*, 90(431):928–934, 1995.

[11] Neil D. Lawrence and Bernhard Schölkopf. Estimating a kernel fisher discriminant in the presence of label noise. In *In Proceedings of the 18th International Conference on Machine Learning*, pages 306–313, 2001.

[12] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[13] Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[14] David F. Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.

[15] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, pages 841–848, 2001.

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

[17] Umaa Rebbapragada and Carla E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.

[18] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[19] Nicola Segata, Enrico Blanzieri, and Pádraig Cunningham. A scalable noise reduction technique for large case-based systems. In *Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, pages 328–342, 2009. ISBN 978-3-642-02997-4.

[20] Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

[21] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[22] Michael R. Smith and Tony Martinez. A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. *Computational Intelligence*, page to appear, 2014. URL `http://arxiv.org/pdf/1312.4986`.

[23] ChohMan Teng. Combining noise correction with feature selection. In *Data Warehousing and Knowledge Discovery*, volume 2737 of *Lecture Notes in Computer Science*, pages 340–349. 2003.

[24] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

Chapter 6

A Comparative Evaluation of Curriculum Learning with Filtering and Boosting in Supervised Classification Problems

Computational Intelligence, accepted, 2014. (DOI: 10.1111/coin.12047)

Abstract

Not all instances in a data set are equally beneficial for inferring a model of the data and some instances (such as outliers) can be detrimental. Several machine learning techniques treat the instances in a data set differently during training such as curriculum learning, filtering, and boosting. However, it is difficult to determine how beneficial an instance is for inferring a model of the data. In this paper, we present an automated method that orders the instances in a data set by complexity based on their likelihood of being misclassified (*instance hardness*) for supervised classification problems that generates a *hardness ordering*. The underlying assumption of this method is that instances with a high likelihood of being misclassified represent more complex concepts in a data set. Using a hardness ordering allows a learning algorithm to focus on the most beneficial instances. We integrate a hardness ordering into the learning process using curriculum learning, filtering, and boosting. We find that focusing on the simpler instances during training significantly increases generalization accuracy. Also, the effects of curriculum learning depend on the learning algorithm that is used. In general, filtering and boosting outperform curriculum learning and filtering has the most significant effect on accuracy.

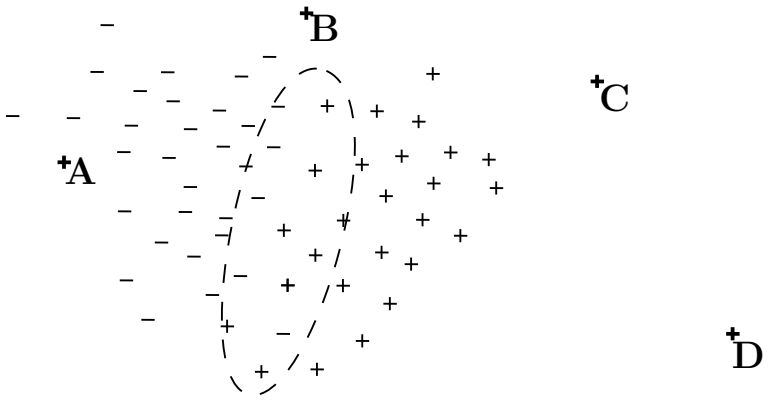

Figure 6.1: A hypothetical 2-dimensional data set.

6.1 Introduction

The goal of supervised machine learning is to model a task by "teaching" a learning algorithm through the presentation of labeled instances from a data set. The training instances are generally presented to a learning algorithm in no particular order and are generally treated as being equally important for inferring a model of the data. This can be problematic for many machine learning algorithms in deciding which initial search direction will lead to the optimal solution. Without guidance, the learning algorithm may choose an inappropriate initial direction to search the hypothesis space from which it may never be able to fully correct due to an inability to unlearn previously learned concepts.

Consider the hypothetical two-dimensional data set in Figure 6.1. Instances A, B, C, and D could be considered outliers and represent differing degrees of their extent of being an outlier. The instances in the dotted oval represent the border points, which could be used to define the classification boundary. Many learning algorithms have a built in mechanism to avoid overfitting the outliers, however, the presence of outliers could still affect the inferred classification border. Knowing before training begins which instances are the most informative instances could improve learning.

A number of methods have been developed that treat individual instances in a dataset differently during training to focus on the most informative ones. Filtering identifies and

removes noisy instances and outliers from a data set prior to training and generally results in an increase in classification accuracy on non-filtered test data [5, 11, 24]. Boosting also treats instances differently during training by incrementally adjusting the weights of the instances during training [9, 23]. Boosting iteratively trains m base learners and reweights the training data after each model is inferred such that the probability for selecting instances that are misclassified increases. Boosting, however, has been shown to be prone to overfitting outliers and noisy instances.

Curriculum learning was recently formalized by Bengio et al. [2] as a means of using an ordering of the training data from simplest to most complex to train a learning algorithm. Instances representing simple concepts are given a weight of 1 while instances that represent complex concepts are initially given a weight of 0, similar to filtering. As training progresses and the simper training instances are learned, a subset of the more complex training instances receive a weight of 1. This process continues until all of the training instances receive a weight of 1. From a cognitive point of view, curriculum learning is based on how humans acquire knowledge. For example, in schools, subject matter is organized into curricula such that simpler or foundational ideas are presented first. As learning progresses, more complex concepts and ideas can be learned by using the already learned simpler ideas. The main deficiencies of most previous work in curriculum learning are: 1) that there is no general method for ordering the instances by complexity and 2) that there is no method for determining when to add more complex instances to the training set. In previous work for curriculum learning, the ordering was done by hand or by some heuristic specific to the learning task (e.g., the number of words in a sentence).

In this paper, we present an automated method for ordering the instances in a data set based on their likelihood of being misclassified (*instance hardness* [26]) which we call a *hardness ordering*. The underlying assumption of ordering the instances by their hardness is that harder instances represent more complex and/or outlier instances. We use a hardness ordering to examine curriculum learning, filtering, and avoiding overfitting in boosting using

multilayer perceptrons (MLPs) and decision trees (DTs) on a set of 52 supervised classification problems from the UCI data repository. As filtering and boosting have received considerable attention [14, 17], we focus primarily on developing curriculum learning and comparing curriculum learning with filtering and boosting.

We find that ordering the instances in a data set by complexity significantly improves classification accuracy. Specifically, curriculum learning significantly increases classification accuracy for MLPs and significantly *decreases* the classification accuracy for DTs. Curriculum learning is better suited to learning algorithms that can be incrementally updated and learning algorithms that are more prone to getting stuck in local optima such as MLPs and especially in deeper networks. To examine the effectiveness of curriculum learning in common supervised classification tasks we limit our investigation to curriculum learning in MLPs with a single hidden layer in contrast to most other curriculum learning work that has examined curriculum learning in deeper networks. Filtering the most complex instances achieves higher classification accuracy than curriculum learning and boosting. We also examine boosting and curriculum learning with filtering. Boosting and curriculum learning with filtering significantly increases the accuracy over boosting and curriculum learning without filtering. Filtering has the most significant effect on the accuracy. We postulate that the significant change in accuracy when filtering is due to the fact that filtering creates a simpler surface for a learning algorithm to model by removing outliers and noisy instances. This, in turn, increases the generalization accuracy. By contrast, curriculum learning assumes that previously unlearnable complex instances become learnable once the simpler and foundational instances are learned. By adding the more complex instances into the training process, curriculum learning generates more complex models and the generalization accuracy decreases. Curriculum learning can also be seen as a regularization method where the amount of regularization is controlled by the relative weight given to early easy instances versus the hard instances introduced late in the training process.

The contributions of this paper include: 1) an automated method for ordering the instances in a data set based on their likelihood of being misclassified, 2) an examination of curriculum learning in a large number of supervised classification problems, whereas, previously, curriculum learning has only been examined in limited situations, and 3) a comparison of curriculum learning, filtering and boosting.

The remainder of the paper is organized as follows: In Section 6.2, we review related works. In Section 6.3, we present how to generate a hardness ordering. Implementation details of how we implement curriculum learning using a hardness ordering and a comparison with filtering and boosting are presented in Section 6.4. Section 6.5 gives conclusions and directions for future work.

6.2 Related Works

Emphasizing the most important instances has led to success in a number of previous works. Our work is motivated by the lack of a method to order the instances in curriculum learning and the importance of the early stages of training a learning algorithm. Elman [7] realized that the early stages of training dictate what solutions are possible in multilayer perceptrons. This is true for most gradient descent or greedy learning algorithms. During the early stages of training, the initial direction to search the hypothesis space is chosen. All practical machine learning algorithms avoid the computational cost of storing all possible hypotheses consistent with the training data by choosing a promising hypothesis at step t. The hypothesis space is then searched in a step-wise fashion, meaning that the hypothesis found at step $t+1$ is a continuation of the search of the hypothesis space at step t. It can also be difficult for most machine learning algorithms to backtrack and unlearn learned concepts. This adds more importance to the initial search direction that is chosen to follow. To aid in this problem, Elman introduced the idea of "starting small" meaning that only simple concepts should be used during the early stages of training. The simple concepts guide the initial search direction of the learning algorithm and can facilitate learning more complex

concepts for some situations (Elman used grammar rules). Other work has demonstrated the utility of starting small in specific application areas [18, 22, 27, 28]. Each approach shares the idea of breaking the learning task into subcomponents and then, starting with the simplest concepts, train by gradually increasing concept complexity.

Bengio et al. [2] formalized these ideas in *curriculum learning*. The idea behind curriculum learning is to first optimize a smoothed objective function and gradually reduce the degree of smoothing during the training process. At a more concrete level, curriculum learning is a weighting scheme for training. Each training instance is assigned a weight which controls how the instance is used in training. Initially, the weights on the training instances favor the "easier" instances or those that represent simpler concepts. As training proceeds, the weights on the training instances are updated such that "harder" instances and more complex concepts are introduced into the training set. This continues until all of the instances in the target training set are uniformly weighted. Bengio et al. examined curriculum learning in a couple of toy data sets where there was a clear distinction between harder and easier instances (i.e. the number of irrelevant features that do not have a zero value). In another experiment that built a language model from fixed-size sequences of text, they ordered the instances based on the frequency of the words in the text sequences.

Kumar et al. [15] recently presented self-paced learning for latent variable models, building on the idea of curriculum learning. In self-paced learning, a set of latent variables are learned that indicate which instances should be included for training for a specific latent-variable model. The choice of which instances are used in training is left to the optimization technique and the number of instances used is controlled with a variable which is annealed to eventually use all of the instances. Although self-paced learning creates an ordering, it is limited to latent variable models. Most latent variable models have been used to model observations in generative probabilistic models. However, discriminative models have been shown to yield better performance on classification tasks [19]. As we are focusing on classification tasks and do not require the full joint distribution we use discriminative models

but the use of latent variables in discriminative models is not well explored [31]. Thus, the primary shortcomings of previous work in curriculum learning are that there is no automated method for discriminative models without latent variables that produces a general ordering of the instances in a data set by complexity and that there is no method to determine when to add more complex instances to the training set.

Boosting is another approach that incrementally adjusts the weights of the instances during training [9, 23]. Boosting is an algorithm designed to target misclassified instances during training such that as training continues, uninformative instances that are well-classified are suppressed. Boosting iteratively trains m base learners on a subset of the training data to form an ensemble. After an iteration of training, the data is weighted such that the probability of selecting instances that are misclassified increases and the probability of selecting the instances that are classified correctly decreases. These techniques assume that the misclassified instances are the most informative and should be weighted more. However, this can lead to overfitting noise and new methods were proposed to ignore suspected outliers and weight the more informative or boundary instances higher [14]. Boosting differs from curriculum learning in that it is an ensemble method and the influence of the easier instances decrease as training progresses.

Filtering (removing outlier or noisy instances prior to training) is a similar approach to curriculum learning. Outliers and noisy instances have been observed to adversely affect an induced model [24]. Thus, the goal of filtering is to reduce the effects of outlier or noisy instances by removing them prior to training. One of the difficulties with outlier and noise identification is that there is no agreed-upon definition of what constitutes an outlier or noise. As such, a variety of different noise and outlier detection methods exist, such as statistical methods [1], density-based clustering [4], and classification-based methods [5, 13]. These methods have been used to identify and remove outliers prior to training, resulting in higher classification accuracy [5, 11, 17].

6.3 Ordering the Instances

Many machine learning techniques could benefit from knowing how beneficial an instance is to inferring a model of the data. In this paper, we use *instance hardness* [26] to order the instances by complexity. Instance hardness posits that each instance in a data set has a hardness property that indicates the likelihood that it will be misclassified. For example, outliers and mislabeled instances are expected to have high instance hardness since a learning algorithm will have to overfit to classify them correctly.

Instance hardness analyzes classification problems at the instance level rather than the data set level as is the case with most machine learning problems that seek to maximize $p(h|t)$, where $h : X \rightarrow Y$ is a hypothesis or function mapping input feature vectors X to their corresponding label vectors Y, and $t = \{(x_i,y_i) : x_i \in X \wedge y_i \in Y\}$ is a training set. With the assumption that the pairs in t are drawn i.i.d., the notion of instance hardness is found through a decomposition of $p(h|t)$ using Bayes' theorem:

$$\begin{aligned} p(h|t) &= \frac{p(t|h)\ p(h)}{p(t)} \\ &= \frac{\prod_{i=1}^{|t|} p(x_i,y_i|h)\ p(h)}{p(t)} \\ &= \frac{\prod_{i=1}^{|t|} p(y_i|x_i,h)\ p(x_i|h)\ p(h)}{p(t)}. \end{aligned}$$

For an instance $\langle x_i,y_i \rangle$, the quantity $p(y_i|x_i,h)$ measures the probability that h assigns the label y_i to the input feature vector x_i. The larger $p(y_i|x_i,h)$ is, the more likely h is to assign the correct label to x_i, and the smaller it is, the less likely h is to produce the correct label for x_i. Hence, we obtain the following definition of instance hardness, with respect to h:

$$IH_h(\langle x_i,y_i \rangle) = 1 - p(y_i|x_i,h).$$

In practice, h is induced by a learning algorithm g trained on t with hyperparameters α, i.e., $h = g(t,\alpha)$. Thus, the hardness of an instance is dependent on the instances in the training data and the algorithm used to produce h. To gain a better understanding of what causes instance hardness in general, the dependence of instance hardness on a specific hypothesis can be lessened by summing instance hardness over the set of hypotheses $\mathcal{H}$ and weighting each $h \in \mathcal{H}$ by $p(h|t)$:

$$\begin{aligned} IH(\langle x_i,y_i \rangle) &= \sum_{\mathcal{H}} (1 - p(y_i|x_i,h))p(h|t) \\ &= \sum_{\mathcal{H}} p(h|t) - \sum_{\mathcal{H}} p(y_i|x_i,h)p(h|t) \\ &= 1 - \sum_{\mathcal{H}} p(y_i|x_i,h)p(h|t). \end{aligned} \tag{6.1}$$

When calculating instance hardness, the instance $\langle x_i,y_i \rangle$ is not included in t to induce h.

Practically, to sum over $\mathcal{H}$, one would have to sum over the complete set of hypotheses, or, since $h = g(t,\alpha)$, over the complete set of learning algorithms and hyperparameters associated with each algorithm. This, of course, is not feasible. In practice, instance hardness can be estimated by restricting attention to a carefully chosen set of representative algorithms (and parameters). Also, it is important to estimate $p(h|t)$ because if all hypotheses were equally likely, then all instances would have the same instance hardness value under the no free lunch theorem [30]. A natural way to approximate the unknown distribution $p(h|t)$, or equivalently $p(g(t,\alpha))$, is to weight a set of representative learning algorithms, and their associated parameters, $\mathcal{L}$, a priori with a non-zero probability while treating all other learning algorithms as having zero probability. Given such a set $\mathcal{L}$ of learning algorithms, we can then approximate Equation 6.1 with the following:

$$IH_{\mathcal{L}}(\langle x_i,y_i \rangle) = 1 - \frac{1}{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} p(y_i|x_i,g_j(t,\alpha)) \tag{6.2}$$

where $p(h|t)$ is approximated as $\frac{1}{|\mathcal{L}|}$ for the learning algorithms in $\mathcal{L}$[1] and the distribution $p(y_i|x_i,g_j(t,\alpha))$ is estimated using the indicator function since not all learning algorithms return a probability distribution. For simplicity, we refer to $IH_{\mathcal{L}}$ as simply IH proceeding forward.

In this paper, we estimate instance hardness by biasing the selection of representative learning algorithms to those that 1) have shown utility, and 2) are widely used in practice. We call such classification learning algorithms the *empirically successful learning algorithms* (ESLAs). To get a good representation of $\mathcal{H}$, and hence a reasonable estimate of IH, we select a diverse set of ESLAs using unsupervised metalearning [16]. Unsupervised metalearning uses Classifier Output Difference (COD) [20] to measure the diversity between learning algorithms. COD measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions. Unsupervised metalearning then clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. Here, we considered 20 commonly used learning algorithms with their default parameters as set in Weka [12]. The resulting dendrogram is shown in Figure 6.2, where the height of the line connecting two clusters corresponds to the distance (COD value) between them. A cut-point of 0.18 was chosen and a representative algorithm from each cluster was used to create $\mathcal{L}$ as shown in Table 6.1.

Table 6.1: Set $\mathcal{L}$ of ESLAs used to calculate instance hardness.

* RIpple DOwn Rule learner (RIDOR)
* Naïve Bayes
* Multilayer Perceptron trained with Back Propagation
* Random Forrest
* Locally Weighted Learning (LWL)
* 5-nearest neighbors (5NN)
* Nearest Neighbor with generalization (NNge)
* Decision Tree (C4.5 [21])
* Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

[1]The quantity $p(h|t)$ is set to zero for all other learning algorithms and parameter settings.

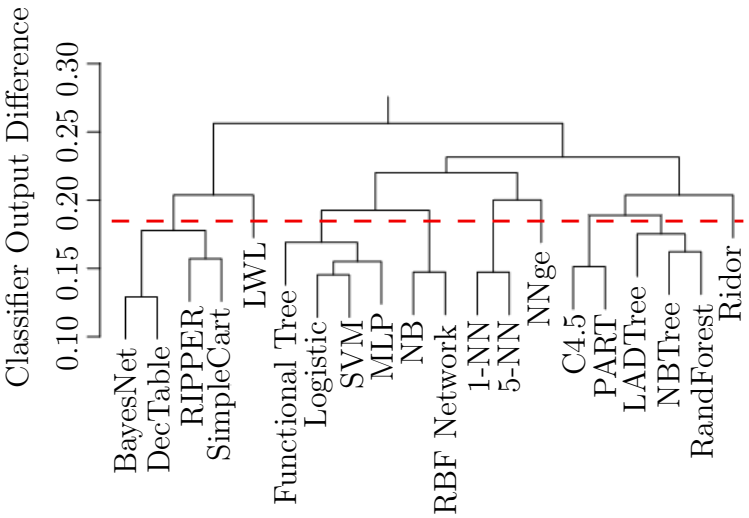

Figure 6.2: Dendrogram of the considered learning algorithms clustered using unsupervised meta-learning.

We recognize that instance hardness could be calculated with either more specific or broader sets of learning algorithms, and each set would obtain somewhat different results. We also recognize that the set of ESLAs is constantly evolving and thus no exact solution is possible. As the set of ESLAs grows and evolves, instance hardness can follow this evolution by simply adjusting $\mathcal{L}$. The size and exact make up of $\mathcal{L}$ are not as critical as getting a representative sample of ESLAs. While more learning algorithms may give a more accurate estimate of instance hardness, we demonstrate that both efficiency and accuracy can be achieved with a relatively small and diverse set of learning algorithms.

With this approach, the instance hardness of an instance is dependent both on the learning algorithm trying to classify it and on its relationship to the other instances in the data set as demonstrated in the hypothetical two-dimensional data set shown in Figure 6.1. Instances A, C, and D could be considered outliers, though they vary in how hard they are to classify correctly: instance A would almost always be misclassified while instances C and D would almost always be correctly classified. The instances inside of the dashed oval represent *border points*, which would have a greater degree of hardness than the non-outlier instances that lie outside the dashed oval. Obviously, some instances are harder for some learning algorithms than for others. For example, some instances (such as instance B) are

harder for a linear classifier than for a non-linear classifier because a non-linear classifier is capable of producing more complex decision boundaries.

6.4 Empirical Evaluation

With the ordering of the instances provided by instance hardness, we examine how to use a hardness ordering in the learning process. We examine using a hardness ordering on a set of 52 UCI data sets [8] for supervised classification problems shown in Table 6.2. We use a hardness ordering in curriculum learning, filtering and boosting. As curriculum learning is less fully explored, we focus on developing curriculum learning. We then compare curriculum learning with filtering and boosting. We implement the methods using multilayer perceptrons (MLPs) trained with backpropagation and decision trees (DTs) trained using C4.5 [21]. We chose a MLP because the training set can be augmented during training, which is a natural fit for curriculum learning. Other incremental learning algorithms could also be used. We train a MLP until convergence, where convergence is determined by a diminishing learning rate. Initially, the learning rate is set to 0.3 and it is reduced by 70% if the error on the training data does not decrease over an entire epoch. Training continues until the learning rate is less than 0.001. This helps protect against the choice of learning rate. We also examine curriculum learning in DTs to show that curriculum learning is not appropriate for all learning algorithms and to demonstrate the effects of filtering and boosting in multiple learning algorithms. Curriculum learning could also be implemented in other learning algorithms besides MLPs and DTs. However, curriculum learning requires 1) that the training set for a learning algorithm can be augmented during training and 2) that training with the augmented training set is a continuation of what was already learned by the learning algorithm. For most learning algorithms, meeting these requirements is a non-trivial task. Developing other learning algorithms that meet these requirements and incorporating curriculum learning into them is left as future work.

Table 6.2: Datasets used organized by number of instances, number of attributes, and attribute type.

# Inst	# Att	Attribute Type		
		Categorical	Numerical	Mixed
$\vee 100$	$k < 10$	Contact Lenses		Post-Operative
	$10 < k < 100$	Lung Cancer		Labor Trains
$100 < M < 1000$	$k < 10$	Breast-w Breast Cancer	Iris Ecoli Pima Indians Glass Bupa Balance Scale	Teaching- Assistant
	$10 < k < 100$	Audiology Soybean(large) Lymphography Congressional- Voting Records Vowel Primary-Tumor Zoo	Ionosphere Wine Sonar Heart-Statlog	Annealing Dermatology Credit-A Credit-G Horse Colic Heart-c Hepatitis Autos Heart-h
	$k > 100$			Arrhythmia
$1000 < M < 10000$	$k < 10$	Car Evaluation Chess Titanic	Yeast	Abalone
	$k < 100$	Splice	Waveform-5000 Segment Spambase Ozone level- Detection	Thyroid- (sick & hypothyroid)
$M > 10000$	$k < 10$	Nursery	MAGIC Gamma- Telescope	
	$k < 100$	Chess- (King-Rook vs. King-Pawn) Letter		Adult-Census- Income (KDD)

Hyperparameter optimization for MLPs, DTs, and curriculum learning uses 10 random searches of the parameter space rather than a grid search. Filtering, boosting and curriculum learning use the same hyperparameters that are found from optimization as well as the default parameters. We choose to use random parameter selection based on the work by Bergstra and Bengio [3]. The premise of the work by Bergstra and Bengio is that most machine learning algorithms have very few parameters that considerably affect the final model while most of the other parameters have little to no effect on the final model. Random search provides a greater variety of the parameters that considerably affect the model, thus allowing for better parameter selection of these parameters. For reproducibility, the exact process of parameter optimization for the learning algorithms is provided in the appendix.

For the experiments, we report the average accuracy obtained over the 52 data sets. As average accuracy is not very meaningful across multiple data sets, we also compare pairs of algorithms using the Wilcoxon signed-ranks test as suggested by Demšar [6] and include a count of the number of times an algorithm achieves an accuracy greater than, equal to, or less than a compared algorithm on the set of data sets. We also provide the accuracy for each individual data set. To get a better idea of the impact of curriculum learning and filtering using instance hardness, the accuracies for each data set from the learning algorithms used to compute instance hardness (ESLAs) are provided in Table 6.A.1 in the appendix. Table 6.A.1 also reports the accuracy from using the ESLAs as the base classifiers for an equally-weighted voting ensemble. Overall, using instance hardness, whether for curriculum learning or filtering, significantly improves the classification accuracy over each individual ESLA. With parameter optimization, curriculum learning and filtering also achieve a significantly higher classification accuracy than the voting ensemble.

6.4.1 Curriculum Learning

Curriculum learning trains a learning algorithm on the simplest concepts prior to training on the more complex instances, analogous to teaching a child a subject such as mathematics:

addition and subtraction are taught prior to algebra which is taught prior to calculus, etc. In general, one of the shortcomings of curriculum learning is that there is no automated way to generate curricula for curriculum learning, although some work generates a curriculum for a specific objective function for latent variable models [15]. Instance hardness is a natural fit for curriculum learning, providing an ordering of the instances from easiest to hardest. Ordering the instances with instance hardness requires the computational resources and time to compute a curriculum. After the curriculum is created, however, it is available to any algorithm that makes use of a hardness ordering.

Implementation Details

Even with an ordering for curriculum learning, a couple of questions need to be addressed for general supervised machine learning classification problems: 1) the initial complexity of the training instances, and 2) when to add more complex training instances.

The complexity of the initial training instances has a high likelihood of affecting the final model since the initial training instances determine the search direction that will be continued as more instances are added to the training set. Providing a learning algorithm with instances that are too easy and uninformative could lead to a learning algorithm choosing an arbitrary and/or incorrect gradient to follow. On the other hand, if the initial complexity is too complex, subconcepts may not be learned since they are grouped with the concepts that build on them.

As training begins, instances with an instance hardness value below a threshold are added to the training set. To gain insight into curriculum learning, we try different values for the initial complexity of the training set: 0, 0.25, 0.5, and 0.75. Each time new instances are added to the training set, the instance hardness threshold (IH) is incremented by 0.1 until all of the training instances are used for training.

When to add more complex instances to a training set could also have an impact on the final outcome of the induced model. If a learning algorithm does not train on the initial

instances long enough, a learning algorithm may not find the optimal gradient to follow. Training too long on a subset of the instances has the potential for a learning algorithm to overfit the subset. This could hinder learning the more complex instances that will be introduced later in the training process.

For MLPs, we consider two techniques of when to add more complex instances to the training set.

Every n epochs. For this method, instances with increasing instance hardness values are added to the training set every n epochs. This method is simple and it provides an indication of whether it may be better to let the MLP train more or less before adding more instances to the training set. We set n to 25, 50, 100, 200, 300, 400, and 500 to give an indication of how long to train before adding more instances. Other values could have been used as well.

Convergence. This method trains the MLP to convergence on the training set before adding more complex instances to the training set.

For DTs, more complex instances are added to the training set after a DT is induced using the given training set. Once more complex instances are added to the training set, the more complex training instances are propagated to a leaf node and the instances are evaluated to determine if the tree should be expanded at this leaf node. As the training set is augmented, the previously trained portions of the DT are not modified. For DTs, we consider pruning and not pruning the tree before adding more complex instances to the training set. For pruning at a given node, the error is estimated for its descendant branches as well as if the node was a leaf node. If the estimated error of the node as a leaf node is lower that the estimated error of the descendant branches, the node's descendants are pruned.

Results

The results from implementing curriculum learning in MLPs and DTs with default parameters are shown in Table 6.3. The bottom rows of Table 6.3 provide the aggregate statistics for

each method of adding more complex instances to the training set (average over all datasets, p-value from the Wilcoxon signed-rank test, and number of times that the accuracy from curriculum learning is greater than, equal to, or less than the original).

For MLPs, curriculum learning significantly increases classification accuracy with an alpha value of 0.05 for all of the methods of when to add more complex instances to the training set examined in this work. Adding more complex instances to the training set after training for 100 epochs is the most significant and results in the highest average classification accuracy. The average accuracy increases from 81.41% to 82.32% when more complex instances are added to the training set after training for 100 epochs. It is interesting to note that curriculum learning considerably decreases the classification accuracy on some of the datasets. The nursery dataset decreases in accuracy from 99.87% to around 97.42% regardless of when more complex training instances are added to the training set. The chess dataset decreases in accuracy as well. The reasoning behind this could be that there are no subconcepts in the dataset. On other datasets, curriculum learning increases classification accuracy considerably. For example, the post-operativePatient dataset increases in accuracy from 55.11% to 71.11% and the contact lenses dataset increases from 72.49% to 81.66%.

For DTs, curriculum learning significantly decreases classification accuracy. Despite this, on a few datasets, curriculum learning considerably increases classification accuracy. The labor dataset increases from 73.68%, to 85.26% and the anneal.ORIG dataset increases from 90.98% to 94.32%. However, other datasets show a considerable decrease in classification accuracy such as the breast-cancer dataset which decreases from 75.52% to less than 69%. Clearly, DTs are not as well suited for curriculum learning as MLPs are. This may be due to the use of entropy in C4.5 and the inability to backtrack and recover from splitting on a suboptimal attribute. MLPs, on the other hand, can partially recover from this through weight updates.

The poor performance of curriculum learning in DTs is more likely a result of the way that curriculum learning is implemented in DTs rather than the DTs themselves. DTs are

Table 6.3: Comparison of different strategies of when to add more complex instances in curriculum learning for MLPs and DTs.

	MLP								DT		
Dataset	Orig	25	50	100	200	400	500	conv	Orig	Prune	noPr
abalone	27.23	27.34	**27.70**	27.50	27.40	27.29	27.11	27.49	21.16	20.18	**21.55**
adult	84.15	84.74	84.88	**85.00**	84.86	84.91	84.89	84.26	**86.23**	84.64	84.42
anneal	**98.79**	98.15	98.24	98.28	98.26	98.24	98.24	98.35	90.98	**94.32**	92.43
arrhyth	68.45	69.29	69.38	68.98	68.49	**69.46**	69.42	68.71	**64.38**	63.19	62.30
audio	**83.18**	82.12	82.03	81.15	79.82	79.82	79.38	81.59	**77.88**	75.22	67.61
autos	76.00	78.24	78.14	78.82	78.04	78.04	77.65	**79.41**	81.95	**84.29**	77.27
balance	90.81	90.75	91.23	91.00	91.04	91.64	**91.96**	90.91	76.64	77.98	**79.20**
breast	67.62	71.53	71.81	72.65	73.21	73.84	**74.19**	71.32	**75.52**	68.81	65.31
breast-w	95.27	96.19	96.62	96.82	96.79	96.79	**96.88**	96.30	**94.56**	93.88	93.36
bupa	69.91	70.55	71.47	70.66	71.47	**71.53**	71.01	71.36	**68.70**	64.93	64.00
carEval	**99.44**	99.24	99.28	99.05	99.20	99.09	99.16	99.30	92.36	**93.09**	93.03
chess	**63.18**	54.78	54.38	53.83	53.57	52.88	52.88	60.97	56.58	**58.33**	55.45
KRvKP	99.32	**99.39**	99.32	**99.39**	99.37	99.31	**99.39**	99.32	99.44	**99.47**	98.39
colic	81.52	84.40	**84.83**	84.29	84.18	83.91	84.51	84.72	**85.33**	81.90	81.68
contact	72.49	75.83	74.99	75.83	80.00	**81.66**	**81.66**	80.83	**83.33**	75.00	64.17
credit-a	83.33	85.01	85.47	86.28	85.73	**86.55**	86.28	84.86	**86.09**	80.93	82.43
credit-g	71.96	74.20	74.72	74.92	74.22	74.88	**75.26**	74.78	**70.50**	68.22	68.94
derma	96.17	96.44	96.44	96.44	96.55	**96.72**	96.66	92.02	**93.99**	93.17	91.58
ecoli	84.76	85.65	85.71	85.17	85.53	85.77	**86.07**	83.03	**84.23**	83.57	81.01
glass	68.31	69.15	67.57	67.57	68.78	69.15	**71.12**	68.97	66.82	**68.04**	65.23
heart-c	80.92	82.31	83.10	82.64	82.44	**83.36**	82.50	81.91	**77.56**	76.57	74.13
heart-h	79.59	79.86	80.95	82.44	82.24	82.44	**82.65**	81.36	80.95	78.98	**83.20**
heart-s	79.55	80.07	82.51	83.18	**83.55**	83.03	82.81	80.66	76.67	**77.70**	73.78
hepa	81.80	83.22	83.74	84.64	84.38	84.64	**84.77**	82.58	**83.87**	77.29	74.19
hypo	97.55	**97.64**	97.55	97.59	97.43	97.32	97.30	97.56	**99.58**	99.53	97.68
iono	**91.05**	90.76	90.08	89.97	89.40	89.74	90.02	90.88	**91.45**	89.80	89.57
iris	**96.80**	96.66	96.53	**96.80**	96.66	96.40	96.66	96.40	**96.00**	94.67	95.20
labor	89.12	89.47	89.12	89.12	89.12	88.77	88.77	**92.63**	73.68	78.25	**85.26**
letter	**83.12**	81.70	81.80	81.85	81.84	81.64	81.47	82.57	**87.98**	87.40	81.97
lungCan	44.37	43.75	46.25	48.12	**48.75**	48.12	48.12	45.00	**50.00**	46.25	38.13
lympho	83.64	82.56	83.24	84.05	**84.18**	83.64	83.64	82.56	**77.03**	74.19	70.95
MagicTel	86.13	86.15	86.28	86.30	86.33	**86.34**	86.27	86.20	**85.06**	85.01	84.34
nursery	**99.87**	97.43	97.43	97.43	97.42	97.43	97.42	97.43	97.05	**98.63**	97.59
ozone	96.26	96.24	96.25	96.19	96.28	**97.12**	**97.12**	**97.12**	**96.33**	96.11	95.54
pimaDia	75.98	76.90	76.43	76.84	76.53	76.87	**77.00**	76.82	73.83	**74.06**	71.09
post-op	55.11	57.33	59.11	61.33	60.83	57.55	55.33	**71.11**	**70.00**	50.22	45.56
prim-tu	39.29	42.24	42.24	42.65	43.18	43.36	43.42	**44.89**	39.82	**40.65**	38.64
segment	**97.04**	96.98	96.96	96.72	96.94	96.88	96.84	96.86	**96.93**	96.67	94.99

Continued on next page

Table 6.3: **(Cont.)** Comparison of different strategies of when to add more complex instances in curriculum learning for MLPs and DTs.

	MLP								DT		
Dataset	Orig	25	50	100	200	400	500	conv	Orig	Prune	noPr
sick	**97.63**	97.45	97.51	97.43	97.49	97.39	97.55	97.50	**98.81**	**98.81**	97.68
sonar	80.19	**85.09**	84.90	84.80	84.90	84.61	84.90	81.24	71.15	**73.75**	70.10
soyb	93.49	93.64	93.79	93.64	**94.05**	93.82	93.52	91.12	**91.51**	90.69	79.21
spam	93.51	93.54	93.56	93.44	93.45	93.49	93.49	**93.62**	**92.98**	92.36	90.56
splice	95.57	95.66	95.68	95.68	95.54	95.74	**95.82**	95.56	**94.36**	91.50	86.41
ta	61.05	61.72	60.66	64.50	64.10	**66.22**	63.97	63.84	**52.98**	49.01	46.89
titanic	78.77	78.86	78.90	78.91	78.89	78.91	**78.94**	78.74	**78.92**	78.89	78.89
trains	70.00	**80.00**	**80.00**	**80.00**	70.00	70.00	70.00	70.00	**80.00**	70.00	70.00
vote	94.66	95.21	95.03	95.08	94.98	**95.31**	95.26	94.98	**96.32**	95.82	94.48
vowel	**92.70**	90.52	89.79	87.61	85.97	85.01	85.07	91.27	81.52	**81.66**	78.75
wave	83.80	84.70	85.14	85.32	**85.57**	85.48	85.26	84.13	**75.08**	74.72	72.78
wine	**97.86**	**97.86**	97.64	97.75	97.75	97.75	97.75	97.75	**93.82**	92.81	90.56
yeast	59.69	59.73	59.59	60.18	**60.29**	59.95	59.85	59.98	**56.00**	53.94	53.33
zoo	**95.24**	95.04	94.65	94.85	94.65	94.45	94.65	82.17	92.08	**93.27**	90.30
Ave	81.41	81.99	82.13	82.32	82.15	82.20	82.15	82.02	80.11	78.62	76.56
p-values		**0.004**	**0.004**	**0.001**	**0.002**	**0.002**	**0.003**	**0.034**		0.998	1
>,=,<		35,1,16	31,3,18	35,2,15	34,2,16	33,2,17	33,2,17	31,2,19		15,1,36	7,0,45

not designed to be incrementally updated as MLPs are. Thus, using curriculum learning is more natural and effective in learning algorithms that can be incrementally updated.

One potential problem is that there may not be enough information provided in the initial training set to infer a model of the data. To test this, we adjusted the instance hardness value of the initial instances in the training set. (Originally, only instances with an instance hardness of 0 were used in the initial training set). The aggregate results from setting the instance hardness value of the initial training instances to 0, 0.25, 0.5, and 0.75 are shown in Table 6.4 for adding more complex instances to the training set after 100 training epochs and training until convergence for MLPs and for pruning and not pruning before adding more complex instances in DTs. The p-values and counts are with respect to the initial training instances having an instance hardness value of 0. The results for each data set are provided in the appendix.

Table 6.4: Comparison of different initial complexity levels (instance hardness) for the training set.

		Inital IH:	0	0.25	0.5	0.75
MLP	100	Average	82.32	82.16	**82.42**	82.33
		p-values		0.382	0.100	0.334
		greater-equal-less		26-2-24	28-1-23	25-2-25
	Conv	Average	82.02	82.51	82.84	**83.09**
		p-values		0.107	**0.006**	**< 0.001**
		greater-equal-less		29-3-20	32-3-17	36-3-13
DT	Prune	Average	**78.62**	**78.62**	**78.62**	79.61
		p-values		1	1	0.605
		greater-equal-less		0-52-0	0-52-0	2-49-1
	NoP	Average	76.55	77.72	78.80	**79.22**
		p-values		**< 0.001**	**< 0.001**	**< 0.001**
		greater-equal-less		34-3-15	43-3-6	44-2-6

For MLPs, increasing the instance hardness value of the initial training set significantly increases the classification accuracy with an alpha value of 0.05 when training until convergence before adding more complex instances. When adding more complex instances to the training set after 100 epochs, increasing the initial complexity did not significantly increase the classification accuracy. For DTs with pruning, the initial instance hardness value has very little effect–all of the datasets have the same accuracy for initial instance hardness values of 0.25 and 0.5 and only three datasets change in classification accuracy with an initial hardness value of 0.75. Thus, pruning before adding more complex instances to the training set appears to lessen the effects of the initial instance hardness value. When DTs are not pruned before adding more complex instance to the training set, an initial instance hardness of 0.75 significantly increases classification accuracy over having an initial instance hardness value of 0. (It should be remembered that the original classification accuracy for DTs is 80.12%, thus, any form of curriculum learning for DTs examined so far does not increase the accuracy over the original).

The decrease in accuracy in curriculum learning could be lessened by including a hyperparameter that indicates whether to use curriculum learning or not. The value of this

hyperparameter could be based on the performance of curriculum learning on a validation set. Thus, neutral results could be obtained when curriculum learning decreases the classification accuracy. However, the focus of this paper is on the positive *and* negative results of curriculum learning. Therefore, we use curriculum learning for all of the data sets and learning algorithms but recognize that first determining whether to use curriculum learning or not for certain data sets and learning algorithms could improve the overall performance of curriculum learning.

Hyper-Parameter Optimization

The hyperparameters of a learning algorithm can have a drastic impact on their performance. In this section, we examine curriculum learning with hyperparameter optimization. The purpose of the previous section without hyperparameter optimization is to examine the effects of curriculum learning independent of hyperparameter optimization. As hyperparameter optimization is important for real-world problems, we examine curriculum learning with hyperparameter optimization in the context of what was learned from the results of using the default hyperparameter values. We follow the procedure by Bergstra and Bengio [3]–our methodology is contained in the appendix. For curriculum learning, we also optimize the values of the initial complexity of the training instances, and the number of epochs to train for MLPs. The results are shown in Table 6.5. The values in bold represent the highest accuracy value for each dataset and learning algorithm. The last three rows summarize the table–giving the average accuracy for each method, the p-values comparing when curriculum learning is used and when it is not used, and the number of times that using curriculum learning is greater than, equal to, or less than not using curriculum learning.

As expected, parameter optimization significantly increases the classification accuracy for MLPs and DTs. Also, curriculum learning significantly increases the classification accuracy for MLPs, increasing the average accuracy from 83.21% to 84.07%. The most significant increase in accuracy occurs for the lung cancer dataset which increases from 46.88% to 60.00%.

Table 6.5: Comparison of curriculum learning with parameter optimization for MLPs and DTs.

Dataset	MLP	MLP w/ CL	DT	DT w/ CL
abalone	**27.75**	27.65	**25.44**	23.76
adult	84.86	**85.65**	**86.33**	85.12
anneal	**99.55**	99.35	93.83	**95.43**
arrhyth	**70.58**	69.87	**70.71**	70.66
audio	83.19	**83.54**	77.35	**82.57**
autos	77.07	**78.63**	**84.39**	84.29
balance	87.20	**87.94**	**79.52**	78.24
breast	69.58	**74.62**	**74.69**	73.08
breast-w	95.99	**96.82**	**94.96**	94.54
bupa	**72.17**	71.07	**67.25**	66.72
carEval	**99.94**	99.80	**93.99**	93.09
chess	**69.73**	68.76	**59.85**	58.30
KRvKP	**99.50**	99.39	99.65	99.65
colic	82.34	**85.98**	**85.87**	85.16
contact	75.00	**75.83**	**87.50**	75.00
credit-a	85.94	**87.25**	86.35	**86.52**
credit-g	74.40	**76.68**	**72.74**	72.52
derma	97.27	**97.81**	93.06	**93.33**
ecoli	85.42	85.42	**83.33**	82.26
glass	**72.90**	72.06	69.53	69.53
heart-c	**84.82**	84.55	**77.69**	76.96
heart-h	82.65	**84.01**	**81.43**	81.29
heart-s	82.96	**84.37**	**83.19**	82.89
hepa	86.45	**86.97**	80.39	**83.61**
hypo	97.85	**97.99**	99.53	**99.57**
iono	**91.74**	90.88	**90.03**	89.80
iris	**96.67**	96.53	**94.93**	94.67
labor	89.47	**92.98**	82.11	82.11
letter	**89.70**	88.79	88.18	**88.44**
lungCan	46.88	**60.00**	65.00	65.00
lympho	**84.46**	83.78	**76.08**	75.54
MagicTel	86.95	**86.95**	**85.29**	85.02
nursery	**99.78**	99.29	97.27	**98.84**
ozone	**97.24**	97.15	**97.07**	96.77
pimaDia	**77.60**	77.24	75.03	75.03
post-op	62.22	**71.11**	71.11	71.11
prim-tu	44.84	**46.37**	**43.13**	42.48

Continued on next page

Table 6.5: **(Cont.)** Comparison of curriculum learning with parameter optimization for MLPs and DTs.

Dataset	MLP	MLP w/ CL	DT	DT w/ CL
segment	97.49	**97.52**	**97.05**	96.99
sick	**97.69**	97.47	**98.83**	98.82
sonar	82.69	**84.62**	75.77	**76.35**
soyb	**94.29**	93.65	**91.89**	91.22
spam	93.46	**93.59**	**92.93**	92.56
splice	95.77	**96.16**	**94.24**	92.15
ta	**62.91**	62.52	**63.84**	63.05
titanic	79.06	79.06	**79.06**	78.89
trains	80.00	80.00	90.00	90.00
vote	96.09	**96.74**	**96.55**	96.00
vowel	95.76	**95.98**	**84.67**	84.46
wave	86.22	**87.19**	**76.07**	75.70
wine	97.19	**98.43**	**93.60**	93.03
yeast	59.64	**60.19**	**58.06**	57.88
zoo	**96.04**	95.64	**94.06**	93.27
Average	83.21	84.07	81.93	81.52
p-values		**0.009**		0.999
>,=,<		29,3,20		9,8,36

Similar to using the default parameters, not using curriculum learning is significantly better for DTs. However, in the few cases in which curriculum learning does increase the classification accuracy, the increase can be considerable–the audiology dataset increases from 77.35% to 82.57% and the hepatitis dataset increases from 80.39% to 83.61%. Thus, curriculum learning is dependent on both the dataset and the learning algorithm that is being used.

6.4.2 Comparison with Filtering and Boosting

In this section, we compare curriculum learning with filtering and boosting. The filtering technique employed removes any instance with an instance hardness value greater than or equal to a threshold (we use 0.75) [25]. We denote the filtering method as $\text{IH}_{.75}$. Evaluation for filtering is done on unfiltered test data. We compare curriculum learning with two boost-

Table 6.6: A pair-wise comparison of curriculum learning with filtering and boosting for MLPs. The first row gives the p-values from the Wilcoxon signed-rank test for statistical significance. The second row gives the number of data sets with greater, equal, or lower classification accuracy.

	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
Average	83.21	84.83	82.80	82.82	84.07	85.07	84.87	85.07
Orig	1	1	0.116	0.085	0.991	1	0.999	1
	0,52,0	14,3,35	26,7,19	23,10,19	20,3,29	12,6,34	16,7,29	15,2,35
$IH_{.75}$	$<$ **0.001**	1	$<$ **0.001**	$<$ **0.001**	**0.007**	0.435	0.289	0.721
	35,3,14	0,52,0	36,3,13	34,4,14	31,3,18	23,6,23	25,6,21	22,3,27
AB	0.886	1	1	0.812	0.997	1	1	1
	19,7,26	13,3,36	0,52,0	13,22,17	20,1,31	10,4,38	15,3,34	13,1,38
MB	0.917	1	0.194	1	0.999	1	1	1
	19,10,23	14,4,34	17,22,13	0,52,0	18,1,33	12,3,37	16,2,34	15,1,36
CL	**0.009**	0.993	**0.003**	**0.001**	1	0.987	0.979	1
	29,3,20	18,3,31	31,1,20	33,1,18	0,52,0	21,1,30	20,2,30	15,1,36
$AB_{.75}$	$<$ **0.001**	0.569	$<$ **0.001**	$<$ **0.001**	**0.013**	1	0.062	0.771
	34,6,12	23,6,23	38,4,10	37,3,12	30,1,21	0,52,0	13,29,10	20,6,26
$MB_{.75}$	**0.001**	0.715	$<$ **0.001**	$<$ **0.001**	**0.022**	0.941	1	0.923
	29,7,16	21,6,25	34,3,15	34,2,16	30,2,20	10,29,13	0,52,0	18,5,29
$CL_{.75}$	$<$ **0.001**	0.282	$<$ **0.001**	$<$ **0.001**	$<$ **0.001**	0.232	0.079	1
	35,2,15	27,3,22	38,1,13	36,1,15	36,1,15	26,6,20	29,5,18	0,52,0

ing techniques: AdaBoost [10] (AB) and MultiBoost [29] (MB). In all of the experiments, hyperparameter optimization is used.

For curriculum learning with DTs, pruning is not done before adding more complex instances to the training set since it achieves higher classification accuracy. A statistical comparison of curriculum learning, filtering, and boosting for MLPs and DTs is given in Tables 6.6 and 6.7 respectively. The results for each data set are included in the appendix.

The first row shows the average accuracy for each method. Each following pair of rows gives the p-value from the Wilcoxon signed-rank test and the number of times that a method achieved an accuracy greater than-equal to-less than the method in the column heading. Curriculum learning (CL) significantly increases classification accuracy over boosting for MLPs.

Table 6.7: A pair-wise comparison of curriculum learning with filtering and boosting for DTs. The first row gives the p-values from the Wilcoxon signed-rank test for statistical significance. The second row gives the number of data sets with greater, equal, or lower classification accuracy.

	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
Average	81.93	85.03	83.90	84.14	81.52	85.64	85.03	83.26
Orig	1 0,52,0	1 8,5,39	1 10,3,39	1 4,4,44	$<$ **0.001** 36,7,9	1 1,4,47	1 4,3,45	1 7,4,41
$IH_{.75}$	$<$ **0.001** 39,5,8	1 0,52,0	0.979 16,2,34	0.998 14,4,34	$<$ **0.001** 41,4,7	1 3,5,44	1 6,5,41	0.812 22,3,27
AB	$<$ **0.001** 39,3,10	**0.021** 34,2,16	1 0,52,0	0.844 19,11,22	$<$ **0.001** 44,2,6	1 6,6,40	1 16,2,34	0.095 30,2,20
MB	$<$ **0.001** 44,4,4	**0.002** 34,4,14	0.159 22,11,19	1 0,52,0	$<$ **0.001** 46,2,4	1 7,5,40	1 11,4,37	**0.016** 31,3,18
CL	0.999 9,7,36	1 7,4,41	1 6,2,44	1 4,2,46	1 0,52,0	1 2,2,48	1 4,1,47	1 6,1,45
$AB_{.75}$	$<$ **0.001** 47,4,1	$<$ **0.001** 44,5,3	$<$ **0.001** 40,6,6	$<$ **0.001** 40,5,7	$<$ **0.001** 48,2,2	1 0,52,0	$<$ **0.001** 29,10,13	$<$ **0.001** 43,4,5
$MB_{.75}$	$<$ **0.001** 45,3,4	$<$ **0.001** 41,5,6	$<$ **0.001** 34,2,16	$<$ **0.001** 37,4,11	$<$ **0.001** 47,1,4	0.999 13,10,29	1 0,52,0	$<$ **0.001** 41,5,6
$CL_{.75}$	$<$ **0.001** 41,4,7	0.191 27,3,22	0.907 20,2,30	0.984 18,3,31	$<$ **0.001** 45,1,6	1 5,4,43	1 6,5,41	1 0,52,0

We also examined filtering prior to boosting and curriculum learning. Curriculum learning with filtering is denoted as $CL_{.75}$ and boosting with filtering is denoted as $AB_{.75}$ and $MB_{.75}$ for AdaBoost and MultiBoost. When a method is augmented with filtering, the filtered method significantly outperforms (in terms of classification accuracy) the method without filtering. This is particularly apparent with AdaBoost. AdaBoost does not significantly increase the classification accuracy over any of the other methods for MLPs and only over the original for DTs. Yet, with filtering, $AB_{.75}$ significantly increases the classification accuracy over the other non-filtered methods. This shows how prone AdaBoost is to overfitting. These results also show the importance of treating instances differently during training, particularly in the case of AdaBoost and filtering. Filtering has the most significant effect on accuracy, achieving higher classification accuracy than the original algorithm, curriculum

learning, and boosting. Applying filtering to curriculum learning and boosting also increases the classification accuracy. $AB_{.75}$ achieves the highest classification accuracy out of all of the investigated methods increasing the accuracy to 85% for MLPs and DTs.

Concerning curriculum learning, these results lead to the question of what is gained by using curriculum learning and under what circumstances is using curriculum learning most appropriate. Originally, curriculum learning was proposed as a continuation method for training deep networks (i.e. a MLP with more than a single hidden layer) used in specific tasks that could be divided into subconcepts. Curriculum learning helps avoid local minima which are prevalent in deeper networks. This correlates to why curriculum learning has a greater impact on MLPs than DTs since the problem of local minima is more severe for MLPs. We further investigate when curriculum learning is appropriate to use by examining the linearity of the data sets and data set hardness value. This investigates the perceived notion that curriculum learning may be better suited for more difficult tasks. The linearity of a dataset is estimated by taking the difference in accuracy between a linear classifier and a nonlinear classifier. We used the relative percentage difference between the accuracy of a MLP trained with backpropagation and a perceptron (MLP-per) and between the accuracy of a random forest and a linear SVM (RF-SVM). Data set hardness is the average of the instance hardness values for the instances in a data set.

We compared the linearity and data set hardness measures for each data set with the accuracies from curriculum learning with filtering and boosting. The difficulty measures are inconclusive. For example, for the autos dataset, curriculum learning provides a boost in accuracy as is expected since both MLP-per and RF-SVM are positive. The classification accuracy for the autos dataset is considerably higher for CL Prune (using a DT), increasing almost 5 percent On the other hand, an increase in accuracy is expected on the colic dataset, yet the classification accuracy decreased from about 86% to 82%. The difficulty of a data set, therefore, is not a sufficient condition for curriculum learning to outperform filtering the

dataset. It is difficult to know when to use curriculum learning for general machine learning tasks, especially when a simple method such as filtering produces similar results.

6.5 Conclusions

In this paper we presented a method for ordering the instances in a data set by complexity (hardness ordering) for supervised classification problems. A hardness ordering uses instance hardness to order the instances in a data set based on the their likelihood of being misclassified. The hardness ordering allows a learning algorithm to focus on the most informative instances. Using instance hardness to order the instances provides an explicit ordering of the easy instances to difficult instances and can be used in any algorithm that uses an ordering of the instances. However, computing instance hardness requires running N learning algorithms, which can be expensive for large data sets. Future work includes discovering a less expensive method for ordering the instances.

We integrated the hardness ordering for a data set into the learning process in curriculum learning. One of the main shortcomings of curriculum learning is the lack of a method for developing curricula. As curriculum learning is a relatively new approach, we examined using a hardness ordering as a general approach to implement curriculum learning. We examined curriculum learning on a set of 52 UCI data sets using MLPs and DTs and compared curriculum learning with filtering and boosting.

Our exploration with curriculum learning has shed interesting, and unexpected, light that curriculum learning performs strikingly similar to filtering and boosting in shallow MLPs. The similarity of curriculum learning in MLPs to filtering is somewhat expected. Elman [7] pointed out that one of the reasons starting small is so important is due to backpropagation's inflexibility of learning late in the learning process. As training progresses, weights become rigid and only very small changes are made during training. As complex instances are not trained on in curriculum learning until late in the learning process, they will only have a very minor (if any) effect on the trained model, especially with a large number of training

epochs between adding more instances. Thus, the harder instances are not expected to have a large impact on the final model.

For the DTs, the hope was that by starting with easier instances, more appropriate attribute splits would be chosen early in the learning process. As more difficult (and possibly noisy) instances are added later they would have less of an effect on the tree. DTs may not be an ideal candidate for curriculum learning because they are somewhat robust to noise due to using entropy to choose which attribute to split on and because pruning helps avoid overfitting the data.

The claim that curriculum learning is well suited for more difficult problems may or may not be true. Curriculum learning has an intuitive motivation and has proven to work well in specific past applications. For general supervised classification problems, however, it would appear that the benefits may not be as great as originally intended. Curriculum learning may be well suited for tasks that can be broken down into subtasks.

Our finding that curriculum learning does not out perform filtering and boosting in general classification problems may be an artifact of how we ordered the instances in a data set and of the particular training schedule used here. Using instance hardness to develop curricula orders the instances based on how hard they are to correctly classify but may not take into account if some concepts are sub-concepts, how concepts are related, etc. A better understanding of how the concepts in a data set are related could aid in creating more appropriate curricula for a data set. Future work for curriculum learning includes better understanding of how the instances in a data set are related to each other and developing curricula that accounts for the concepts contained in the instances.

We showed the positive impact of filtering instances prior to training. Filtering the noisy and outlier instances prior to training significantly increases the accuracy for the original learning algorithm as well as for curriculum learning and boosting. In fact, filtering prior to using AdaBoost produces significantly higher classification accuracy than all other methods investigated for MLPs and DTs. By filtering the noisy and outlier instances prior to training,

AdaBoost does not overfit the noise since it is no longer in the training data. AdaBoost further increases classification accuracy by focusing on the instances that are hardest to correctly classify.

References

[1] Vic Barnett and Toby Lewis. *Outliers in statistical data.* John Wiley & Sons Ltd., 2nd edition edition, 1978.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. *SIGMOD Record*, 29(2):93–104, June 2000. ISSN 0163-5808.

[5] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

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

[7] Jeffery L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

[8] A. Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

[9] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

[10] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[11] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. *Applied Artificial Intelligence*, 14(2):205–223, 2000.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[13] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179, 1995.

[14] Nir Krause and Yoram Singer. Leveraging the margin more carefully. In *Proceedings of the 21st International Conference on Machine Learning*, pages 63–70, 2004.

[15] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197. 2010.

[16] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[17] Hancong Liu, Sirish Shah, and Wei Jiang. On-line outlier detection and data cleaning. *Computers & Chemical Engineering*, 28(9):1635–1647, 2004.

[18] Gerhard Neumann, Wolfgang Maass, and Jan Peters. Learning complex motions by sequencing simpler motion templates. In *Proceedings of the 26th International Conference on Machine Learning*, pages 753–760, 2009.

[19] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, pages 841–848, 2001.

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

[21] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[22] Terence D. Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Transactions on Robotics and Automation*, 10(3), June 1994.

[23] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[24] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[25] Michael R. Smith and Tony Martinez. An extensive evaluation of filtering misclassified instances in supervised classification tasks. *In submission*, 2014. URL `http://arxiv.org/abs/1312.3970`.

[26] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.

[27] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. From baby steps to leapfrog: "how less is more" in unsupervised dependency parsing. In *Proceedings of Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759, 2010.

[28] Kewei Tu and Vasant Honavar. On the utility of curricula in unsupervised learning of probabilistic grammars. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

[29] Geoffrey I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[30] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[31] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1169–1176, 2009.

Appendix for "A Comparative Evaluation of Curriculum Learning with Filtering and Boosting in Supervised Classification Problems"

6.A Accuracies from the Learning Algorithms used to Compute Instance Hardness

This section presents the accuracies for each individual learning algorithm used to calculate instance hardness (ESLAs) for curriculum learning and filtering (listed in Table 6.1). Each ESLA uses the default parameters as set in Weka [2]. This section also presents the accuracies from a voting ensemble composed of the ESLAs. The results are presented in Table 6.A.1. Note that Ridor did not finish on the letter data set. In that case, instance hardness was calculated using the results from all of the other ESLAs.

Table 6.A.1: The accuracies for each learning algorithm (ESLA) on the examined data sets that were used to compute instance hardness as well as the accuracy for a voting ensemble composed of the ESLAs.

Data set	MLP	DT	5NN	LWL	NB	NNge	RF	Rid	RIP	Vote
abalone	27.23	21.16	20.54	22.84	23.84	20.40	22.41	20.64	19.03	25.01
adult	84.15	86.23	79.42	75.92	83.43	79.05	84.34	83.19	84.54	84.96
anneal	98.79	90.98	95.66	91.54	75.84	92.32	95.55	97.44	95.32	97.57
arrhyth	68.45	64.38	52.88	57.30	62.39	65.93	66.81	68.36	70.80	70.79
audio	83.18	77.88	77.88	47.79	73.45	71.24	78.76	73.01	73.01	75.39
autos	76.00	81.95	76.10	48.29	56.10	80.00	83.90	70.73	73.17	75.94
balance	90.81	76.64	86.56	55.20	90.40	81.92	80.48	79.52	80.80	86.32
breast	67.62	75.52	72.38	72.38	71.68	65.04	68.53	70.98	70.98	72.71
breast-w	95.27	94.56	95.14	90.27	95.99	95.99	96.42	95.85	95.42	96.78
bupa	69.91	68.70	62.90	59.13	55.36	66.67	68.41	63.19	64.64	68.78
carEval	99.44	92.36	93.52	70.02	85.53	94.50	92.77	96.30	86.46	95.52
chess	63.18	56.58	73.05	26.41	36.06	58.83	65.13	48.53	43.56	63.50
KRvKP	99.32	99.44	96.28	71.87	87.89	98.53	98.72	98.59	99.19	99.30
colic	81.52	85.33	81.25	81.52	77.99	80.43	85.60	83.70	84.24	85.85
contact	72.49	83.33	79.17	70.83	70.83	70.83	70.83	75.00	75.00	70.00
credit-a	83.33	86.09	81.16	85.51	77.68	82.61	85.22	83.33	85.80	86.61
credit-g	71.96	70.50	72.00	70.00	75.40	70.50	72.60	71.90	71.70	74.64
derma	96.17	93.99	94.54	87.70	97.27	96.17	94.81	93.17	86.89	97.38
ecoli	84.76	84.23	80.36	64.58	85.42	85.42	83.63	81.55	81.25	87.68
glass	68.31	66.82	70.56	44.86	48.60	70.09	72.90	63.08	68.69	70.56
heart-c	80.92	77.56	76.24	73.60	83.50	80.86	81.52	79.54	81.52	81.39
heart-h	79.59	80.95	76.87	79.25	83.67	79.59	78.23	80.95	78.91	81.73
heart-s	79.55	76.67	75.19	71.85	83.70	78.15	78.15	78.15	78.89	82.89
hepa	81.80	83.87	80.65	73.55	84.52	84.52	83.87	78.71	78.06	83.27
hypo	97.55	99.58	91.52	95.39	95.28	98.70	99.07	99.44	99.34	99.28
iono	91.05	91.45	86.32	82.34	82.62	90.03	92.88	88.03	89.74	92.22
iris	96.80	96.00	95.33	93.33	96.00	96.00	95.33	94.00	94.00	95.00
labor	89.12	73.68	82.46	85.96	89.47	77.19	87.72	80.70	77.19	93.68
letter	83.12	87.98	96.03	41.25	64.12	91.43	94.56	DNF	82.34	94.78
lungCan	44.37	50.00	37.50	46.88	50.00	62.50	56.25	43.75	43.75	50.91
lympho	83.64	77.03	82.43	73.65	83.11	78.38	81.08	85.14	77.70	84.20
MagicTel	86.13	85.06	80.94	72.53	72.69	82.19	86.95	82.73	84.77	86.31
nursery	99.87	97.05	98.38	87.83	90.32	96.27	98.25	95.69	96.84	98.33
ozone	96.26	96.33	95.27	97.12	70.78	97.04	97.16	97.04	96.10	97.01
pimaDia	75.98	73.83	70.18	71.22	76.30	73.96	74.09	75.00	76.04	74.92
post-op	55.11	70.00	53.33	67.78	67.78	60.00	62.22	71.11	70.00	71.67
prim-tu	39.29	39.82	39.23	38.94	50.15	40.71	41.89	37.17	38.35	48.32

Continued on next page

Table 6.A.1: **(Cont.)** The accuracies for each learning algorithm (ESLA) on the examined data sets that were used to compute instance hardness as well as the accuracy for a voting ensemble composed of the ESLAs.

Data set	MLP	DT	5NN	LWL	NB	NNge	RF	Rid	RIP	Vote
segment	97.04	96.93	97.14	80.48	80.22	96.28	97.75	96.15	95.71	97.19
sick	97.63	98.81	96.18	96.55	92.60	96.90	98.09	98.17	98.22	98.37
sonar	80.19	71.15	86.54	73.56	67.79	72.12	80.77	73.56	73.08	78.86
soyb	93.49	91.51	91.22	57.83	92.97	91.80	93.12	89.31	91.95	94.21
spam	93.51	92.98	90.78	78.46	79.29	92.13	94.72	91.83	92.39	94.36
splice	95.57	94.36	74.67	76.49	95.36	86.52	89.31	92.10	94.14	96.16
ta	61.05	52.98	66.23	54.30	50.33	65.56	64.24	39.74	39.07	55.49
titanic	78.77	78.92	78.92	77.74	77.87	67.61	79.06	78.42	78.33	77.90
trains	70.00	80.00	60.00	60.00	60.00	60.00	70.00	50.00	50.00	70.00
vote	94.66	96.32	92.41	95.63	90.11	96.09	96.55	94.25	95.40	95.86
vowel	92.70	81.52	99.29	36.36	63.74	87.47	96.06	77.68	69.70	92.67
wave	83.80	75.08	73.62	57.26	80.00	77.86	81.80	77.72	79.20	84.27
wine	97.86	93.82	94.94	88.76	97.19	97.75	97.19	91.01	91.57	98.00
yeast	59.69	56.00	52.29	40.70	57.61	54.25	58.02	53.50	58.09	61.45
zoo	95.24	92.08	96.04	86.14	95.05	95.05	89.11	94.06	86.14	95.59
Average	81.41	80.11	79.03	69.36	75.68	79.45	81.59	78.09	77.83	82.53

6.B Individual Results for Adjusting the Initial Complexity Level

This section provides the accuracy for the 52 investigated data sets when adjusting the initial complexity level of the training instances and for comparing curriculum learning, filtering, and boosting. The results for adjusting the initial complexity level of the training instances are shown in Tables 6.B.1 and 6.B.2 for MLPs and DTs respectively. Tables 6.B.1 and 6.B.2 are an expansion of Table 6.4 in the paper. The results for comparing curriculum learning, filtering, and boosting are shown in Tables 6.B.3 and 6.B.4 for MLPs and DTs and are an expansion of Tables 6.6 and 6.7 in the paper. The bold values represent the greatest classification accuracy for each dataset and learning algorithm.

Table 6.B.1: Comparison of different initial complexity levels (instance hardness) for the training set for curriculum learning in MLPs for each data set.

	MLP 100				MLP Conv			
Dataset	0	0.25	0.5	0.75	0	0.25	0.5	0.75
abalone	27.50	27.45	27.53	**27.59**	27.49	27.73	27.00	**27.74**
adult	**85.00**	84.88	84.87	84.74	84.26	**84.83**	84.31	84.77
anneal	98.28	98.64	**99.15**	98.37	98.35	98.77	**99.15**	98.41
arrhyth	68.98	69.07	69.20	**69.29**	68.71	69.07	**70.26**	69.64
audio	81.15	82.56	82.47	**83.00**	**81.59**	77.43	79.91	80.97
autos	78.82	78.14	78.92	**79.70**	**79.41**	76.97	78.43	78.92
balance	**91.00**	90.84	90.65	90.81	**90.91**	90.24	90.27	90.11
breast	72.65	**73.21**	72.44	71.81	71.32	**74.33**	72.93	73.91
breast-w	**96.82**	96.59	96.79	95.56	96.30	96.36	**96.82**	96.02
bupa	70.66	71.13	**72.00**	69.27	71.36	**72.17**	72.05	71.24
carEval	99.05	99.16	99.28	**99.32**	99.30	99.15	99.36	**99.47**
chess	53.83	54.87	59.61	**62.94**	60.97	59.32	59.88	**62.45**
KRvKP	**99.39**	99.33	99.25	99.33	99.32	99.24	99.23	**99.37**
colic	84.29	**84.94**	84.23	84.13	84.72	84.61	**84.83**	83.96
contact	75.83	73.33	**76.66**	**76.66**	80.83	78.33	**82.50**	80.83
credit-a	**86.28**	85.91	85.50	84.98	84.86	85.15	84.57	**85.44**
credit-g	74.92	74.98	**75.26**	74.68	74.78	75.88	75.12	**76.32**
derma	**96.44**	96.33	96.28	96.17	92.02	**96.66**	96.33	96.28
ecoli	85.17	85.71	**85.83**	84.99	83.03	**86.66**	86.13	86.19
glass	67.57	68.97	**70.18**	69.34	68.97	67.94	**72.71**	71.02
heart-c	82.64	83.10	83.56	**83.63**	81.91	82.50	82.04	**84.42**
heart-h	**82.44**	81.76	82.04	81.63	81.36	82.78	80.34	**83.87**
heart-s	83.18	82.81	82.59	**84.14**	80.66	84.59	82.66	**84.66**
hepa	84.64	**86.06**	84.00	85.80	82.58	84.77	84.51	**87.48**
hypo	97.59	**97.68**	97.55	97.58	97.56	**97.69**	97.58	97.62
iono	89.97	**91.28**	90.54	90.54	**90.88**	90.48	90.37	90.82
iris	**96.80**	**96.80**	**96.80**	96.66	**96.40**	95.59	95.86	**96.40**
labor	89.12	88.77	**89.47**	89.12	**92.63**	90.87	90.52	89.12
letter	81.85	82.08	82.73	**83.41**	82.57	81.14	82.47	**83.26**
lungCan	**48.12**	46.25	45.00	42.50	45.00	**48.12**	47.50	46.87
lympho	**84.05**	83.64	83.64	83.10	82.56	83.37	**86.08**	83.37
MagicTel	86.30	86.24	**86.45**	86.23	86.20	86.17	**86.35**	86.22
nursery	97.43	97.44	98.56	**99.87**	97.43	97.44	98.58	**99.87**
ozone	96.19	96.24	96.45	**96.50**	**97.12**	**97.12**	96.31	96.64
pimaDia	**76.84**	75.93	76.17	76.61	76.82	**77.29**	76.17	77.21
post-op	61.33	60.66	60.88	**62.88**	**71.11**	**71.11**	**71.11**	70.66
prim-tu	42.65	**43.42**	42.83	42.77	44.89	46.96	46.60	**48.84**

Continued on next page

Table 6.B.1: **(Cont.)** Comparison of different initial complexity levels (instance hardness) for the training set for curriculum learning in MLPs for each data set.

	MLP 100				MLP Conv			
Dataset	0	0.25	0.5	0.75	0	0.25	0.5	0.75
segment	96.72	97.01	97.08	**97.10**	96.86	96.91	97.09	**97.14**
sick	97.43	97.48	97.50	**97.52**	97.50	**97.57**	97.46	97.50
sonar	84.80	84.61	**86.44**	82.11	81.24	82.49	**85.38**	82.98
soyb	93.64	93.58	93.67	**93.82**	91.12	**94.31**	94.20	**94.31**
spam	93.44	**93.59**	93.51	93.32	**93.62**	93.61	93.51	93.51
splice	**95.68**	95.67	95.67	95.56	95.56	95.67	**95.80**	95.63
ta	**64.50**	61.32	61.58	61.05	**63.84**	62.64	63.31	63.70
titanic	78.91	78.91	78.89	**79.00**	78.74	78.53	**79.05**	79.00
trains	**80.00**	70.00	70.00	70.00	**70.00**	**70.00**	**70.00**	**70.00**
vote	95.08	94.75	94.75	**95.21**	94.98	**95.54**	95.49	95.26
vowel	87.61	90.02	92.82	**93.03**	91.27	89.29	92.52	**92.98**
wave	85.32	**85.47**	85.12	84.70	84.13	84.17	84.75	**84.76**
wine	**97.75**	97.52	97.64	**97.75**	97.75	97.30	97.75	**97.97**
yeast	60.18	60.37	**60.41**	60.16	59.98	59.25	60.24	**60.70**
zoo	94.85	**96.03**	95.44	95.24	82.17	94.65	94.45	**94.85**
Average	82.32	82.16	82.42	82.33	82.02	82.51	82.84	83.09
p-values		0.382	0.100	0.334		0.107	**0.006**	$<$ **0.001**
>,=,<		26,2,24	28,1,23	25,2,25		29-3-20	32-3-17	36-3-13

Table 6.B.2: Comparison of different initial complexity levels (instance hardness) for the training set for curriculum learning in DTs for each data set.

	DT Prune				DT no Prune			
Dataset	0	0.25	0.5	0.75	0	0.25	0.5	0.75
abalone	**20.18**	**20.18**	**20.18**	**20.18**	21.55	21.01	**21.90**	21.14
adult	**84.64**	**84.64**	**84.64**	**84.64**	84.42	84.31	84.70	**84.80**
anneal	**94.32**	**94.32**	**94.32**	**94.32**	92.43	94.12	94.16	**94.21**
arrhyth	**63.19**	**63.19**	**63.19**	**63.19**	62.30	64.78	64.34	**66.19**
audio	**75.22**	**75.22**	**75.22**	**75.22**	67.61	68.76	72.21	**75.84**
autos	**84.29**	**84.29**	**84.29**	**84.29**	77.27	74.05	**81.17**	79.90
balance	**77.98**	**77.98**	**77.98**	**77.98**	79.20	**80.06**	79.10	77.95
breast	**68.81**	**68.81**	**68.81**	**68.81**	65.31	67.55	67.48	**68.39**
breast-w	**93.88**	**93.88**	**93.88**	**93.88**	93.36	92.19	92.90	**94.05**
bupa	**64.93**	**64.93**	**64.93**	**64.93**	64.00	65.57	65.10	**67.59**
carEval	**93.09**	**93.09**	**93.09**	**93.09**	93.03	**93.18**	93.14	93.06
chess	**58.33**	**58.33**	**58.33**	**58.33**	55.45	60.32	62.54	**62.56**
KRvKP	**99.47**	**99.47**	**99.47**	**99.47**	98.39	99.34	99.31	**99.35**
colic	**81.90**	**81.90**	**81.90**	**81.90**	81.68	83.32	**83.48**	82.99
contact	**75.00**	**75.00**	**75.00**	**75.00**	64.17	72.50	**75.00**	**75.00**
credit-a	**80.93**	**80.93**	**80.93**	**80.93**	82.43	83.07	82.58	**83.39**
credit-g	**68.22**	**68.22**	**68.22**	**68.22**	68.94	70.08	70.14	**71.00**
derma	93.17	93.17	93.17	**93.28**	91.58	93.17	**93.39**	93.22
ecoli	**83.57**	**83.57**	**83.57**	**83.57**	81.01	80.95	83.87	**84.76**
glass	**68.04**	**68.04**	**68.04**	**68.04**	65.23	64.21	67.48	**69.16**
heart-c	**76.57**	**76.57**	**76.57**	**76.57**	74.13	76.17	76.63	**77.76**
heart-h	**78.98**	**78.98**	**78.98**	**78.98**	**83.20**	78.10	79.52	79.86
heart-s	**77.70**	**77.70**	**77.70**	**77.70**	73.78	76.52	77.04	**77.63**
hepa	**77.29**	**77.29**	**77.29**	**77.29**	74.19	79.61	**82.19**	80.00
hypo	**99.53**	**99.53**	**99.53**	**99.53**	97.68	99.20	99.48	**99.50**
iono	**89.80**	**89.80**	**89.80**	**89.80**	89.57	89.52	89.80	**90.37**
iris	**94.67**	**94.67**	**94.67**	**94.67**	**95.20**	94.40	94.00	93.33
labor	**78.25**	**78.25**	**78.25**	77.19	**85.26**	85.26	82.81	77.19
letter	**87.40**	**87.40**	**87.40**	**87.40**	81.97	83.73	86.60	**87.36**
lungCan	**46.25**	**46.25**	**46.25**	**46.25**	38.13	41.25	54.38	**60.00**
lympho	**74.19**	**74.19**	**74.19**	**74.19**	70.95	**77.30**	76.49	77.03
MagicTel	**85.01**	**85.01**	**85.01**	**85.01**	84.34	84.40	84.70	**85.06**
nursery	**98.63**	**98.63**	**98.63**	**98.63**	97.59	98.51	**98.79**	98.63
ozone	**96.11**	**96.11**	**96.11**	**96.11**	95.54	**95.95**	95.79	95.56
pimaDia	**74.06**	**74.06**	**74.06**	**74.06**	71.09	70.68	72.47	**73.91**
post-op	**50.22**	**50.22**	**50.22**	**50.22**	**45.56**	44.89	**45.56**	44.89
prim-tu	**40.65**	**40.65**	**40.65**	**40.65**	38.64	37.46	**41.06**	40.71

Continued on next page

Table 6.B.2: **(Cont.)** Comparison of different initial complexity levels (instance hardness) for the training set for curriculum learning in DTs for each data set.

	DT Prune				DT no Prune			
Dataset	0	0.25	0.5	0.75	0	0.25	0.5	0.75
segment	**96.67**	**96.67**	**96.67**	**96.67**	94.99	95.55	96.55	**96.60**
sick	**98.81**	**98.81**	**98.81**	**98.81**	97.68	98.02	98.18	**98.66**
sonar	**73.75**	**73.75**	**73.75**	**73.75**	70.10	70.87	69.23	**75.00**
soyb	**90.69**	**90.69**	**90.69**	**90.69**	79.21	90.98	89.99	**91.42**
spam	**92.36**	**92.36**	**92.36**	**92.36**	90.56	91.09	91.82	**92.43**
splice	**91.50**	**91.50**	**91.50**	**91.50**	86.41	91.33	91.37	**91.59**
ta	**49.01**	**49.01**	**49.01**	**49.01**	46.89	46.89	48.21	**49.67**
titanic	**78.89**	**78.89**	**78.89**	**78.89**	**78.89**	**78.89**	**78.89**	**78.89**
trains	**70.00**	**70.00**	**70.00**	**70.00**	**70.00**	**70.00**	**70.00**	**70.00**
vote	**95.82**	**95.82**	**95.82**	**95.82**	94.48	94.85	95.59	**95.77**
vowel	**81.66**	**81.66**	**81.66**	**81.66**	78.75	79.52	**82.75**	81.82
wave	**74.72**	**74.72**	**74.72**	**74.72**	72.78	73.02	74.66	**74.90**
wine	92.81	92.81	92.81	**92.92**	90.56	88.88	91.91	**92.36**
yeast	**53.94**	**53.94**	**53.94**	**53.94**	53.33	52.95	53.96	**54.04**
zoo	**93.27**	**93.27**	**93.27**	**93.27**	90.30	**93.27**	**93.27**	**93.27**
Average	78.62	78.62	78.62	78.61	76.56	77.72	78.80	79.23
p-values		1	1	0.605		$<$ **0.001**	$<$ **0.001**	$<$ **0.001**
>,=,<		0,52,0	0,52,0	2,49,1		34-3-15	43,3,6	44,2,6

Table 6.B.3: A comparison of curriculum learning, filtering, and boosting for MLPs for each data set.

Data set	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
abalone	27.75	**28.99**	27.82	27.82	27.65	28.30	28.39	28.54
adult	84.86	85.66	83.01	83.38	85.65	85.36	82.71	**85.71**
anneal	99.55	99.55	**99.67**	99.55	99.35	**99.67**	99.44	99.29
arrhyth	70.58	**72.12**	66.81	67.92	69.87	70.58	70.58	70.62
audio	83.19	79.65	**84.51**	**84.51**	83.54	80.53	80.53	80.27
autos	77.07	**82.44**	78.05	77.07	78.63	80.49	80.49	80.49
balance	87.20	86.72	**95.20**	91.84	87.94	90.88	89.76	87.10
breast	69.58	75.17	70.63	70.63	74.62	75.52	75.87	**76.01**
breast-w	95.99	96.28	95.85	96.14	96.82	96.57	96.57	**96.97**
bupa	72.17	**74.20**	70.43	71.01	71.07	70.14	72.17	71.94
carEval	**99.94**	99.71	99.88	99.88	99.80	99.71	99.71	99.77
chess	69.73	**71.03**	55.98	57.88	68.76	64.90	62.28	68.74
KRvKP	99.50	**99.53**	99.44	99.44	99.39	99.37	99.37	99.52
colic	82.34	85.60	82.34	82.34	85.98	85.33	85.60	**86.41**
contact	75.00	**87.50**	75.00	75.00	75.83	**87.50**	**87.50**	**87.50**
credit-a	85.94	**88.12**	84.35	85.51	87.25	86.81	86.96	87.77
credit-g	74.40	78.20	74.50	75.00	76.68	77.10	76.90	**78.90**
derma	97.27	97.54	96.17	96.17	97.81	97.27	97.27	**97.92**
ecoli	85.42	87.50	84.23	83.93	85.42	87.20	87.20	**87.56**
glass	72.90	**73.36**	71.50	71.96	72.06	70.56	70.56	72.43
heart-c	84.82	**86.14**	80.20	83.83	84.55	**86.14**	**86.14**	**86.14**
heart-h	82.65	**84.69**	82.99	81.63	84.01	84.35	84.35	84.56
heart-s	82.96	**85.56**	80.37	80.37	84.37	**85.56**	**85.56**	85.19
hepa	86.45	**88.39**	81.94	82.58	86.97	87.74	87.74	87.10
hypo	97.85	97.83	98.04	97.91	97.99	**98.09**	97.75	97.88
iono	91.74	90.88	**92.31**	92.02	90.88	88.89	88.89	91.00
iris	96.67	97.33	97.33	97.33	96.53	**98.00**	**98.00**	96.67
labor	89.47	89.47	89.47	89.47	**92.98**	89.47	89.47	92.63
letter	89.70	84.68	92.32	88.52	88.79	**93.92**	91.61	88.16
lungCan	46.88	62.50	46.88	46.88	60.00	**65.63**	**65.63**	61.88
lympho	84.46	85.81	84.46	84.46	83.78	86.49	86.49	**87.16**
MagicTel	86.95	86.83	86.77	86.70	**86.95**	86.80	86.94	86.82
nursery	99.78	98.71	**99.96**	**99.96**	99.29	99.70	99.41	99.64
ozone	97.24	97.12	96.41	96.88	97.15	97.24	**97.32**	97.12
pimaDia	77.60	**79.04**	77.34	77.21	77.24	78.26	77.47	77.86
post-op	62.22	71.11	61.11	62.22	71.11	**73.33**	**73.33**	72.44
prim-tu	44.84	51.62	44.54	45.13	46.37	**51.92**	**51.92**	51.39
segment	97.49	97.62	**97.75**	**97.75**	97.52	97.45	97.45	97.64
sick	97.69	97.53	97.67	**97.72**	97.47	97.24	97.27	97.62

Continued on next page

Table 6.B.3: **(Cont.)** A comparison of curriculum learning, filtering, and boosting for MLPs for each data set.

Data set	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
sonar	82.69	85.10	83.17	83.17	84.62	84.13	84.62	**86.25**
soyb	94.29	**94.88**	94.14	94.14	93.65	94.73	94.73	94.70
spam	93.46	93.68	93.63	**93.72**	93.59	93.70	93.70	93.64
splice	95.77	95.92	95.42	95.42	96.16	95.96	95.96	**96.26**
ta	62.91	**68.87**	66.89	65.56	62.52	**68.87**	66.89	67.95
titanic	79.06	78.96	78.92	**79.24**	79.06	79.06	79.06	79.06
trains	80.00	80.00	80.00	80.00	80.00	**90.00**	**90.00**	**90.00**
vote	96.09	**96.78**	94.25	94.25	96.74	95.86	95.86	96.60
vowel	95.76	94.95	98.38	98.08	95.98	**98.48**	97.68	94.99
wave	86.22	86.04	85.26	85.44	**87.19**	86.24	86.20	87.12
wine	97.19	97.75	98.31	98.31	**98.43**	98.31	97.19	98.31
yeast	59.64	61.66	58.15	57.75	60.19	62.06	**62.53**	61.13
zoo	**96.04**	95.05	**96.04**	**96.04**	95.64	**96.04**	**96.04**	95.45
Average	83.21	84.83	82.80	82.82	84.07	85.07	84.87	85.07

Table 6.B.4: A comparison of curriculum learning, filtering, and boosting for DTs for each data set.

Data set	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
abalone	25.44	27.85	26.79	26.79	23.76	**30.81**	29.73	28.99
adult	86.33	86.62	85.37	86.45	85.12	86.89	**86.91**	86.57
anneal	93.83	94.72	95.55	**96.33**	95.43	95.77	95.99	**96.33**
arrhyth	70.71	70.62	72.79	73.89	70.66	**75.66**	74.56	71.15
audio	77.35	80.00	**84.07**	**84.07**	82.57	81.86	81.86	82.04
autos	84.39	84.88	85.85	86.83	84.29	**87.32**	**87.32**	85.46
balance	79.52	79.84	85.92	84.16	78.24	**87.84**	85.76	81.06
breast	74.69	75.87	72.38	75.52	73.08	**77.27**	75.87	76.57
breast-w	94.96	94.99	96.14	96.71	94.54	**97.42**	97.00	95.22
bupa	67.25	69.10	72.75	72.75	66.72	73.91	**75.36**	71.88
carEval	93.99	94.79	96.76	95.08	93.09	**96.82**	95.54	93.14
chess	59.85	62.71	63.44	60.28	58.30	**68.09**	64.78	63.22
KRvKP	99.65	99.62	**99.69**	**99.69**	99.65	99.56	99.59	99.55
colic	85.87	85.87	82.34	85.87	85.16	86.41	**86.68**	85.92
contact	**87.50**	**87.50**	83.33	**87.50**	75.00	**87.50**	**87.50**	**87.50**
credit-a	86.35	86.26	86.67	86.67	86.52	**88.12**	87.68	86.87
credit-g	72.74	75.84	74.20	75.50	72.52	**77.10**	76.00	75.02
derma	93.06	95.30	97.27	96.99	93.33	97.54	**98.36**	93.11
ecoli	83.33	84.88	85.71	85.42	82.26	87.80	**88.69**	83.87
glass	69.53	73.36	77.57	76.17	69.53	**78.50**	75.70	71.78
heart-c	77.69	79.74	82.51	83.50	76.96	**84.82**	84.49	81.45
heart-h	81.43	82.59	80.95	83.67	81.29	**84.01**	83.33	82.65
heart-s	83.19	85.26	82.59	83.33	82.89	**85.93**	84.81	83.85
hepa	80.39	84.52	83.87	83.23	83.61	**88.39**	86.45	85.55
hypo	99.53	99.62	**99.71**	**99.71**	99.57	99.68	99.66	99.66
iono	90.03	90.03	94.02	**94.30**	89.80	93.45	92.02	91.11
iris	94.93	**96.00**	95.33	95.33	94.67	**96.00**	**96.00**	**96.00**
labor	82.11	82.11	89.47	91.23	82.11	89.47	**92.98**	79.65
letter	88.18	88.45	**95.50**	94.58	88.44	95.42	94.16	88.83
lungCan	65.00	**68.75**	56.25	59.38	65.00	**68.75**	**68.75**	68.13
lympho	76.08	80.41	84.46	81.76	75.54	**86.49**	82.43	80.00
MagicTel	85.29	86.11	86.39	87.26	85.02	**87.73**	87.62	86.00
nursery	97.27	97.32	99.64	99.08	98.84	**99.67**	99.14	98.85
ozone	97.07	97.12	**97.32**	97.20	96.77	97.20	97.24	97.12
pimaDia	75.03	76.43	75.65	76.04	75.03	**78.26**	**78.26**	77.89
post-op	71.11	71.11	71.11	71.11	71.11	71.11	**72.22**	72.00
prim-tu	43.13	46.67	41.89	44.84	42.48	**49.85**	**49.85**	46.37
segment	97.05	97.21	98.31	98.01	96.99	**98.40**	97.84	97.17
sick	98.83	98.82	98.99	**99.07**	98.82	98.99	98.65	98.45

Continued on next page

Table 6.B.4: **(Cont.)** A comparison of curriculum learning, filtering, and boosting for DTs for each data set.

Data set	Orig	$IH_{.75}$	AB	MB	CL	$AB_{.75}$	$MB_{.75}$	$CL_{.75}$
sonar	75.77	75.19	83.17	83.17	76.35	**85.58**	85.10	77.31
soyb	91.89	92.53	94.29	93.70	91.22	**95.17**	94.73	92.09
spam	92.93	93.57	95.35	95.37	92.56	95.61	**95.63**	93.34
splice	94.24	94.48	95.02	94.61	92.15	95.08	**95.11**	93.22
ta	63.84	67.68	63.58	60.93	63.05	64.24	54.97	**67.81**
titanic	**79.06**	79.05	**79.06**	78.92	78.89	**79.06**	78.92	**79.06**
trains	**90.00**	**90.00**	**90.00**	**90.00**	**90.00**	**90.00**	**90.00**	**90.00**
vote	96.55	**97.10**	96.09	95.86	96.00	96.78	96.55	96.55
vowel	84.67	84.97	93.43	92.12	84.46	**94.34**	92.42	84.65
wave	76.07	78.51	81.84	82.62	75.70	83.22	**83.44**	77.35
wine	93.60	93.60	**97.19**	**97.19**	93.03	**97.19**	**97.19**	92.36
yeast	58.06	59.78	59.03	60.51	57.88	**64.02**	63.75	62.16
zoo	94.06	95.05	96.04	95.05	93.27	97.03	97.03	**97.43**
Average	81.93	83.08	83.90	84.14	81.52	85.64	85.03	83.26

6.C Methodology for Hyper-Parameter Optimization

The method for selecting hyper-parameters for MLPs, DTs, and curriculum learning are included here for reproducibility. We followed the process outlined by [1] for hyper parameter optimization using a random search of the parameter space. The premise of using a random search for parameter selection is that most machine learning algorithms have very few parameters that considerably affect the final model while most of the other parameters have little to no effect on the final model. However, different parameters considerably affect the final model for different data sets. Random search provides a greater variety of the parameters that considerably affect the model, thus, allowing for better parameter selection of these parameters. The random values were sampled from distributions that give appropriate and reasonable values for each parameter. For example, having a multilayer perceptron with 0 or 10,000 hidden nodes is not as reasonable as having 10 hidden nodes. We used 10 random parameter settings for each learning algorithm, exploring more of the effects of each parameter than a typical grid search, which would only evaluate 3-4 parameter settings to maintain reasonable computational costs. Table 6.C.1 shows which parameter values were optimized and the distribution that they were sampled from. The distributions that were sampled from are:

Normal Distribution ($\sim \mathcal{N}(\mu,\sigma^2)$). A value drawn from a normal distribution with mean μ and variance σ^2. The values were chosen such that reasonable values were drawn but also allowing for a search of the hyper-parameter space. For cases were the values were outside the permissible range (i.e. between 0 and 1), the value was wrapped around (if -0.03 was chosen it would become 0.03).

Geometrically ($\sim \mathcal{G}(\mathbf{min},\mathbf{max})$). Sampling a hyper-parameter value geometrically from min to max means drawing a value uniformly in the log domain between $log(min)$ and $log(max)$, exponentiating to get a number between A and B, and then rounding to the nearest integer. This distribution favors values that are closer to the min value.

Discrete Uniform Distribution ($\sim \mathcal{U}$). A value drawn uniformly gives equal probability to each possible value.

The exact values for the distribution parameters (i.e. μ and σ^2 for the normal distribution) are not critical as long as the values are reasonable and the distribution provides sufficient variability to search the hyper-parameter space.

Table 6.C.1: The parameters that were optimized using a random search and the distributions from which the parameter values were drawn.

Multilayer Perceptron trained with Back Propagation (MLP)

Parameter	Distribution
Learning Rate	$\sim \mathcal{N}(0.3,0.167))$
Momentum	$\sim \mathcal{N}(0.2,0.167)$
Number of hidden nodes	$\sim \mathcal{G}(2,124)$
Whether to decay the learning rate	$\sim \mathcal{U}$

Decision Tree (DT)

Parameter	Distribution
Whether to prune the tree	$\sim \mathcal{U}$
If the tree is pruned, the confidence value	$\sim \mathcal{N}(0.3, 0.167)$
Minimum number of items in each leaf node	$\sim \mathcal{G}(1,32)$
Whether to use Laplacian smoothing	$\sim \mathcal{U}$

Curriculum Learning

Parameter	Distribution
Initial Complexity Level	$\sim \mathcal{N}(0.5, 0.125)$
Number of epochs to train (for MLPs)	$\sim \mathcal{N}(100, 50)$

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

Chapter 7

Becoming More Robust to Label Noise with Classifier Diversity

In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2015)*,to appear, 2015.

Abstract

It is widely known in the machine learning community that class noise can be (and often is) detrimental to inducing a model of the data. Many current approaches use a single, often biased, measurement to determine if an instance is noisy. A biased measure may work well on certain data sets, but it can also be less effective on a broader set of data sets [22]. In this paper, we conduct a large empirical evaluation of noise handling techniques; examining 12 noise handling techniques on a set of 54 data sets and 5 learning algorithms. The chosen set of noise handling techniques includes biased and ensembled approaches. Included in the investigation is the proposed *noise identification using classifier diversity* (*NICD*). NICD lessens the bias of the noise measure by selecting a diverse set of classifiers to determine which instances are noisy. We examine NICD as a technique for filtering, instance weighting, and selecting the base classifiers of a voting ensemble. We find that lessening the bias of the noise handling techniques significantly improves performance over a broad set of data sets.

7.1 Introduction

The goal of supervised machine learning is to induce an accurate generalizing function from a set of labeled training instances. However, most real-world data sets are noisy. Generally, two types of noise are considered: attribute noise and label noise. Previous work has found

that label noise is generally more harmful than attribute noise [17, 29]. The consequences of label noise include 1) a deterioration of classification performance, 2) increased learning requirements and model complexity, and 3) a distortion of observed frequencies [8]. Filtering noisy instances generally increases classification accuracy [10, 23]. Knowing which instances are noisy and/or detrimental is non-trivial as, in most cases, all that is known about a task is contained in the set of training instances.

Prior work has examined handling label noise using a variety of approaches that are generally specific to, or inspired by, an individual learning algorithm or information theoretic measure. One commonly used approach removes the instances that are misclassified by a learning algorithm [26]. Although such an approach is biased towards the learning algorithm that is used, it has generally been shown to work well on the examined data sets and learning algorithms especially with the addition of artificial noise. However, it has also been shown that in some cases, using a noise handling technique *reduces* the classification accuracy – especially in the absence of artificial noise [22]. As ensembles often perform better than any one of its constituent base classifiers in classification [5], other prior work has used ensemble techniques to improve handling class noise [2]. For an ensemble to be more accurate than any individual classifier of the ensemble, the base classifiers need to be accurate (better than random) and diverse [12]. As in classification, using an ensemble technique for identifying noise implicitly lessens the dependence on a single hypothesis.

None of the previous work in noise handling has explicitly focused on selecting diverse base classifiers when using an ensemble approach. We propose *noise identification using classifier diversity* (NICD). NICD builds on previous work that use ensembles to identify noisy instances by using classifier diversity. NICD first explicitly selects a set of diverse learning algorithms where diversity is determined by the predictions of the classifiers. The diversity lessens the dependence of a noise measure on a specific hypothesis. Without diversity, the same hypothesis could be over-represented if two hypotheses always classify the same way.

We examine using the set of diverse learning algorithms to 1) filter the instances, 2) weight the instances, and 3) as the base classifiers for a voting ensemble.

One of our primary contributions is a comprehensive empirical evaluation of noise handling techniques on a set of 54 data sets and 5 learning algorithms: C4.5, 5-NN, MLP trained with backpropagation, random forest, and RIPPER. We examine NICD and 9 other noise handling techniques, and a voting ensemble composed of diverse base classifiers We find that NICD has the most significant impact on the performance of a classifier across a *broad* set of data sets, learning algorithms and noise levels – demonstrating a robustness to label noise. The term *broad* refers to the characteristic that the noise handling method was *not* developed specifically for a given set of data sets and learning algorithms. Overall, using NICD in a voting ensemble to select a diverse set of base classifiers achieves significantly higher classification accuracy then using a standard noise handling technique. This finding is in contrast to previous work [2] that found that a voting ensemble does not perform as well as filtering in the presence of noisy data. Related works will be discussed through out the paper.

7.2 Noise Identification using Classifier Diversity

Noise identification using classifier diversity (NICD) is the process of selecting and using a diverse set of learning algorithms and then using this set of learning algorithms in the learning process to handle class noise. In this paper, NICD uses the diverse set of learning algorithms for 1) filtering, 2) weighting, and 3) as the set of base classifiers for a voting ensemble.

7.2.1 Identifying Noisy Instances

The first step in any noise handling technique is to identify noisy instances. A variety of approaches have been used to remove noisy instances, often using heuristics or biases from current learning algorithms or information theoretic measures.

In this paper, the probability $p(y|x)$ that an instance $\langle x,y\rangle$ will be correctly classified is used to determine how noisy each training instance is. In practice, $p(y|x)$ is generally estimated using a specific hypothesis h induced from a learning algorithm (i.e. $p(y|x) \approx p(y|x,h)$) – which is a biased approximation. The dependence of $p(y|x)$ on a specific h can be removed by summing over all possible hypotheses h in $\mathcal{H}$ and multiplying each $p(y|x,h)$ by the prior $p(h)$:

$$p(y|x) = \sum_{h \in \mathcal{H}} p(y|x,h)p(h) \tag{7.1}$$

where $p(h)$ denotes the probability of the hypothesis h before observing the training data. This formulation is infeasible, though, because 1) it is not practical (or possible) to sum over the set of all hypotheses, 2) calculating $p(h)$ is non-trivial, and 3) not all learning algorithms produce a probability distribution. The use of probabilistic generative models, such as the kernel Fisher discriminant algorithm, can mitigate these issues [15]. However, because discriminative models generally have a lower asymptotic error than generative models for classification tasks [18] and because our examination focuses on the behavior of discriminative models on classification tasks, we use discriminative models to model $p(y|x)$.

To approximate $p(y|x)$ independent of a specific h, we estimate $p(y|x)$ using a diverse set of learning algorithms. The reasoning for using a diverse set of learning algorithms is to lessen the bias that each algorithm has for handling noise. The diversity of the learning algorithms refers to the learning algorithms not having the same classification for all of the instances and is determined using unsupervised meta-learning (UML) [16]. UML clusters learning algorithms based on their performance. One of the underlying ideas of UML is that by clustering the learning algorithms by their performance similarity, only one of the member learning algorithms from a cluster needs to be examined to gain an intuition of how the other learning algorithms in the cluster will perform. Thus, we use UML to identify clusters of similar learning algorithms and then select a learning algorithm from each cluster. UML first uses classifier output difference (COD) [19] to measure the diversity between learning algorithms. COD measures the distance between two learning algorithms as the probability

that the learning algorithms make different predictions:

$$COD(l_1,l_2) = \frac{|\{x : l_1(x,T) \neq l_2(x,T)\}|}{N}$$

where l_j is a learning algorithm, $l_j(x,T)$ returns the prediction for x from l_j trained on T, and N is the number of test instances. UML then clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. UML uses 20 learning algorithms from Weka [11] selected to be representative of several model classes: Bayesian-based, function-based, instance-based, tree-based, and rule-based. The learning algorithms are implemented with their default parameters. The COD values for the learning algorithms are then calculated using 129 data sets: 72 UCI data sets [7], 45 from the Gene Expression Machine Learning Repository [6], and 12 data sets from ASUs Multi-class Protein Fold Recognition data [25]. A cut-point of 0.18 was chosen to create 9 clusters and a representative algorithm from each cluster was chosen to create a diverse set of learning algorithms. The set of learning algorithms $\mathcal{L}$ that are used to approximate $p(y|x)$ are listed in Table 7.1.

Using 10-fold cross-validation on the training set, $p(y|x)$ is first approximated for each training instance (the instance $\langle x,y \rangle$ is *not* used to induce the model h). Using the set of hypotheses induced by the diverse set of learning algorithms in $\mathcal{L}$, $p(y|x)$ is approximated as:

$$p(y|x) \approx p(y|x,\mathcal{L}) = \frac{1}{|\mathcal{L}|}\sum_{j=1}^{|\mathcal{L}|} p(y|x,l_j(T)) \tag{7.2}$$

where $l_j(T)$ is the hypothesis from the j^{th} learning algorithm trained on training set T. Following Equation 7.1, $p(h)$ is estimated as $\frac{1}{|\mathcal{L}|}$ for the hypotheses generated from training the learning algorithms in $\mathcal{L}$ on T and as zero for all of the other hypotheses in $\mathcal{H}$ as it is nontrivial to compute $p(h)$ for many learning algorithms. Since not all learning algorithms produce a probability distribution and to be consistent, we use the Kronecker delta function $\delta(h(x),y)$ instead of $p(y|x,l_j(T))$ (from Equation 7.2) to produce a real-valued score for

Table 7.1: Set of diverse learning algorithms sampled from the clusters generated by UML.

Learning Algorithms
* Multilayer Perceptron trained with Back Propagation (MLP)
* Decision Tree (C4.5) [20]
* Locally Weighted Learning (LWL)
* 5-Nearest Neighbors (5-NN)
* Nearest Neighbor with generalization (NNge)
* Naïve Bayes (NB)
* RIpple DOwn Rule learner (RIDOR)
* Random Forest (RandForest)
* Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

$p(y|x,\mathcal{L})$. The Kronecker delta function is:

$$\delta(h(x),y) = \begin{cases} h(x) = y & : 1 \\ h(x) \neq y & : 0 \end{cases}$$

where h is an induced hypothesis and $h(x)$ returns the predicted class given the input features x. Thus, here $p(y|x)$ is not a true probability but more of a confidence score.

7.2.2 Handling Noisy Instances

After noisy instances have been identified, the next step is to handle the noisy instances. In this paper, NICD uses the diverse set of learning algorithms for 1) filtering, 2) weighting, and 3) as the set of base classifiers for a voting ensemble.

For filtering the training instances using $p(y|x,\mathcal{L})$ ($\mathcal{L}$-filter), training instances that are misclassified by 50 percent of the learning algorithms in the $\mathcal{L}$ are filtered from the training set. The remaining training instances are then used by the learning algorithm to induce a model of the data. Note that other percentages could also be used. We found that 50% generally produces good results compared to values of 70% and 90%. In practice a validation set is often used to determine the percentage that would be used.

Table 7.2: How instance weighting is integrated into the examined learning algorithms.

Learn Alg	Orig	NICD
MLP	$(t-o)f'(net)$	$p(y\|x)(t-o)f'(net)$
Random Forest	Uniform dist	Weighted by $p(y\|x,\mathcal{L})$
C4.5, 5-NN, RIPPER	Count number of instances, i.e. $\frac{\sum_{c_i} 1}{\sum_T 1}$	Sum $p(y\|x,\mathcal{L})$ $\frac{\sum_{c_i} p(y_i\|x_i)}{\sum_T p(y_i\|x_i)}$

For weighting the training instances with $p(y|x,\mathcal{L})$ ($\mathcal{L}$-weighting), a training instance is weighted in the investigated learning algorithms as follows:

MLPs. For MLPs trained with backpropagation, the error $((t-o)f'(net))$ is scaled by $p(y|x,\mathcal{L})$ where $(t-o)$ is the difference between the target value and the output of the network, $f'(net)$ is the derivative of the activation function f and net is the sum of the product of each input i_j and its corresponding weight w_j: $net = \sum_j w_j i_j$.

Random Forests. For Random Forests, the distribution for selecting instances in the random trees is weighted by $p(y|x,\mathcal{L})$ rather than being uniformly weighted.

Other algorithms. For the other learning algorithms that keep track of counts, each instance is weighted by $p(y|x,\mathcal{L})$.

Table 7.2 summarizes how an instance is weighted. Although we estimate $p(y|x)$ as $p(y|x,\mathcal{L})$, other methods could be used.

For an ensemble using $p(y|x,\mathcal{L})$ (the $\mathcal{L}$-ensemble), the ensemble is composed of the learning algorithms in the set $\mathcal{L}$. Each learning algorithm has an equally-weighted vote for the classification of an instance.

7.3 Other Noise Handling Approaches

There are many techniques for handling noise. In this section, we describe the noise handling methods that are included in our examination. The compared noise handling techniques are listed below with a brief explanation. For additional details, consult the cited work. All

of the previous techniques were used with their default parameters as implemented in the KEEL toolkit [1] except for the biased approaches, RENN and PWEM.

7.3.1 Filtering Methods

Listed below are the investigated instance filtering methods that include both biased (using a single noise identification measure) and less biased (ensembled noise identification measures) methods. In general, the methods differ in how noisy instances are identified.

Biased Filter. For a biased filtering approach, instances that are misclassified by the same learning algorithm that are being used to induce a model of the data is filtered from the training set. An instance from the training set is determined to be misclassified using 10-fold cross-validation on the training set.

Repeated-edited nearest neighbor (RENN) [26]. RENN repeatedly removes the instances that are misclassified by a nearest neighbor classifier and has been shown to produce good results. Here we set the number of nearest neighbors to 5.

Saturation filter [10]. The saturation filter is based on the premise that removing noisy instances reduces the complexity of the least correct hypothesis (CLCH) while removing correctly labeled instances does not. The instance whose removal from the training set decreases the CLCH the most is filtered from the training set. This process continues until the CLCH can no longer be reduced or there are no training instances.

Classification filter [9]. The classification filter removes instances that are misclassified by a learning algorithm using 10-fold cross-validation on the training set. The default learning algorithm is a 1-nearest neighbor.

Ensemble filter [2]. The ensemble filter removes instances that are misclassified by $n\%$ of the base classifiers. For the ensemble filter, the authors chose three well-known learning algorithms (C4.5, IB1, and thermal linear machine [3]). They then examined removing instances that were misclassified by the majority or all of the learning algorithms. In this paper, instances are removed that are misclassified by all of the learning algorithms following

Brodley and Friedl that the majority filter tends to perform better than the consensus filter on average. The $\mathcal{L}$-filter is distinguished from the ensemble filter in that the $\mathcal{L}$-filter uses more learning algorithms and that the learning algorithms were specifically chosen to be diverse.

Automatic noise removal filter (ANR) [28]. ANR estimates the probability of each possible class for the training instances. The probabilities are used during training such that an instance that has a low probability for its assigned class is "corrected" to the class that has the highest probability. Instances that were corrected during training are then filtered from the data set.

Cross-validated committees filter [27]. The cross validated committees filter partitions a data set into n subsets of approximately equal size and a learning algorithm is induced n times, each time leaving out one of the subsets from the training data. Instances that are misclassified by all of the n classifiers are then filtered from the training set. The base classifier used is a decision tree. The diversity of the learning algorithms is determined by the data that is used to induce the base classifier rather than using different algorithms.

Iterative-partitioning filter [14]. The iterative partitioning filter first partitions the training data into n subsets and a model is induced on each subset. Instances that are misclassified by all of the induced models are filtered. This process is repeated until the number of noisy instances that are removed is less than 1% of the size of the original training set. The base classifier used is a decision tree trained using C4.5. We use the consensus filter (misclassified by all of the models) rather than the majority filter following their findings that the majority filter did not significantly improve the performance of the induced models. Like the cross-validated committees filter, this filter uses an ensemble based on training a learning algorithm on different subsets of the training data.

7.3.2 Weighting Methods

Listed below are the investigated instance weighting methods. Instance weighting is much less explored compared to filtering, thus, there are fewer methods. The following methods differ from $\mathcal{L}$-weighting in how the weights are calculated.

Biased Weighting. For a biased weighting approach, $p(y|x)$ is approximated as $p(y|x,h)$ where the hypothesis h is induced by the same learning algorithm that is used to induce a model of the data. To get a real-value for biased instance weighting from a single hypothesis, we compute a classifier score for each instance from the learning algorithm that induces h. The quantity $p(y|x,h)$ is estimated using 10-fold cross validation on the training set. Below, we present how we calculate the classifier scores for the investigated learning algorithms.

Multilayer Perceptron (MLP): For multiple classes, each class from a data set is represented with an output node. After training with backpropagation, the classifier score is the largest value of the output nodes normalized between zero and one:

$$\hat{p}(y|x) = \frac{o_i(x)}{\sum_i^{|Y|} o_i(x)}$$

where y is a class from the set of possible classes Y and o_i is the value from the output node corresponding to class y_i.

Decision Tree: To calculate a classifier score, an instance first follows the induced set of rules until it reaches a leaf node. The classifier score is the number of training instances that have the same class as the examined instance divided by all of the training instances that also reach the same leaf node.

5-NN: The classifier score is the percentage of the nearest-neighbors that agree with the class label of an instance.

Random Forest: Random forests return the class counts from the leaf nodes of each tree in the forest. The counts for each class are summed together and then normalized between 0 and 1.

RIPPER: RIPPER returns the percentage of training instances that are covered by a rule and share the same class as the examined instance.

Obviously, a classifier score does not produce a true probability. However, the classifier scores approximate the confidence of an induced model for the class label of an instance.

Pair-wise expectation maximization (PWEM) [21]. PWEM weights each instance using the EM algorithm. First each data set is binarized. For each pair of classes, the instances that belong to the two classes are clustered using EM where the number of clusters is determined using the Bayesian Information Criterion [13]. Given the $Y-1$ clusterings (Y is the number of classes), $p(y|x)$ is calculated as:

$$p(y|x) = \sum_{\theta} p(\theta)p(y|x,\theta) = \sum_{\theta} p(\theta) \sum_{c=1}^{k} p(y|c_k,\theta)p(c_k|x,\theta) \tag{7.3}$$

where θ is a clustering model induced using the EM algorithm, c is a cluster in θ and k is the number of clusters.

7.4 Methodology

In this section, we present our experimental methodology. We conduct a large empirical investigation allowing for general observations to be made. We examine noise handling using C4.5, 5-NN, MLP trained with backpropagation, random forest, and RIPPER on a set of 54 data sets from the UCI data repository [7]. Table 7.1 shows the data sets used in this study including the number of instances, number of attributes, and attribute type. The LWL, NNge, and Ridor learning algorithms are not used for analysis because they do not scale well, not finishing due to memory overflow or large amounts of running time on the larger data sets.[1]

Each method for handling noise is evaluated by averaging the results from ten runs of each experiment. Following the methodology used to evaluate PWEM [21], for each experiment,

[1]For the data sets on which the learning algorithms did finish, the effects of filtering on LWL, NNge, and Ridor are consistent with the other algorithms.

Table 7.1: Datasets used in our study as well as the number of instances, the number of attributes, and the attribute type.

Name	#Inst	#Att	Att Type	Name	#Inst	#Att	Att Type
abalone	4177	8	Mixed	hypothyroid	3772	29	Cat
adult-census	32561	14	Cat	ionosphere	351	34	Mixed
anneal.ORIG	898	38	Cat	iris	150	4	Cat
arrhythmia	452	279	Cat	labor	57	16	Mixed
audiology	226	69	Mixed	letter	20000	16	Mixed
autos	205	25	Real	lungCancer	32	56	Real
badges2	294	10	Mixed	lymphography	148	18	Mixed
balance-scale	625	4	Mixed	MagicTele	19020	10	Real
balloons	20	4	Mixed	nursery	12963	8	Cat
breast-cancer	286	9	Real	ozone	2536	72	Cat
breast-w	699	9	Real	pimaDiabetes	768	8	Cat
bupa	345	6	Cat	post-opPatient	90	8	Mixed
carEval	1728	6	Mixed	primary-tumor	339	17	Real
chess	28056	6	Mixed	segment	2310	19	Cat
chess-KRvKP	3196	36	Cat	sick	3772	29	Real
colic	368	25	Real	sonar	208	60	Cat
colon	63	2000	Real	soybean	683	35	Cat
contact-lenses	24	4	Cat	spambase	4601	57	Cat
credit-a	690	15	Real	splice	3190	60	Mixed
credit-g	1000	24	Cat	TA	151	5	Mixed
dermatology	366	34	Real	titanic	2201	3	Real
ecoli	336	7	Cat	vote	435	16	Real
glass	214	9	Real	vowel	990	13	Mixed
heart-c	303	13	Real	waveform-5000	5000	40	Mixed
heart-h	294	13	Mixed	wine	178	13	Cat
heart-statlog	270	13	Mixed	yeast	1484	8	Cat
hepatitis	155	19	Mixed	zoo	101	17	Mixed

the data is shuffled and then split into 2/3 for training and 1/3 for testing. The training and testing sets are stratified. Random noise is then introduced by randomly changing the class label of $n\%$ of the training instances where a new label is chosen uniformly from the possible class labels (noisy completely at random [8]). The random noise levels are examined at 0%, 10%, 20%, 30%, and 40%. As suggested by Demšar [4], statistical significance is determined using the Friedman test to reject the null hypothesis that algorithms perform the same at a 0.05 confidence level. If the null hypothesis is rejected, statistically significant differences between the methods are identified using the Nemenyi post-hoc test.

7.5 Results

In this section, we present the results of our experiments. We compare $\mathcal{L}$-filtering with the biased-filter and seven other filtering approaches presented in Section 7.3. $\mathcal{L}$-weighting is compared with biased-weighting and PWEM. In some cases, an algorithm did not finish running on all of the data sets. For example, since the saturation filter iteratively removes one instance, it requires large amounts of memory/time to run on large data sets and did not finish in some cases. Rather than remove the large data sets from the entire examination, we include the comparison on the data sets on which the algorithms did finish. For the tables in this section, the algorithm in the first row is the baseline algorithm that the algorithms in the subsequent rows are compared against. The values in the "better,equal,worse" (or "b,e,w") rows represent the number of times that the accuracy from the baseline algorithm is greater than, equal to, or less than the compared algorithm.

7.5.1 Application of Noise Handling without Artificial Noise

Many of the previous works in noise handling were inspired by and, often, biased towards a given learning algorithm or measure. Most previous work added artificial noise to the data sets to evaluate their techniques and showed significant improvements on the data sets with artificial noise. Due to the biased nature of the noise handling techniques, it is not surprising that the broad application of a noise handling approach to data sets without artificial noise has been shown be detrimental in some cases [22, 24]. We briefly re-examine the application of noise handling to a broad set of data sets and learning algorithms. Table 7.1 compares no noise handling with the considered noise handling techniques averaged over all 54 data sets with *no artificially added noise*. The value in parentheses is the average rank for the technique. The percent reduction in error (%RE) is the percentage of error that is reduced when a noise handling technique is used compared to the error obtained with no

noise handling:

$$\%RE = \frac{noise - orig}{100 - orig} * 100$$

where *noise* is the accuracy achieved when using a noise handling technique and *orig* is the accuracy obtained when not using a noise handling technique. The values in parentheses represent the average rank for the method. There is only one case for which noise handling significantly increases the classification accuracy – $\mathcal{L}$-Weighting in a MLP. On the other hand, RENN, ANR, and the classification filter significantly *decrease* the classification accuracy for many/all of the investigated learning algorithms. This is an important example that highlights a point that is often overlooked in the noise handling literature – noise handling can be detrimental if used in all cases. In most cases, however, on average, there is no significant difference when using a noise handling technique with no added noise. Previous work has generally considered only a few data sets where noise handling is beneficial.

Most previous work in noise handling has shown significant improvement when there is a high amount of noise in the data. As there is no way to determine if an instance is noisy or mislabeled without the use of a domain expert, most previous work has added artificial noise to show the impact of noise and how handling noise improves the accuracy. Generally, once there are large amounts of noise, using a noise handling approach significantly increases the classification accuracy. In addition to not adding artificial noise, we also examine the effectiveness of the noise handling techniques with the addition of artificial noise. Figure 7.1 graphs the percent reduction in error for each learning algorithm and noise handling method for the considered artificial noise levels of 0%, 10%, 20%, 30%, and 40%. The x-axis represents no change in error from not using a noise handling technique, negative values represent an increase in error when using a noise handling technique and positive values represent a decrease in error compared to not using noise handling. In general, the reduction in the percentage of error increases as the amount of noise increases. With the exception of ANR and the classification filter all of the noise handling techniques significantly increase the classification accuracy as the noise level increases. In the cases of ANR and the classification

Table 7.1: The results of the 5 considered learning algorithms using the investigated noise handling approaches with no artificial noise added to the data sets averaged over the 54 data sets. Bold accuracy values represent cases where noise handling significantly *increases* the accuracy; underlined, gray cells represent cases where noise handling significantly decreases the accuracy.

	C4.5	5-NN	MLP	RF	RIP
Orig	79.31(6.8)	79.37(5.7)	81.67(6.7)	81.18(6.4)	78.35(6.3)
NICD: $\mathcal{L}$-Weighted	78.36(6.6)	78.72(6.0)	**82.26(4.3)**	80.82(6.5)	77.86(6.9)
%RE	-0.05	-0.03	0.03	-0.02	-0.02
better,equal,worse	26,1,27	27,4,23	18,3,33	28,2,24	26,2,26
Biased-Weighted	79.29(6.5)	78.34(7.8)	81.49(6.5)	80.94(6.8)	77.98(7.0)
%RE	0.00	-0.05	-0.01	-0.01	-0.02
better,equal,worse	23,7,24	32,7,15	23,5,26	26,4,24	29,4,21
PWEM	76.41(8.1)	78.02(7.7)	82.79(6.6)	81.51(7.7)	74.17(8.9)
%RE	-0.14	-0.07	0.06	0.02	-0.19
better,equal,worse	30,3,21	33,3,18	23,3,28	34,1,19	39,1,14
NICD: $\mathcal{L}$-Filter	79.55(5.5)	79.40(4.9)	81.80(6.2)	81.66(5.3)	78.98(5.4)
%RE	0.01	0.00	0.01	0.03	0.03
better,equal,worse	25,11,18	23,9,22	23,4,27	28,2,24	27,5,22
Biased-Filter	79.34(6.6)	76.99(9.2)	81.39(6.9)	81.16(6.3)	77.20(7.9)
%RE	0.00	-0.12	-0.02	0.00	-0.05
better,equal,worse	25,7,22	35,4,15	24,10,20	21,12,21	30,7,17
RENN	76.83(8.5)	76.99(9.2)	78.80(9.4)	78.20(9.4)	76.65(8.2)
%RE	-0.12	-0.12	-0.16	-0.16	-0.08
better,equal,worse	32,3,19	35,4,15	38,1,15	35,2,17	34,2,18
ANR	67.25(9.3)	67.64(9.6)	69.17(9.4)	68.20(9.8)	68.18(10.0)
%RE	-0.58	-0.57	-0.68	-0.69	-0.47
better,equal,worse	34,3,17	34,7,13	37,2,15	36,3,15	40,1,12
ClassificationFilter	73.65(8.2)	73.09(8.6)	75.65(9.4)	75.11(8.2)	72.48(8.2)
%RE	-0.27	-0.30	-0.33	-0.32	-0.27
better,equal,worse	32,3,19	36,4,14	34,2,18	32,1,21	34,3,17
CVCommittees	79.41(6.5)	79.37(5.3)	82.33(5.6)	81.72(5.3)	78.83(5.3)
%RE	0.00	0.00	0.04	0.03	0.02
better,equal,worse	18,10,26	22,14,18	20,4,30	21,2,31	16,8,30
EnsembleFilter	79.53(5.5)	79.06(5.6)	81.62(7.0)	80.91(6.8)	78.87(5.6)
%RE	0.01	-0.01	0.00	-0.01	0.02
better,equal,worse	24,2,28	32,1,21	28,1,25	31,1,22	23,3,28
IterPartitionFilter	79.31(6.7)	79.27(5.3)	82.18(6.0)	81.51(5.7)	78.78(5.6)
%RE	0.00	0.00	0.03	0.02	0.02
better,equal,worse	16,13,25	23,10,21	21,4,29	24,4,26	23,5,26
SaturationFilter	78.70(6.6)	79.08(6.3)	81.07(7.5)	80.28(7.3)	78.46(6.1)
%RE	-0.01	-0.01	-0.03	-0.03	0.02
better,equal,worse	21,9,20	24,9,18	26,4,20	29,2,19	22,7,21

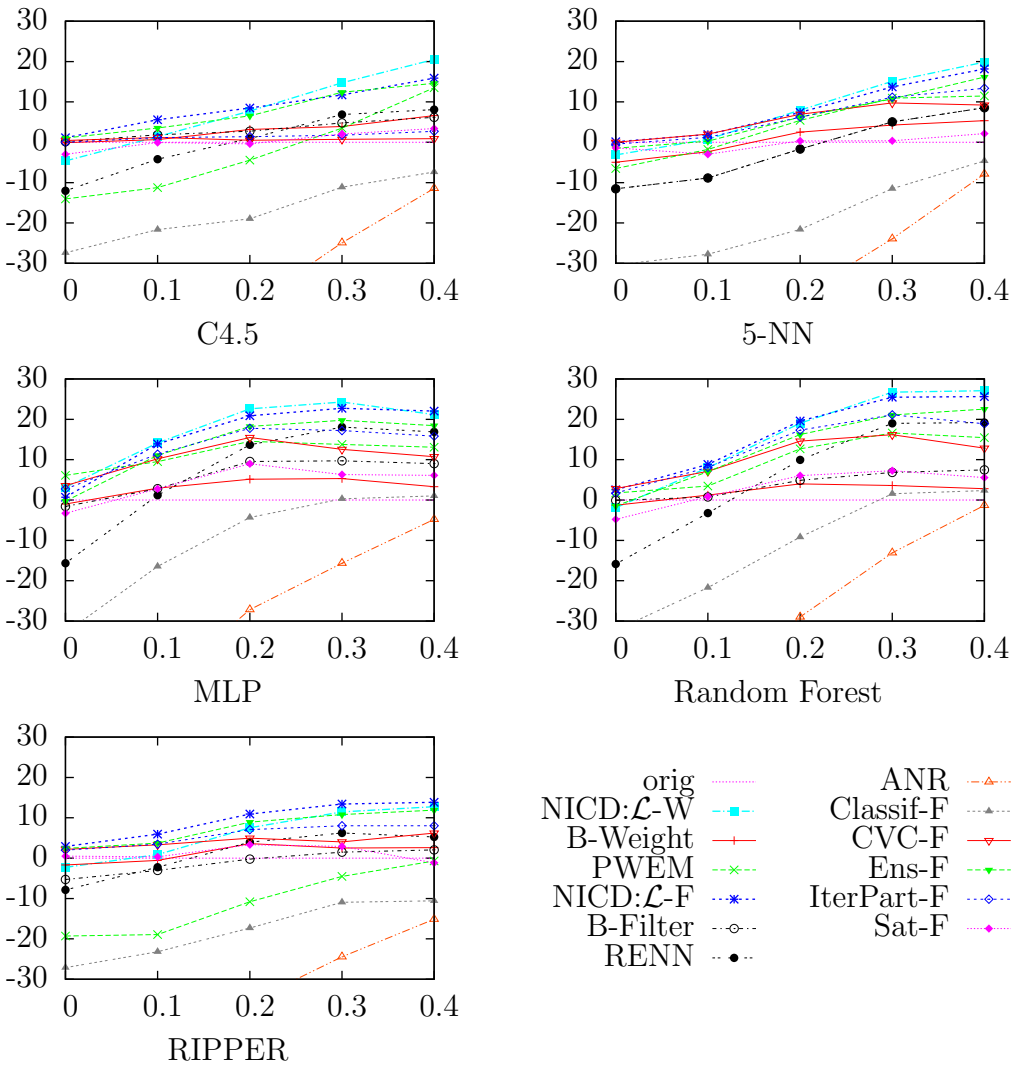

Figure 7.1: The average percent reduction in error (y-axis) for each learning algorithm when using the investigated noise handling techniques for various noise levels (x-axis).

filter, a broad application of the techniques is detrimental although they may be beneficial in specific tasks. The decrease in error is greatest for $\mathcal{L}$-weighting and $\mathcal{L}$-filtering for each learning algorithm. Recall that the learning algorithms in $\mathcal{L}$ were chosen to be diverse so as to represent more of the hypothesis space $\mathcal{H}$. This suggests that a better estimation of $p(y|x)$ produces more significant results for instance weighting and filtering. This is shown empirically as $\mathcal{L}$-weighting and $\mathcal{L}$-filtering have the greatest reduction in error for each learning algorithm (Figure 7.1 and Table 7.2). However, there is an obvious trade-off since obtaining a more accurate estimate of $p(y|x)$ is more computationally expensive.

Examining the performance of the considered learning algorithms, we note that MLPs and random forests generally achieve the highest classification accuracy and may be the most tolerant to the noise inherent in each data set. However, MLPs and random forests also appear to be the least robust as they obtain the lowest average classification accuracy when more than 10% of the instances are corrupted with noise. With no artificial noise, MLPs and random forests achieve about 81% accuracy. With 20% artificial noise, the average accuracy decreases to about 72%. Bear in mind, however, that the default number of trees in the random forest is 10. With more trees, a random forest may be more robust to noise. On the other hand, C4.5, 5-NN, and RIPPER achieve an average accuracy of about 79% with no artificial noise and an average accuracy of about 74% with 20% artificial noise. With high degrees of noise, the built-in noise handling mechanisms of learning algorithms become more beneficial.

7.5.2 Comparison of Noise Handling Techniques

Comparison of Weighting Methods

We compare the $\mathcal{L}$-weighting with the weighting techniques that do not use classifier diversity. Table 7.3 compares $\mathcal{L}$-weighting using the learning algorithms listed in Table 7.1 with biased weighting (B-W), and PWEM. The values in bold represent the cases where $\mathcal{L}$-weighting achieves significantly higher classification accuracy than the compared weighting method. The row "Rank" refers to the average rank of the method. $\mathcal{L}$-weighting significantly outperforms the other weighting schemes in most cases: 24 out of the 25 cases for PWEM and 20 out of the 25 for B-W. Biased-weighting and PWEM never achieve a significantly higher classification accuracy than $\mathcal{L}$-weighting.

Comparison of Filtering Methods

The results are similar for filtering. Table 7.4 compares $\mathcal{L}$-filtering ($\mathcal{L}$-F) with 1) the biased filter (B-F), 2) the cross-validated committees filter (CVC), 3) the ensemble filter (Ens),

Table 7.2: The percent reduction in error for the investigated noise handling techniques for each learning algorithm. The values correspond with Figure 7.1.

C4.5

	L-W	B-W	PEM	L-F	B-F	REN	ANR	CF	CVC	Ens	IPF	Sat
0	-4.6	-0.1	-14.0	1.1	0.1	-12.0	-58.3	-27.4	0.4	1.0	0.0	-3.0
10	1.4	0.7	-11.2	5.6	1.9	-4.2	-48.4	-21.6	1.2	3.5	1.1	-0.1
20	7.7	3.2	-4.4	8.5	2.7	0.9	-40.6	-19.0	0.5	6.5	1.4	-0.4
30	14.7	3.9	3.5	11.7	4.8	6.9	-24.9	-11.1	0.7	12.4	1.7	2.0
40	20.5	6.6	13.5	15.9	6.2	8.1	-11.4	-7.4	0.9	14.6	2.7	3.4

5-NN

	L-W	B-W	PEM	L-F	B-F	REN	ANR	CF	CVC	Ens	IPF	Sat
0	-3.1	-5.0	-6.5	0.1	-11.5	-11.5	-56.8	-30.4	0.0	-1.5	-0.5	-1.4
10	0.7	-2.4	-1.8	1.9	-8.9	-8.9	-51.9	-27.8	2.0	0.2	1.2	-3.0
20	8.0	2.5	5.4	7.4	-1.7	-1.7	-38.0	-21.6	6.9	6.2	6.2	0.3
30	15.1	4.3	10.9	13.7	5.1	5.1	-23.9	-11.5	9.8	10.8	11.2	0.4
40	19.9	5.3	11.4	18.2	8.6	8.6	-7.9	-4.6	9.2	16.1	13.4	2.1

MLP

	L-W	B-W	PEM	L-F	B-F	REN	ANR	CF	CVC	Ens	IPF	Sat
0	3.2	-1.0	6.1	0.7	-1.6	-15.7	-68.2	-32.9	3.6	-0.3	2.7	-3.3
10	14.1	2.9	9.5	13.9	2.8	1.2	-46.9	-16.5	10.2	11.1	11.3	2.8
20	22.6	5.2	14.5	20.9	9.5	13.7	-27.1	-4.3	15.5	18.2	17.8	9.0
30	24.3	5.3	13.8	22.7	9.7	18.1	-15.6	0.3	12.5	19.8	17.2	6.3
40	21.2	3.3	13.1	22.0	9.0	16.9	-4.8	1.0	10.7	18.5	15.8	6.1

Random Forest

	L-W	B-W	PEM	L-F	B-F	REN	ANR	CF	CVC	Ens	IPF	Sat
0	-1.9	-1.3	1.8	2.5	-0.1	-15.8	-69.0	-32.3	2.8	-1.5	1.7	-4.8
10	8.0	1.2	3.5	8.8	0.8	-3.3	-50.8	-21.7	7.1	7.0	8.1	0.9
20	19.0	4.0	12.7	19.6	4.9	10.0	-29.0	-9.2	14.6	16.2	17.3	6.0
30	26.7	3.6	16.6	25.5	6.8	19.0	-13.1	1.5	16.2	21.1	21.1	7.3
40	27.1	2.8	15.5	25.6	7.5	19.2	-1.3	2.3	12.9	22.5	18.9	5.5

RIPPER

	L-W	B-W	PEM	L-F	B-F	REN	ANR	CF	CVC	Ens	IPF	Sat
0	-2.3	-1.7	-19.3	2.9	-5.3	-7.9	-47.0	-27.1	2.2	2.4	2.0	0.5
10	0.9	-0.5	-18.9	6.0	-3.0	-2.2	-40.4	-23.2	3.3	3.7	3.5	0.3
20	7.5	3.6	-10.8	10.9	-0.2	4.0	-35.6	-17.3	4.9	8.9	7.1	3.3
30	11.5	2.5	-4.5	13.4	1.5	6.2	-24.5	-10.9	4.0	10.8	8.0	2.9
40	12.8	2.6	-0.7	13.8	2.1	5.2	-15.1	-10.6	6.2	11.9	8.0	-1.2

Table 7.3: A comparison of the effect of the investigated instance weighting methods on the considered learning algorithms averaged over the 54 data sets. Bold accuracy values represent cases where $\mathcal{L}$-weighting achieves significantly higher accuracy than the compared technique.

	C4.5					5-NN				
noise	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
$\mathcal{L}$-W	78.19	77.03	76.33	74.32	71.08	78.72	78.09	76.77	74.8	70.30
Rank	1.8	1.6	1.6	1.4	1.5	1.6	1.5	1.5	1.4	1.5
B-W	79.29	77.14	**75.20**	**71.06**	**65.74**	**78.34**	**77.41**	**75.39**	**71.59**	**64.92**
Rank	1.9	2.0	2.2	2.4	2.5	2.1	2.4	2.5	2.6	2.5
b,e,w	26,4,24	33,2,19	38,0,16	44,1,9	43,1,10	34,7,13	41,3,10	43,1,10	46,1,7	44,1,8
PWEM	**76.41**	**74.50**	**73.34**	**70.94**	**68.28**	**78.02**	**77.54**	**76.12**	**73.56**	**67.18**
Rank	2.3	2.4	2.3	2.2	2.0	2.2	2.2	2.0	2.0	2.0
b,e,w	35,4,15	40,3,11	39,4,11	41,0,13	37,0,16	34,4,16	36,2,16	37,1,16	37,3,14	36,3,14
	MLP					Random Forest				
noise	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
$\mathcal{L}$-W	82.26	80.71	78.64	75.33	69.12	80.82	79.73	78.08	75.87	70.7
Rank	1.7	1.5	1.4	1.3	1.3	1.9	1.5	1.3	1.2	1.2
B-W	**81.49**	**78.19**	**73.84**	**69.14**	**62.1**	80.94	**78.24**	**74.01**	**68.25**	**60.93**
Rank	2.1	2.4	2.6	2.6	2.6	1.9	2.4	2.7	2.8	2.7
b,e,w	34,2,18	42,1,11	47,1,6	48,0,6	47,1,5	25,1,28	42,2,10	48,0,6	48,1,4	48,1,4
PWEM	**82.79**	**79.67**	**76.42**	**71.90**	**65.95**	81.51	**78.73**	**76.37**	**72.54**	**66.03**
Rank	2.2	2.1	2.0	2.0	2.0	2.2	2.1	2.0	2.0	2.1
b,e,w	34,4,16	38,2,14	39,1,14	42,0,12	41,0,12	33,4,17	34,4,16	40,4,10	47,1,5	48,1,4
	RIPPER									
noise	0%	10%	20%	30%	40%					
$\mathcal{L}$-W	77.86	76.62	75.53	73.38	69.53					
Rank	1.8	1.5	1.4	1.4	1.4					
B-W	77.98	76.28	**74.5**	**70.7**	**65.97**					
Rank	1.8	1.8	2.0	2.3	2.3					
b,e,w	27,3,24	31,2,21	36,3,15	45,2,6	44,0,9					
PWEM	**74.17**	**71.94**	**70.68**	**68.57**	**64.82**					
Rank	2.4	2.6	2.5	2.4	2.3					
b,e,w	36,4,14	45,2,7	45,4,5	39,2,12	40,1,12					

Table 7.4: A comparison of $\mathcal{L}$-filtering with the other filtering techniques for the 5 considered learning algorithms averaged over the 54 data sets. Bold accuracy values represent cases where $\mathcal{L}$-filtering achieves significantly higher accuracy than the compared technique.

	C4.5					5-NN				
noise	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
$\mathcal{L}$-F	79.55	78.35	76.65	73.42	69.14	79.4	78.35	76.62	74.38	69.67
Rank	2.7	2.0	1.7	1.9	1.8	2.5	2.6	2.4	2.3	2.3
B-F	79.34	**77.49**	**75.17**	**71.32**	**65.58**	**76.99**	**75.98**	**74.33**	**71.82**	**66.12**
Rank	3.2	3.5	3.6	3.4	3.3	4.2	4.3	3.9	3.6	3.6
b,e,w	31,8,15	42,1,11	44,4,6	44,1,9	46,0,8	41,4,9	45,0,9	43,0,11	37,1,16	39,1,14
CVC	79.41	**77.35**	**74.60**	**70.11**	**63.64**	79.37	78.38	76.5	**73.21**	**66.36**
Rank	3.1	3.4	3.8	4.0	4.1	2.6	2.7	3.0	3.4	3.5
b,e,w	25,8,21	43,1,10	50,1,3	47,3,4	48,2,4	26,7,21	26,5,23	34,2,18	37,2,15	39,2,13
Ens	79.53	**77.88**	76.15	73.62	68.66	79.06	77.99	76.33	73.51	68.92
Rank	2.8	2.8	2.4	2.0	2.0	3.0	2.8	2.9	2.9	2.7
b,e,w	27,2,25	34,3,17	35,3,16	26,1,27	30,1,23	32,7,15	29,3,22	29,4,21	34,1,19	32,0,22
IterP	79.31	**77.33**	**74.82**	**70.41**	**64.31**	79.27	78.2	76.33	73.63	67.90
Rank	3.1	3.5	3.6	3.6	3.8	2.7	2.6	2.8	2.8	2.9
b,e,w	25,10,19	42,0,12	44,1,9	45,0,9	49,0,5	24,4,26	25,4,25	31,3,20	35,1,18	38,1,15

	MLP					RandForest				
noise	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
$\mathcal{L}$-F	81.80	80.66	78.17	74.82	69.44	81.66	79.91	78.23	75.45	70.11
Rank	2.9	2.2	2.0	2.0	1.8	2.8	2.6	2.1	1.8	1.8
B-F	81.39	**78.16**	**75.04**	**70.57**	**64.35**	81.16	**78.13**	**74.25**	**69.32**	**62.80**
Rank	3.3	4.1	3.9	3.7	3.8	3.1	4.0	4.3	4.2	4.3
b,e,w	25,4,25	46,0,8	44,1,9	44,0,10	46,1,7	28,1,25	42,1,11	47,1,6	50,0,4	50,0,4
CVC	82.33	79.83	**76.68**	**71.5**	**65.02**	81.72	79.53	**76.88**	**72.40**	**64.97**
Rank	2.6	2.9	3.4	3.8	3.7	2.7	3.0	3.2	3.6	3.6
b,e,w	22,3,29	33,2,19	43,2,9	46,0,8	48,1,5	23,4,27	31,2,21	40,1,13	47,1,6	47,1,6
Ens	81.62	80.02	**77.44**	73.85	**68.04**	80.91	79.52	**77.32**	**74.01**	68.86
Rank	3.3	2.9	2.9	2.5	2.6	3.5	2.9	3.0	2.8	2.4
b,e,w	32,3,19	36,2,16	32,2,20	30,1,23	34,0,20	38,1,15	31,1,22	39,2,13	38,1,15	34,0,20
IterP	82.18	80.08	**77.31**	**73.02**	**67.02**	81.51	79.75	77.61	74.02	**67.40**
Rank	2.8	2.9	2.9	3.0	3.1	2.9	2.5	2.6	2.5	2.9
b,e,w	25,4,25	35,0,19	38,2,14	40,1,13	44,1,9	23,6,25	25,1,28	35,0,19	36,3,15	44,0,10

	RIPPER				
noise	0%	10%	20%	30%	40%
$\mathcal{L}$-F	78.98	77.82	76.43	73.97	69.90
Rank	2.7	2.4	2.0	1.9	2.2
B-F	**77.20**	**75.70**	**73.48**	**70.39**	**65.78**
Rank	3.8	3.8	4.0	4.0	4.0
b,e,w	37,3,14	44,1,9	46,0,8	44,1,9	44,2,8
CVC	78.83	77.19	**74.85**	**71.15**	**67.24**
Rank	2.8	3.0	3.3	3.8	3.4
b,e,w	26,4,24	32,1,21	40,2,12	50,1,3	41,1,12
Ens	78.87	77.29	75.90	73.19	69.22
Rank	2.9	2.9	2.6	2.3	2.1
b,e,w	28,3,23	32,4,18	36,1,17	32,1,21	24,1,29
IterP	78.78	77.23	**75.41**	**72.36**	**67.87**
Rank	2.8	2.9	3.1	3.0	3.3
b,e,w	27,3,24	30,2,22	38,3,13	42,1,11	43,1,10

and 4) the iterative partitioning filter (IterP). The CVC, Ens and IterP filters were chosen as representative filtering algorithms as they achieved the highest increase in classification accuracy. For filtering, no filtering approach significantly outperforms the $\mathcal{L}$-filter, yet the $\mathcal{L}$-filter achieves significantly higher classification accuracy in most cases (bold accuracy values in Table 7.4). The $\mathcal{L}$-filter is an expansion of the ensemble filter. The additional diverse learning algorithms can significantly improve the accuracy, however, in most cases, the difference in performance between the $\mathcal{L}$-filter and the ensemble filter is not significant.

7.5.3 Comparison with an Ensemble

Table 7.5 compares the $\mathcal{L}$-ensemble with $\mathcal{L}$-weighting, $\mathcal{L}$-filtering, and the ensemble filter for the five learning algorithms. These noise handling methods were chosen to be representative as they achieved the highest and most significant gains in classification accuracy in a single learning algorithm. Despite the significant increase in classification accuracy over the original learning algorithm by the noise handling method, the $\mathcal{L}$-ensemble achieves significantly higher classification accuracy over all of the considered learning algorithms at all of the noise levels. This demonstrates the $\mathcal{L}$-ensemble's robustness to label noise. This also contradicts the finding by Brodley and Friedl that while an ensemble outperforms a single learning algorithm, it cannot replace filtering [2].

The increase in accuracy by the voting ensemble is not too surprising as ensembles have been shown to perform better than any one of their constituent base classifiers [5]. We focus our attention on the MLP and random forest learning algorithms since they obtain the highest classification accuracy of the considered learning algorithms. Although the increase in accuracy by a voting ensemble is significant, it is generally within 1 to 2 percent of the average accuracy achieved by a MLP or a random forest with $\mathcal{L}$-weighting or $\mathcal{L}$-filtering. Noise handling does increase the classification accuracy, but the $\mathcal{L}$-ensemble is a safe choice to use in general.

Table 7.5: A comparison of the $\mathcal{L}$-ensemble against the investigated noise handling approaches for the 5 considered learning algorithm averaged over the 54 data sets.

noise	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
$\mathcal{L}$-Ens	83.36	82.06	79.87	77.09	72.04					
	C4.5					5-NN				
Ens Rank	1.4	1.6	1.7	1.7	2.1	1.4	1.6	1.7	1.8	2.0
$\mathcal{L}$-Weight	**78.36**	**77.39**	**76.44**	**74.32**	70.84	**78.72**	**78.09**	**76.77**	**74.80**	**70.30**
Rank	3.1	2.9	2.5	2.4	2.2	3.0	2.7	2.6	2.4	2.4
b,e,w	46,1,7	44,0,10	39,0,15	37,0,17	29,0,24	44,4,6	44,0,10	38,2,14	37,1,16	33,3,17
$\mathcal{L}$-Filter	**79.55**	**78.35**	**76.65**	**73.42**	**69.14**	**79.40**	**78.35**	**76.62**	**74.38**	**69.67**
Rank	2.7	2.6	2.7	3.0	2.8	2.6	2.8	2.7	2.7	2.7
b,e,w	45,3,6	44,0,10	42,0,12	46,1,7	37,0,17	47,2,5	42,2,10	42,2,10	42,1,11	36,0,18
Ens Filt	**79.53**	**77.88**	**76.15**	**73.62**	**68.66**	**79.06**	**77.99**	**76.33**	**73.51**	**68.92**
Rank	2.8	3.0	3.1	2.9	3.0	3.0	2.9	3.0	3.1	2.9
b,e,w	46,1,7	45,0,9	42,0,12	43,0,11	40,0,14	44,1,9	45,0,9	42,0,12	41,1,12	37,0,17
	MLP					Random Forest				
Ens Rank	1.7	1.7	1.7	1.7	1.8	1.5	1.5	1.5	1.7	1.8
$\mathcal{L}$-Weight	82.26	**80.71**	**78.64**	**75.33**	**69.12**	**80.82**	**79.73**	**78.08**	75.87	70.7
Rank	2.2	2.5	2.4	2.4	2.4	2.9	2.5	2.6	2.3	2.1
b,e,w	37,3,14	42,0,12	36,2,16	36,1,17	36,0,17	44,0,10	41,3,10	41,2,11	35,1,17	30,0,23
$\mathcal{L}$-Filter	**81.80**	**80.66**	**78.17**	**74.82**	**69.44**	**81.66**	**79.91**	**78.23**	**75.45**	**70.11**
Rank	2.9	2.6	2.8	2.8	2.6	2.5	2.9	2.6	2.7	2.8
b,e,w	40,5,9	39,1,14	41,0,13	40,2,12	36,0,18	41,4,9	49,0,5	45,0,9	42,1,11	40,0,14
Ens Filt	**81.62**	**80.02**	**77.44**	**73.85**	**68.04**	**80.91**	**79.52**	**77.32**	**74.01**	**68.86**
Rank	3.2	3.2	3.2	3.1	3.2	3.1	3.1	3.3	3.4	3.3
b,e,w	42,1,11	44,2,8	46,0,8	46,2,6	46,1,7	45,1,8	47,0,7	47,1,6	44,4,6	45,2,7
	RIPPER									
Ens Rank	1.3	1.3	1.6	1.7	2.0					
$\mathcal{L}$-Weight	**77.86**	**76.62**	**75.53**	**73.38**	**69.53**					
Rank	3.2	3.1	3.0	2.6	2.6					
b,e,w	47,2,5	48,0,6	41,1,12	39,0,14	32,1,20					
$\mathcal{L}$-Filter	**78.98**	**77.82**	**76.43**	**73.97**	**69.90**					
Rank	2.6	2.7	2.5	2.7	2.7					
b,e,w	47,2,5	49,0,5	44,0,10	42,1,11	37,1,16					
Ens Filt	**78.87**	**77.29**	**75.90**	**73.19**	**69.22**					
Rank	2.8	3.0	3.0	3.0	2.6					
b,e,w	50,1,3	48,0,6	45,1,8	40,1,13	35,1,18					

The finding that the $\mathcal{L}$-filter is robust to noise is somewhat surprising, especially in light of the findings by Brodley and Friedl [2] that noise filtering is generally preferable to a voting ensemble. We differ from Brodley and Friedl in the scope of our investigation (we studied noise over a much larger set of data sets), we used classifier diversity to select the set of learning algorithms, and we used a larger set of learning algorithms to identify noisy instances (3 vs. 9). Classifier diversity may have a larger impact on the voting ensemble rather than on filtering or weighting. It may also be due to the larger number of learning algorithms used. Thus, there is a trade-off between the computational resource required for a larger ensemble and the higher accuracy of the induced model. We chose nine learning algorithms for the ensemble as a reasonable trade-off between performance and computational complexity. Further, more ensembled learning algorithms could lead to more improvements in performance.

7.6 Conclusions

In this paper we examined handling class noise using the hypotheses from a diverse set of learning algorithms through a large set of experiments. We introduced *noise identification using classifier diversity* (NICD) which uses a diverse set of learning algorithms to approximate $p(y|x)$. On a set of 5 learning algorithms and 54 data sets, we examined NICD as a filtering technique ($\mathcal{L}$-filtering), a weighting technique ($\mathcal{L}$-weighting), and as the base classifiers in a voting ensemble ($\mathcal{L}$-ensemble) . We also examined several other noise handling techniques. We found that the ensembled noise handling approaches significantly outperform other less diverse noise handling techniques. Our results suggest that a less biased estimate of $p(y|x)$ leads to better noise handling. Compared to no noise handling, $\mathcal{L}$-weighting and $\mathcal{L}$-filtering often achieve a significantly higher classification accuracy and never achieve a significantly lower classification accuracy. Thus, ensembled approaches for handling noise are able to be applied across a broad set of data sets and learning algorithms.

Despite the increase in accuracy exhibited by many of the noise handling techniques, the $\mathcal{L}$-ensemble achieves a significantly higher classification accuracy for all of the learning algorithms and noise handling techniques for all of the examined noise levels – in contrast to previous work. The $\mathcal{L}$-ensemble is a voting ensemble composed of a diverse set of learning algorithms (Section 7.2) with each learning algorithm having an equally-weighted vote. Thus, a voting ensemble of diverse learning algorithms exhibits a robustness to noise that the individual constituent learning algorithms do not possess. The diversity of the base classifiers produces a robustness against label noise.

References

[1] J. Alcalá-Fdez, L. Sánchez, S. García, M. J. Del Jesus, S. Ventura, J. M. Garrell, J. Otero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. Keel: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13 (3):307–318, October 2008. ISSN 1432-7643.

[2] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[3] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

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

[5] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000. ISBN 3-540-67704-6.

[6] Chris H. Q. Ding and Inna Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17(4):349–358, 2001.

[7] A. Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

[8] Benoit Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[9] D. Gamberger, N. Lavrac, and C. Groselj. Experiments with noise filtering in a medical domain. In *Proceedings of the 16th International Conference on Machine Learning*, pages 143–151, 1999.

[10] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. *Applied Artificial Intelligence*, 14(2):205–223, 2000.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[12] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 12(10):993–1001, October 1990.

[13] Robert E. Kass and Larry Wassermann. A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion. *Journal of the American Statistical Association*, 90(431):928–934, 1995.

[14] T.M. Khoshgoftaar and P. Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22:387–396, 2007.

[15] Neil D. Lawrence and Bernhard Schölkopf. Estimating a kernel fisher discriminant in the presence of label noise. In *In Proceedings of the 18th International Conference on Machine Learning*, pages 306–313, 2001.

[16] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[17] David F. Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.

[18] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, pages 841–848, 2001.

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

[20] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[21] Umaa Rebbapragada and Carla E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.

[22] José A. Sáez, Julián Luengo, and Francisco Herrera. Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. *Pattern Recognition*, 46(1):355–364, 2013.

[23] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[24] Michael R. Smith and Tony Martinez. An extensive evaluation of filtering misclassified instances in supervised classification tasks. *In submission*, 2014. URL `http://arxiv.org/abs/1312.3970`.

[25] G. Stiglic and P. Kokol. Stability of ranked gene lists in large microarray analysis studies. *Journal of biotechnology*, page 9 pages, 2010. , 616385.

[26] Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:448–452, 1976.

[27] Sofie Verbaeten and Anneleen Van Assche. Ensemble methods for noise elimination in classification problems. In *Proceedings of the 4th international conference on multiple classifier systems*, pages 317–325, 2003. ISBN 3-540-40369-8.

[28] Xinchuan Zeng and Tony R. Martinez. A noise filtering method using neural networks. In *Proc. of the int. Workshop of Soft Comput. Techniques in Instrumentation, Measurement and Related Applications*, 2003.

[29] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artificial Intelligence Review*, 22:177–210, November 2004.

Part III

Conclusion

Part III concludes this dissertation and is composed of three chapters. Chapter 8 compares the current and potential accuracies obtainable by filtering instances and hyperparameter optimization. As filtering achieves a much higher potential, Chapter 8 motivates the continued study of learning the interaction of the data. The chapter is currently under review and can be referenced as follows.

Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. "The Potential Benefits of Data Set Filtering and Learning Algorithm Hyperparameter Optimization", in submission 2015.

Chapter 9 presents the framework for a repository of results from machine learning experiments. The repository is designed to provide information at the instance-level to facilitate instance-level investigations and to provide various data set-level measurements that can be computed from the instance-level measurements. The end goal is to facilitate meta-learning at the instance-level as well as the data set-level. The reference for the corresponding paper is given below.

Michael R. Smith, Andrew White, Christophe Giraud-Carrier, and Tony Martinez. "An Easy to Use Repository for Comparing and Improving Machine Learning Algorithm Usage", *The ECAI Workshop on Meta-learning & Algorithm Selection (MetaSel)*, pages 41–48, 2014.

Chapter 10 summarizes this work including its contributions and directions for future work.

Chapter 8

The Potential Benefits of Data Set Filtering and Learning Algorithm Hyperparameter Optimization

Submitted to: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2015)*, 2015.

Abstract

The quality of a model induced by a learning algorithm is dependent upon the training data *and* the hyperparameters supplied to the learning algorithm. Prior work has shown that a model's quality can be significantly improved by filtering out low quality instances or by tuning the learning algorithm hyperparameters. However, the potential impact of filtering and hyperparameter optimization (HPO) is largely unknown. In this paper, we estimate and compare the *potential* benefits of instance filtering and HPO. While both HPO and filtering significantly improve the quality of the induced model, we find that filtering has a greater potential effect on the quality of the induced model than HPO, motivating future work in filtering.

8.1 Introduction

Given a set of training instances composed of input feature vectors and corresponding labels, the goal of supervised machine learning is to induce an accurate generalizing function (hypothesis) that maps feature vectors to labels. The quality of the induced function is dependent on the learning algorithm's hyperparameters *and* the quality of the training data. It is known that no learning algorithm or hyperparameter setting is best for all data sets (no

free lunch theorem [26]) and that the performance of many learning algorithms is sensitive to their hyperparameter settings. It is also well-known that real-world data sets are typically noisy.

Prior work has shown that the generalization performance of an induced model can be significantly improved through hyperparameter optimization (HPO) [1], or by increasing the quality of the training data using techniques such as noise correction [11], instance weighting [17], or instance filtering [20]. Searching the hyperparameter space and improving the quality of the training data have generally been examined in isolation and the potential impact of their usage has not been examined. In this paper, we compare the effects of HPO with the effects of improving the quality of the training data through filtering. The results of our experiments provide insight into the potential effectiveness of both HPO and filtering.

We evaluate 6 commonly used learning algorithms and 46 data sets. We examine the effects of HPO and filtering by: 1) using a standard approach that sets the hyperparameters of an algorithm by maximizing the accuracy on a validation set and 2) using an optimistic approach that sets the hyperparameters for an algorithm using the 10-fold cross-validation accuracy. The standard and optimistic approaches are explained in more detail in Section 8.4. Essentially, the optimistic approach indicates how well a technique *could* perform if the training set were representative of the test set and provides insight into the *potential* benefit of a given technique. The standard approach provides a representative view of HPO and filtering in their present state and allows an evaluation of how well current HPO and filtering techniques fulfill their potential.

Using the standard approach, we find that in most cases both HPO and filtering significantly increase classification accuracy over using a learning algorithm with its default parameters trained on unfiltered data. For the optimistic estimates of HPO and filtering, we find that *filtering significantly improves the classification accuracy over HPO* for all of the investigated learning algorithms–increasing the accuracy more than HPO for almost all of the considered data sets. HPO achieves an average accuracy of 84.8% while filtering achieves

an average accuracy of 89.1%. The standard approach for HPO and filtering achieves an average accuracy of 82.6% and 82.0% respectively. These results provide motivation for further research into developing algorithms that improve the quality of the training data.

8.2 Related Work

Smith et al. [21] found that a significant number of instances are difficult to classify correctly, that the hardness of each instance is dependent on its relationship with the other instances in the training set and that some instances can be detrimental. Thus, there is a need for improving the how detrimental instances are handled during training as they affect the classification of other instances. Improving the quality of the training data has typically fallen into three approaches: filtering, cleaning, and instance weighting [7].

Each technique within an approach differs in how detrimental instances are identified. A common technique for filtering removes instances from a data set that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. Removing the training instances that are suspected to be noise and/or outliers prior to training has the advantage that they do not influence the induced model and generally increase classification accuracy. A negative side-effect of filtering is that beneficial instances can also be discarded and produce a worse model than if all of the training data had been used [18]. Rather than discarding the instances from a training set, noisy or possibly corrupted instances can be cleaned or corrected [11]. However, this could artificially corrupt valid instances. Alternatively, weighting weights suspected detrimental instances rather than discards them and allows for an instance to be considered on a continuum of detrimentality rather than making a binary decision [17].

Other methods exist for improving the quality of the training data, such as feature selection/extraction [8]. While feature selection and extraction can improve the quality of the training data, we focus on improving quality via filtering – facilitating a comparison between filtering and HPO on the same feature set.

Much of the previous work in improving the quality of the training data artificially corrupts training instances to determine how well an approach would work in the presence of noisy or mislabeled instances. In some cases, a given approach only has a significant impact when there are large degrees of artificial noise. In contrast, we do not artificially corrupt a data set to create detrimental instances. Rather, we seek to identify the detrimental instances that are already contained in a data set and show that correctly labeled, non-noisy instances can *also* be detrimental for inducing a model of the data. Properly handling detrimental instances can result in significant gains in classification accuracy.

The grid search and manual search are the most common types of HPO techniques in machine learning and a combination of the two approaches are commonly used [12]. [1] proposed to use a random search of the hyperparameter space. The premise of random HPO is that most machine learning algorithms have very few hyperparameters that considerably affect the final model while the other hyperparameters have little to no effect. Random search provides a greater variety of the hyperparameters that considerably affect the model. Given the same amount of time constraints, random HPO has been shown to outperform a grid search. Bayesian optimization has also been used to search the hyperparameter space [23]. Bayesian optimization techniques model the dependence of an error function $\mathcal{E}$ on the hyperparameters λ as $p(\mathcal{E},\lambda)$ using, for example, a tree-structured Parzen estimator [2] or Gaussian processes [10].

8.3 Preliminaries

In this section, we examine the effects of HPO and the quality of the training data mathematically. Let T represent a training set composed of a set of input vectors $X = \{x_1,x_2,...,x_n\}$ and corresponding label vectors $Y = \{y_1,y_2,...,y_n\}$, i.e., $T = \{\langle x_i,y_i\rangle : x_i \in X \wedge y_i \in Y\}$. Given that in most cases, all that is known about a task is contained in the set of training instances T, at least initially, the training instances are generally considered equally. Most machine learning algorithms seek to induce a hypothesis $h : X \rightarrow Y$ that minimizes a spec-

ified loss function $\mathcal{L}(\cdot)$. As most real-world data sets contain some level of noise, there is generally a model-dependent regularization term $\mathcal{R}(\cdot)$ added to $\mathcal{L}(\cdot)$ that penalizes more complex models and aids in overfit avoidance. The noise in T may arise from errors in the data collection process such as typos or errors in data collection equipment. In addition to noise from errors, there may be non-noisy outlier instances due to the stochastic nature of the task. A hypothesis h is induced by a learning algorithm g trained on T with hyperparameters λ ($h = g(T,\lambda)$), such that:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{|T|} \sum_{\langle x_i, y_i \rangle \in T} \mathcal{L}(h(x_i), y_i) + \alpha \mathcal{R}(h) \tag{8.1}$$

where α is a regularization parameter greater than or equal to 0 that determines how much weight to apply to the regularization term and $h(\cdot)$ returns the predicted class for a given input. The quality of the induced hypothesis h is characterized by its empirical error for a specified error function $\mathcal{E}$ on a test set V:

$$E(h,V) = \frac{1}{|V|} \sum_{\langle x_i, y_i \rangle \in V} \mathcal{E}(h(x_i), y_i)$$

where V can be T or a disjoint set of instances. In k-fold cross-validation, the empirical error is the average empirical error from the k folds (i.e., $1/k \; E(h_i,V_i)$).

Characterizing the success of a learning algorithm at the data set level (e.g., accuracy or precision) optimizes over the entire training set and marginalizes the impact of a single training instance on an induced model. Some sets of instances can be more beneficial than others for inducing a model of the data and some can even be detrimental. By *detrimental instances*, we mean instances that have a negative impact on the induced model. For example, outliers or mislabeled instances are not as beneficial as border instances and are detrimental in many cases. In addition, other instances can be detrimental for inducing a model of the data even if they are labeled correctly. Formally, a set $\mathcal{D}$ of detrimental instances is a

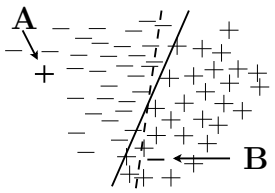

Figure 8.1: Hypothetical 2-dimensional data set that shows the potential effects of detrimental instances in the training data on a learning algorithm.

subset of the training data that, when used in training, increases the empirical error, i.e., $E(g(T,\lambda),V) > E(g(T-\mathcal{D},\lambda),V)$.

The effect of training with detrimental instances is demonstrated in the hypothetical two-dimensional data set shown in Figure 8.1. Instances A and B represent detrimental instances. The solid line represents the "actual" classification boundary and the dashed line represents a potential induced classification boundary. Instances A and B adversely affect the induced classification boundary because they "pull" the classification boundary and cause several other instances to be misclassified that otherwise would have been classified correctly.

Despite most learning algorithms having a mechanism to avoid overfitting, the presence of detrimental instances may still affect the induced model for many learning algorithms. Mathematically, the effect of each instance on the induced hypothesis is shown in Equation 8.1. The loss from each instance in T, including detrimental instances, is equally weighted. Detrimental instances have the most significant impact during the early stages of training where it is difficult to identify them [6]. The presence of $\mathcal{D}$ may also affect the value of $\mathcal{R}(h)$. For example, removing $\mathcal{D}$ from T could produce a "simpler" h that reduces $\mathcal{R}(h)$.

8.3.1 Hyperparameter Optimization

The quality of an induced model by a learning algorithm depends in part on the learning algorithm's hyperparameters. With hyperparameter optimization (HPO), the hyperparameter

space Λ is searched to minimize the empirical error on V:

$$\underset{\lambda \in \Lambda}{\operatorname{argmin}} E(g(T,\lambda),V). \tag{8.2}$$

The hyperparameters can have a significant effect on the quality of the induced model as well as suppressing the effects of detrimental instances. For example, in a support vector machine, [4] use the ramp-loss function which limits the penalty on instances that are too far from the decision boundary rather than the more typical 0-1 loss function to handle detrimental instances. Suppressing the effects of detrimental instances with HPO improves the induced model, but does not change the fact that detrimental instances *still* affect the model. Each instance is still considered during the learning process though its influence may be lessened. We describe the method we use for HPO in Section 8.4.1.

8.3.2 Filtering

The quality of an induced model also depends on the quality of the training data where, for example, the quality of the training data can be measured by the amount of detrimental instances present. Low quality training data results in lower quality induced models. Improving the quality of the training data involves searching the training set space to find an optimal subset that minimizes the empirical error:

$$\underset{t \in \mathcal{P}(T)}{\operatorname{argmin}} E(g(t,\lambda),V)$$

where t is a subset of T and $\mathcal{P}(T)$ is the power set of T. The removed instances obviously have no effect on the induced model. In Section 8.4.2, we describe how we identify detrimental instances and search for an optimal subset of the training data that minimizes empirical error.

8.4 Implementation Details

8.4.1 Bayesian Hyperparameter Optimization

In this paper, we use Bayesian optimization for HPO. Specifically, we use *sequential model-based optimization* (SMBO) [10] as it has been shown to yield better performance than grid and random search [23, 24]. SMBO is a stochastic optimization framework that builds a probabilistic model $\mathcal{M}$ that captures the dependence of $\mathcal{E}$ on λ. SMBO first initializes $\mathcal{M}$. After initializing $\mathcal{M}$, SMBO searches the search space by 1) querying $\mathcal{M}$ for a promising λ to evaluate, 2) evaluating the loss $\mathcal{E}$ of using configuration λ, and then 3) updating $\mathcal{M}$ with λ and $\mathcal{E}$. Once the budgeted time is exhausted, the hyperparameter configuration with the minimal loss is returned.

To select a candidate hyperparameter configuration, SMBO relies on an acquisition function $a_{\mathcal{M}} : \Lambda \rightarrow \mathbb{R}$ which uses the predictive distribution of $\mathcal{M}$ to quantify how useful knowledge about λ would be. SMBO maximizes $a_{\mathcal{M}}$ over Λ to select the most useful hyperparameter configuration λ to evaluate next. One of the most prominent acquisition functions is the *positive expected improvement* (EI) over an existing error rate $\mathcal{E}_{min}$ [19]. If $\mathcal{E}(\lambda)$ represents the error rate of hyperparameter configuration λ, then the EI function over $\mathcal{E}_{min}$ is:

$$EI_{\mathcal{E}_{min}}(\lambda) = max\{\mathcal{E}_{min} - \mathcal{E}(\lambda),0\}.$$

As $\mathcal{E}(\lambda)$ is unknown, the expectation of $\mathcal{E}(\lambda)$ with respect to the current model $\mathcal{M}$ can be computed as:

$$\mathbb{E}_{\mathcal{M}}[EI_{\mathcal{E}_{min}}(\lambda)] = \int_{-\infty}^{\mathcal{E}_{min}} max\{\mathcal{E}_{min} - \mathcal{E},0\} \cdot p(\mathcal{E}|\lambda)d\mathcal{E}.$$

SMBO is dependent on the model class used for $\mathcal{M}$. Following [24], we use sequential model-based algorithm configuration (SMAC) [10] for $\mathcal{M}$ with EI as $a_{\mathcal{M}}$, although others could be used such as the tree-structured Parzen estimator. To model $p(\mathcal{E}|\lambda)$, we use random forests as they tend to perform well with discrete and continuous input data. Using random

forests, SMAC obtains a predictive mean μ_λ and variance σ_λ^2 of $p(\mathcal{E}|\lambda)$ calculated using the predictions from the individual trees in the forest for λ. $p(\mathcal{E}|\lambda)$ is then modeled as a Gaussian distribution $\mathcal{N}(\mu_\lambda,\sigma_\lambda^2)$. To create diversity in the evaluated configurations, every second configuration is selected at random as suggested [24]. For k-fold cross-validation, the **standard approach** finds the hyperparameters that minimize the error for each of the k validation sets as shown in Equation 8.2. The **optimistic approach** finds the hyperparameters that minimize the k-fold cross-validation error:

$$\underset{\lambda\in\Lambda}{\operatorname{argmin}}\frac{1}{k}E(g(T_i,\lambda),V_i)$$

where T_i and V_i are the training and validation sets for the ith fold. The hyperparameter space Λ is searched using Bayesian hyperparameter optimization for both approaches.

8.4.2 Filtering

Identifying detrimental instances is a non-trivial task. Fully searching the space of subsets of training instances generates 2^N subsets of training instances where N is the number of training instances. Even for small data sets, it is computationally infeasible to induce 2^N models to determine which instances are detrimental. There is no known way to determine how a set of instances will affect the induced classification function from a learning algorithm without inducing a classification function with the investigated set of instances removed from the training set.

The Standard Filtering Approach

Previous work in noise handling has shown that class noise (e.g. mislabeled instances) is more detrimental than attribute noise [15]. Thus, searching for detrimental instances that are likely to be misclassified is a natural place to start. In other words, we search for instances where the probability of the class label is low given the feature values (i.e., low $p(y_i|x_i)$). In general,

$p(y_i|x_i)$ does not make sense outside the context of an induced hypothesis. Thus, using an induced hypothesis h from a learning algorithm trained on T, the quantity $p(y_i|x_i)$ can be approximated as $p(y_i|x_i,h)$. After training a learning algorithm on T, the class distribution for an instance x_i can be estimated based on the output from the learning algorithm. Prior work has examined removing instances that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. We filter instances using an ensemble filter that removes instances that are misclassified by more than $x\%$ of the algorithms in the ensemble.

The dependence of $p(y_i|x_i,h)$ on a particular h can be lessened by summing over the space of all possible hypotheses:

$$p(y_i|x_i) = \sum_{h \in \mathcal{H}} p(y_i|x_i,h)p(h|T). \tag{8.3}$$

However, this formulation is infeasible to compute in most practical applications as $p(h|T)$ is generally unknown and $\mathcal{H}$ is large and possibly infinite. To sum over $\mathcal{H}$, one would have to sum over the complete set of hypotheses, or, since $h = g(T,\lambda)$, over the complete set of learning algorithms and hyperparameters associated with each algorithm.

The quantity $p(y_i|x_i)$ can be estimated by restricting attention to a diverse set of representative algorithms (and hyperparameters). The diversity of the learning algorithms refers to the likelihood that the learning algorithms classify instances differently. A natural way to approximate the unknown distribution $p(h|T)$ is to weight a set of representative learning algorithms, and their associated hyperparameters, $\mathcal{G}$, a priori with an equal, non-zero probability while treating all other learning algorithms as having zero probability. We select a diverse set of learning algorithms using unsupervised metalearning (UML) [13] to get a good representation of $\mathcal{H}$, and hence a reasonable estimate of $p(y_i|x_i)$. UML uses Classifier Output Difference (COD) [16] measures the diversity between learning algorithms as the probability that the learning algorithms make different predictions. UML clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. Here,

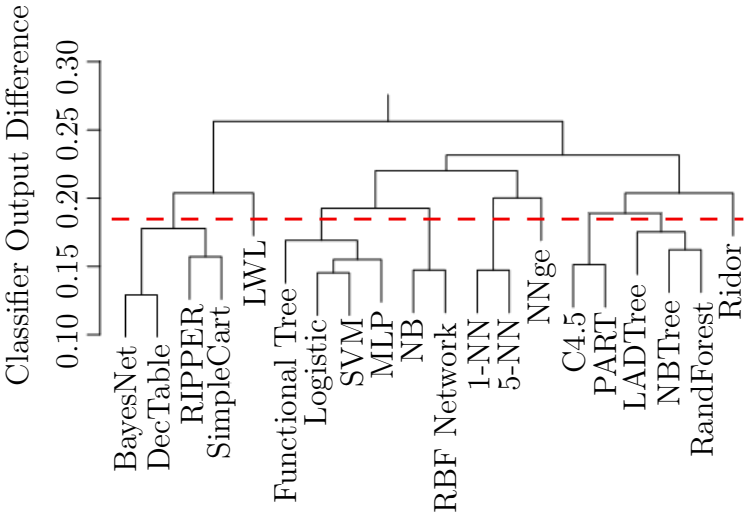

Figure 8.1: Dendrogram of the considered learning algorithms clustered using unsupervised metalearning based on their classifier output difference. The dashed line represents the cut-off value of 0.18 that was used to create 9 clusters of learning algorithms. One learning algorithm was selected from each cluster to create $\mathcal{G}$.

Table 8.1: Set of learning algorithms $\mathcal{G}$ used to estimate $p(y_i|x_i)$.

Learning Algorithms
* Multilayer Perceptron trained with Back Propagation (MLP)
* Decision Tree (C4.5)
* Locally Weighted Learning (LWL)
* 5-Nearest Neighbors (5-NN)
* Nearest Neighbor with generalization (NNge)
* Naïve Bayes (NB)
* RIpple DOwn Rule learner (RIDOR)
* Random Forest (RandForest)
* Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

we consider 20 commonly used learning algorithms with their default hyperparameters as set in Weka [9]. The resulting dendrogram is shown in Figure 8.1, where the height of the line connecting two clusters corresponds to the distance (COD value) between them. A cut-point of 0.18 was arbitrarily chosen to create nine clusters and a representative algorithm from each cluster was used to create $\mathcal{G}$ as shown in Table 8.1.

Given a set $\mathcal{G}$ of learning algorithms, we can approximate Equation 8.3 to the following:

$$p(y_i|x_i) \approx \frac{1}{|\mathcal{G}|}\sum_{j=1}^{|\mathcal{G}|} p(y_i|x_i,g_j(T,\lambda)) \tag{8.4}$$

where $p(h|T)$ is approximated as $\frac{1}{|\mathcal{G}|}$ and g_j is the jth learning algorithm from $\mathcal{G}$. As not all learning algorithms produce probabilistic outputs, the distribution $p(y_i|x_i,g_j(T,\lambda))$ is estimated using the Kronecker delta function in this paper.

The Optimistic Filtering Approach

To measure the *potential* impact of filtering, we need to know how removing an instance or set of instances affects the generalization capabilities of the model. We measure this by dynamically creating an ensemble filter from $\mathcal{G}$ using a greedy algorithm for a given data set and learning algorithm. This allows us to find a specific ensemble filter that is best for filtering a given data set and learning algorithm combination. The adaptive ensemble filter is constructed by iteratively adding the learning algorithm g from $\mathcal{G}$ that produces the highest cross-validation classification accuracy when g is added to the ensemble filter. Because we are using the probability that an instance will be misclassified rather than a binary yes/no decision (Equation 8.4), we also use a threshold ϕ to determine which instances are detrimental. Instances with a $p(y_i|x_i)$ less than ϕ are discarded from the training set. A constant threshold value for ϕ is set to filter the instances for all iterations. The baseline accuracy for the adaptive approach is the accuracy of the learning algorithm without filtering. The search stops once adding one of the remaining learning algorithms to the ensemble filter does not increase accuracy, or all of the learning algorithms in $\mathcal{G}$ have been used.

Even though all of the detrimental instances are included for evaluation, the adaptive filter ($\mathcal{A}$-Filter) overfits the data since the cross-validation accuracy is used to determine which set of learning algorithms to use in the ensemble filter. This allows us to find the detrimental instances to examine the effects that they can have on an induced model. This

is not feasible in practical settings, but provides insight into the potential improvement gained from filtering.

8.5 Filtering and HPO

In this section, we compare the effects of filtering with those of HPO using the optimistic and standard approaches presented in Section 8.4. The optimistic approach provides an approximation of the potential of HPO and filtering. In addition to reporting the average classification accuracy, we also report the average rank of each approach. The average accuracy and rank for each algorithm is determined using 5 by 10-fold cross-validation. Statistical significance between pairs of algorithms is determined using the Wilcoxon signed-ranks test (as suggested by [5]) with an alpha value of 0.05.

8.5.1 Experimental Methodology

For HPO, we use the version of SMAC implemented in auto-WEKA [24] as described in Section 8.4.1 Auto-WEKA searches the hyperparameter spaces for the learning algorithms in the Weka machine learning toolkit [9] for a specified amount of time. To estimate the amount of time required for a learning algorithm to induce a model of the data, we ran our selected learning algorithms with ten random hyperparameter settings and calculated the average and max running times. On average, a model was induced in less than 3 minutes. The longest time required to induce a model was 845 minutes. Based on this analysis, we run auto-WEKA for one hour for most of the data sets. An hour long search explores more than 512 hyperparameter configurations for most of the learning algorithm/data set combinations. The time limit is adjusted accordingly for the larger data sets. Following [24], we run four runs with different random seeds provided to SMAC.

For filtering using the ensemble filter ($\mathcal{G}$-filter), we use thresholds ϕ of 0.5, 0.7, and 0.9. Instances that are misclassified by more than ϕ% of the learning algorithms are removed from the training set. The $\mathcal{G}$-filter uses all of the learning algorithms in the set $\mathcal{G}$ (Table 8.1).

Table 8.1: The results for maximizing the 10-fold cross-validation accuracy for HPO and filtering.

	MLP	C4.5	kNN
ORIG	82.28 (2.98)	81.30 (2.91)	80.56 (2.74)
HPO	86.37 (1.87)	84.25 (1.96)	83.89 (2.22)
VS ORIG	45,0,1	42,1,3	34,1,11
A-FILTER	**89.96 (1.13)**	**88.74 (1.09)**	**91.14 (1.02)**
VS ORIG	**46,0,0**	**46,0,0**	**46,0,0**
VS HPO	39,1,6	41,1,4	45,0,1
	NB	RF	RIP
ORIG	77.66 (2.70)	82.98 (2.89)	79.86 (2.96)
HPO	80.89 (1.96)	86.81 (1.85)	82.08 (1.80)
VS ORIG	34,0,12	44,0,2	**46,0,0**
A-FILTER	**82.74 (1.30)**	**91.02 (1.20)**	**88.16 (1.24)**
VS ORIG	**44,2,0**	**43,3,0**	44,2,0
VS HPO	32,0,14	37,0,9	37,0,9

The accuracy on the test set from the value of ϕ that produces the highest accuracy on the training set is reported.

To show the effect of filtering detrimental instances and HPO on an induced model, we examine filtering and HPO in six commonly used learning algorithms (MLP trained with backpropagation, C4.5, kNN, Naïve Bayes, Random Forest, and RIPPER) on a set of 46 UCI data sets [14]. The LWL, NNge, and Ridor learning algorithms are not used for analysis because they do not scale well with the larger data sets–not finishing due to memory overflow or large amounts of running time.[1]

8.5.2 Optimistic Approach

The optimistic approach indicates how well a model *could* generalize on novel data. Maximizing the cross-validation accuracy is a type of overfitting. However, using 10-fold cross-validation accuracy for HPO and filtering, essentially measures the generalization capability of a learning algorithm for a given data set.

[1]For the data sets on which the learning algorithms did finish, the effects of HPO and filtering on LWL, NNge, and Ridor are consistent with the other learning algorithms.

Table 8.2: The frequency of selecting a learning algorithm when adaptively constructing an ensemble filter. Each row gives the percentage of cases that an algorithm was included in the ensemble filter for the learning algorithm in the column.

	ALL	MLP	C4.5	kNN	NB	RF	RIP
None	5.36	2.69	2.95	3.08	5.64	5.77	1.60
MLP	18.33	16.67	15.77	20.00	**25.26**	23.72	16.36
C4.5	17.17	17.82	15.26	22.82	14.49	13.33	20.74
5NN	12.59	11.92	14.23	1.28	10.00	17.18	16.89
LWL	6.12	3.59	3.85	4.36	23.72	3.33	3.59
NB	7.84	5.77	6.54	8.08	5.13	10.26	4.92
NNge	19.49	**26.67**	21.15	21.03	11.15	**24.74**	23.40
RF	**21.14**	22.95	**26.54**	**23.33**	15.77	15.13	**24.20**
Rid	14.69	14.87	16.79	18.33	11.92	16.54	12.77
RIP	8.89	7.82	7.69	8.85	13.08	7.44	4.39

The results comparing the potential benefits of HPO and filtering are shown in Table 8.1. Each section gives the average accuracy and average rank for each learning algorithm as well as the number of times the algorithm is greater than, equal to, or less than a compared algorithm. HPO and the adaptive filter significantly increase the classification accuracy for all of the investigated learning algorithms. The values in bold represent if HPO or the adaptive filter is significantly greater than the other. For all of the investigated learning algorithms, the $\mathcal{A}$-filter significantly increases the accuracy over HPO. The closest the two techniques come to each other is for NB, where the $\mathcal{A}$-filter achieves an accuracy of 82.74% and an average rank of 1.30 while HPO achieves an accuracy of 80.89% and an average rank of 1.96. For all learning algorithms other than NB, the average accuracy is about 89% for filtering and 84% for HPO. Thus, filtering has a greater potential for increase in generalization accuracy. The difficulty lies in how to find the optimal set of training instances.

As might be expected, there is no set of learning algorithms that is the optimal ensemble filter for all algorithms and/or data sets. Table 8.2 shows the frequency for which a learning algorithm with default hyperparameters was selected for filtering by the $\mathcal{A}$-filter. The greatest percentage of cases an algorithm is selected for filtering for each learning algorithm is in bold. The column "ALL" refers to the average from all of the learning algorithms as the base

learner. No instances are filtered in 5.36% of the cases. Thus, given the right filter, filtering to some extent increases the classification accuracy in about 95% of the cases. Furthermore, random forest, NNge, MLP, and C4.5 are the most commonly chosen algorithms for inclusion in the ensemble filter. However, no one learning algorithm is selected in more than 27% of the cases. The filtering algorithm that is most appropriate is dependent on the data set and the learning algorithm. This coincides with the findings from [18] that the efficacy of noise filtering in the nearest-neighbor classifier is dependent on the characteristics of the data set. Understanding the efficacy of filtering and determining *which* filtering approach to use for a given algorithm/data set is a direction of future work.

Analysis. In some cases HPO achieves a lower accuracy than orig, showing the complexity of HPO. The $\mathcal{A}$-Filter, on the other hand, never fails to improve the accuracy. Thus, higher quality data can compensate for hyperparameter settings and suggests that the instance space may be less complex and/or richer than the hyperparameter space. Of course, filtering does not outperform HPO in all cases, but it does so in the majority of cases.

8.5.3 Standard Approach

The previous results show the potential impact of filtering and HPO. We now examine HPO and filtering using the standard approach to highlight the need for improvement in filtering. The results comparing the $\mathcal{G}$-filter, HPO, and using the default hyperparameters trained on the original data set are shown in Table 8.3. HPO significantly increases the classification accuracy over not using HPO for all of the learning algorithms. Filtering significantly increases the accuracy for all of the investigated algorithms except for random forests. Comparing HPO and the G-Filter, only HPO for naïve Bayes and random forests significantly outperforms the $\mathcal{G}$-filter.

Analysis. In their current state, HPO and filtering generally improve the quality of the induced model. The results justify the computational overhead required to run HPO. Despite these results, using the default hyperparameters result in higher classification accuracy for

Table 8.3: The results comparing the performance of using the default hyperparameters, HPO, and the $\mathcal{G}$-filter.

	MLP	C4.5	kNN
ORIG	82.28 (2.54)	81.3 (2.13)	80.56 (2.26)
HPO	83.08 (1.78)	82.42 (1.76)	82.85 (1.59)
VS ORIG	32,0,14	29,1,16	31,5,10
G-FILTER	84.17 (1.61)	81.9 (1.96)	81.61 (2.00)
VS ORIG	39,0,7	23,5,18	27,1,18
VS HPO	22,3,21	19,1,26	17,1,28
	NB	RF	RIP
ORIG	77.66 (2.28)	82.98 (2.28)	79.86 (2.46)
HPO	**81.11 (1.63)**	**84.72 (1.54)**	81.15 (1.74)
VS ORIG	31,2,13	34,0,12	30,2,14
G-FILTER	79.49 (2.02)	83.49 (2.17)	81.25 (1.72)
VS ORIG	28,1,17	25,0,21	37,0,9
VS HPO	16,0,30	13,0,33	20,2,24

11 of the 46 data sets for C4.5, 12 for kNN, and 12 for NB, highlighting the complexity of searching over the hyperparameter space Λ. The effectiveness of HPO is dependent on the data set as well as the learning algorithm. Typically, as was done here, a single filtering technique is used for a set of data sets with no model of the dependence of a learning algorithm on the training instances. The accuracies for filtering and HPO are significantly lower than the optimistic estimate given in Section 8.5.2 motivating future work in HPO and especially in filtering.

8.6 Conclusion

In this paper, we compared the potential benefits of filtering with HPO. HPO may reduce the effects of detrimental instances on an induced model but the detrimental instances are still considered in the learning process. Filtering, on the other hand, removes the detrimental instances–completely eliminating their effects on the induced model.

We used an optimistic approach to estimate the potential accuracy of each method. Using the optimistic approach, both filtering and HPO significantly increase the classification accu-

racy for all of the considered learning algorithms. However, *filtering has a greater potential effect* on average, increasing the classification accuracy from 80.8% to 89.1% on the observed data sets. HPO increases the average classification accuracy to 84.8%. Future work includes developing models to understand the dependence of the performance of learning algorithms given the instances used for training. To better understand how instances affect each other, we are examining the results from machine learning experiments stored in repositories that include which instances were used for training and their predicted class [22, 25]. We hope that the presented results provide motivation for improving the quality of the training data.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[3] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[4] Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine learning*, pages 201–208, 2006.

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

[6] Jeffery L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

[7] Benoit Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[8] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[10] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Learning and Intelligent Optimization Conference*, pages 507–523, 2011.

[11] Jeremy Kubica and Andrew Moore. Probabilistic noise identification and data cleaning. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 131–138, 2003.

[12] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, pages 473–480, 2007.

[13] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[14] Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[15] David F. Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.

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

[17] Umaa Rebbapragada and Carla E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.

[18] José A. Sáez, Julián Luengo, and Francisco Herrera. Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. *Pattern Recognition*, 46(1):355–364, 2013.

[19] Matthias Schonlau, William J. Welch, and Donald R. Jones. *Global versus local search in constrained optimization of computer models*, volume Volume 34 of *Lecture Notes–Monograph Series*, pages 11–25. Institute of Mathematical Statistics, Hayward, CA, 1998.

[20] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[21] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.

[22] Michael R. Smith, Andrew White, Christophe Giraud-Carrier, and Tony Martinez. An easy to use repository for comparing and improving machine learning algorithm usage. In *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, pages 41–48, 2014.

[23] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. 2012.

[24] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.

[25] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[26] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Chapter 9

An Easy to Use Repository for Comparing and Improving Machine Learning Algorithm Usage

In the *ECAI Workshop on Meta-learning & Algorithm Selection (MetaSel)*, pp41-48, 2014.

Abstract

The results from most machine learning experiments are used for a specific purpose and then discarded. This causes significant loss of information and requires rerunning experiments to compare learning algorithms. Often, this also requires a researcher or practitioner to implement another algorithm for comparison, that may not always be correctly implemented. By storing the results from previous experiments, machine learning algorithms can be compared easily and the knowledge gained from them can be used to improve the performance of future machine learning experiments. The purpose of this work is to provide easy access to previous experimental results for learning and comparison. These stored results are comprehensive – storing the prediction for each test instance as well as the learning algorithm, hyperparameters, and training set that were used in the experiment. Previous experimental results are particularly important for meta-learning, which, in a broad sense, is the process of learning from previous machine learning results such that the learning process is improved. While other experiment databases do exist, one of our focuses is on easy access to the data, eliminating any learning curve required to acquire the desired information. We provide meta-learning data sets that are ready to be downloaded for meta-learning experiments. Easy access to previous experimental results aids other researchers looking to do meta-learning and helps in comparing meta-learning algorithms. In addition, simple queries to the underlying database

can be made if specific information is desired. We also differ from previous experiment databases in that our database is designed at the instance level, where an instance is an example in a data set. We store the predictions of a learning algorithm trained on a specific training set for each instance in the test set. Data set level information can then be obtained by aggregating the results from the instances. The instance level information can be used for many tasks such as determining the diversity of a classifier or algorithmically determining the optimal subset of training instances for a learning algorithm.

9.1 Introduction

The quality of an induced model is dependent on, among other aspects, the learning algorithm that is chosen, the hyperparameter settings for the chosen learning algorithm, and the quality of the training set. Choosing a learning algorithm for a given task, setting its hyperparameters, and selecting which instances to train on, however, is non-trivial. Meta-learning deals with the problem of how to select a learning algorithm and set its hyperparameters based on previous experience (results from previous machine learning experiments). Although some research from the machine learning community has focused on meta-learning (e.g., see [2, 3, 5, 8, 17]), much of the focus of machine learning research has been on developing more learning algorithms and/or applying machine learning in specific domains.

Part of the difficulty of meta-learning is due to the lack of accessible results. As meta-learning requires running several learning algorithms and hyperparameter settings over many data sets, gathering results requires large amounts of computational resources. In addition to the computational requirements, results from the learning algorithms may differ due to slight differences in their implementations. Thus, comparing results among meta-learning studies becomes difficult.

To aid in further research in meta-learning, we have developed the *machine learning results repository* (MLRR) that provides data sets ready for download for meta-learning problems, akin to the UCI data repository for machine learning problems. We refer to the

data sets for meta-learning as *meta-data sets* to distinguish them from the data sets that are used in the machine learning experiments. The meta-data sets provide a snapshot of an underlying database that stores the results of machine learning experiments. Users can update the database with new results from machine learning experiments and then update the meta-data sets for meta-learning. A revision history is kept so that comparisons among meta-learning algorithms is facilitated. As a starting point, meta-data sets are provided by MLRR for typical meta-learning tasks, such as, given a set of meta-features, predict which learning algorithm and/or hyperparameter setting to use.

The MLRR stores instance level meta-features and the predictions made on each instance by the learning algorithms. Providing information at the instance level allows studies to be performed on the instances themselves. Studying the effects of machine learning on a single instance and/or the effects of a single instance on the performance of an algorithm has generally been overlooked. Instance-level information is important in several areas of machine learning, however. In ensembles, computing the classifier diversity of the ensembled classifiers using the predictions for each instance is important in determining the effectiveness of the ensembling technique [1, 6, 12]. In curriculum learning, the training set is incrementally augmented such that "easier" instances are presented to the learning algorithm first, thus creating a need to understand and identify the easier instances [4]. Smith et al. used instance-level predictions to identify and characterize instances that are likely to be misclassified [23] and used this information to create a curriculum [22]. Other work has also used the instance-level predictions for meta-learning. The classifier output difference (COD) measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions on test instances [16]. Unsupervised meta-learning (UML) clusters learning algorithms based on their COD scores (rather than accuracy) to examine the behavior of the learning algorithms [13]. Meta-learning for algorithm selection can then be done over the clusters rather than a larger set of learning algorithms to recommend a cluster of learning algorithms that all behave similarly [14]. Additionally, several techniques

treat instances individually during the training process, such as filtering instances from the training set based on their instance-level meta-features [21] or weighting the instances [18].

Other attempts have been made at creating a repository for machine learning experiments from which learning can be conducted [20, 24]. However, we feel that they lack simplicity and/or extensibility. In addition to providing instance-level information, we hope to bridge this gap with the MLRR. Probably the most prominent and well-developed data repository is ExpDB, an experiment database that provides a framework for reporting experimental results and their associated workflow [24]. The purpose of ExpDB is to comprehensively store the workflow process of all experiments for reproducibility. One of the results of storing the experiments is that the results can be used for meta-learning. Unfortunately, there is a relatively steep learning curve to access the data due to the inherent complexity involved in storing all of the details about exact reproducibility. Because of this complexity and formality, it is difficult to directly access the information that would be most beneficial for meta-learning, which may deter some potential users. Additionally, ExpDB does not currently support storage and manipulation of any instance level features.

We acknowledge that maintaining a database of previous experiments is not a trivial problem. We do, however, add our voice to support the importance of maintaining a repository of machine learning results and offer an effective solution for storing results from previous experiments. Our primary goal is to maintain simplicity and provide easily accessible data for meta-learning to 1) help promote more research in meta-learning, 2) provide a standard set of data sets for meta-learning algorithm comparison, and 3) continue to stimulate research at the instance level.

We next describe our approach for providing a repository for machine learning meta-data that emphasizes ease of access to the meta-data. MLRR currently has the results from 72 data sets, 9 learning algorithms and 10 hyperparameter settings for each learning algorithm. The database description is provided in Section 9.3. How to add new experimental results to the database is detailed in Section 9.4. We then give a more detailed description of the

data set level and instance level meta-features that are used in the MLRR. Conclusions and directions for future work are provided in Section 9.6.

9.2 Meta-data Set Descriptions

The purpose of the *machine learning results repository* (MLRR) is to provide easy access to the results of previous machine learning experiments for meta-learning at the data set and instance levels. This, in turn, would allow other researchers interested in meta-learning and in better understanding machine learning algorithms direct access to prior results without having to re-run all of the algorithms or learn how to navigate a more complex experiment database. The quality of an induced model for a task is dependent on at least three things:

1. the learning algorithm chosen to induce the model,
2. the hyperparameter settings for the chosen learning algorithm, and
3. the instances used for training.

When we refer to an experiment, we mean the results from training a learning algorithm l with hyperparamter settings λ on a training set t. We first describe how we manage experiment information, and then describe the provided meta-data sets.

9.2.1 Experiment Information

The information about each experiment is provided in three tables in MLRR. Which learning algorithm and hyperparameters were used is provided in a file structured as shown in Table 9.1. It provides the toolkit including the version number that was ran, the learning algorithm, and the hyperparameters that were used. This allows for multiple learning algorithms, hyperparameters, and toolkits to be compared. In the examples in Table 9.1, the class names from the Weka machine learning toolkit [9] and the Waffles machine learning toolkit [7] are shown. LA_seed corresponds to the learning algorithm that was used (LA) and to a seed that represents which hyperparameter setting was used (seed). The LA_seed

Table 9.1: The structure of the meta-data set that describes the hyperparameter settings for the learning algorithms stored in the database.

LA_S	Toolkit	Version	Hyperparameters
BP_1	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.261703 -M 0.161703 -H 12 -D
BP_2	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.25807 -M 0.15807 -H 4
BP_3	waffles	13-12-09	neuralnet -addlayer 8 -learningrate 0.1 \ -momentum 0 -windowsepochs 50
⋮	⋮	⋮	⋮
C4.5_1	weka	3.6.11	weka.classifiers.trees.J48 – -C 0.443973 -M 1
⋮	⋮	⋮	⋮

Table 9.2: The structure of the table for mapping learning algorithm hyperparameters between different toolkits for the backpropagation learning algorithm.

	Command line parameters				
toolkit	LR	Mo	HN	DC	WE
weka	-L	-M	-H	-D	?
waffles	-learningrate	-momentum	-addlayer	?	-windowsepochs
⋮	⋮	⋮	⋮	⋮	⋮

will be used in other tables as a foreign key to map back to this table. A seed of -1 represents the default hyperparameter settings as many studies examine the default behavior as given in a toolkit and the default parameters are commonly used in practice.

As the parameter values differ between toolkits, there is a mapping provided to distinguish hyperparameter settings. For example, Weka uses the "-L" parameter to set the learning rate in backpropagation while the Waffles toolkit uses "-learningrate". Also, some toolkits have hyperparameters that other implementations of the same learning algorithm do not include. In such cases, an unknown value will be provided in the meta-data set. This mapping is shown in Table 9.2 for the backpropagation learning algorithm. The first row contains the values used by MLRR. The following rows contain the command-line parameter supplied to a specific toolkit to set that hyperparameter.

Table 9.3: The structure of the meta-data set that indicates which instances were used for training given a random seed.

toolkit_seed_# folds_fold	1	2	3	...
weka_1_10_1	1	1	1	...
weka_1_10_2	1	0	1	...
⋮	⋮	⋮	⋮	
weka_1_10_10	0.74	1	?	...
weka_2_1_10	?	1	1	...
⋮	⋮	⋮	⋮	

A mapping of which instances are used for training is also provided in a separate file. The structure of this table is shown in Table 9.3. Each row represents an experiment as toolkit_seed_numFolds_fold. The toolkit represents which toolkit was used, the seed represents the random seed that was provided to the toolkit, numFolds represents how many folds were ran, and fold represents in which fold an instance was included for testing. The values in the following columns represent if an instance was used for training or testing. There is one column for each instance in the data set. They are stored as real values. This allows for the situations when training instances have associated weights. In the file, an unknown value of "?" represents a testing instance, otherwise a real value represents a training instance. A value of 0 represents a filtered instance, a value of 1 represents an unweighted training instance and any value between 0 and 1 represents the weight for that training instance. In the cases where there are specific training and testing sets, then the row will be labeled as toolkit_0_0_1 and information for the training set can be entered as before. A random test/training split of the data is represented as toolkit_seed_percentSplit_1 where "percentSplit" represents the percentage of the data set that was used for testing as generated by the toolkit.

9.2.2 Meta-data sets

One of the features of MLRR is its focus on storing and presenting instance level information, namely, instance level characteristics and associated predictions from previous experiments. Indeed, the MLRR is designed intentionally from the instance level perspective, from which data set level information can be computed (e.g., accuracy or precision).

As one of the purposes of the MLRR is ease of access, the MLRR stores several data sets in attribute-relation file format (ARFF) which is supported by many machine learning toolkits. In essence, ARFF is a comma or space separated file with attribute information and possible comments. The precomputed meta-data sets include instance level meta-data sets and data set level meta-data sets.

At the instance level, MLRR provides for each data set a meta-data set that stores the instance level meta-features and the prediction from each experiment. This allows for analyses to be done exploring the effects of hyperparameters and learning algorithms at the instance-level, which is currently mostly overlooked. For each data set, a meta-data set is provided that gives the values for the instance level meta-features, the actual class value (stored as a numeric value), and the predicted class value for each experiment. The training set and learning algorithm/hyperparameter information is stored in the column heading as "LA_seed/hyperparameter" where LA is a learning algorithm and hyperparameter is the hyperparameter setting for the learning algorithm. Together, they map to the entries in Table 9.1. The seed represents the seed that was used to partition the data (see Table 9.3). The structure of the instance level meta-data set is shown in Table 9.4. In the given example, instance 77 is shown. The "inst meta" section provides the instance level meta-features for that instance. The actual class label is 2. The predictions from the experiments on this data set are provided in the following columns (i.e., experiment BP_1/1 predicted class 3, BP_N/1 predicted class 2, etc.).

At the data set level, several meta-data sets are provided:

Table 9.4: The structure of the meta-data set at the instance level.

	inst meta				predictions								
#	kAN	MV	...	act	BP_1/1	...	BP_N/1	...	BP_N/M		C4.5_1/1		...
77	0.92	0	...	2	3	...	2	...	2		3		...
⋮	⋮	⋮		⋮	⋮		⋮		⋮		⋮		

Table 9.5: The structure of the meta-data set at the data set level.

	data set meta-features			LA accuracies					
data set	numInst	numAttr	...	BP_1	BP_2	...	BP_N	C4.5_1	...
iris	150	4	...	96.80	95.07	...	93.47	95.60	...
abalone	4177	8	...	20.27	29.84	...	21.91	23.24	...
⋮	⋮	⋮		⋮	⋮	⋮	⋮	⋮	

- a general meta-data set that stores the data set meta-features and the average N by 10-fold cross-validation accuracy for all of the data sets from a learning algorithm with a given hyperparameter setting.
- for each learning algorithm a meta-data set that stores the data set meta-features, the learning algorithm hyperparameter settings, and the average N by 10-fold cross-validation accuracy for all of the data sets for the given hyperparameter setting.

The structure for the general meta-data set is provided in Table 9.5. The structure and information of this meta-data set is typical of that used in previous meta-learning studies that provides a mapping from data set meta-features to accuracies obtained by a set of learning algorithms. Most previous studies have been limited to only using the default hyperparameters, however. The MLRR includes the accuracies from multiple hyperparameter settings. The hyperparameter settings from each learning algorithm are denoted by a "LA_#" where LA refers to a learning algorithm and # refers to which hyperparameter setting was used for that learning algorithm.

The meta-data sets for each learning algorithm are designed to aid in algorithmic hyperparameter estimation, i.e., given a data set, can we predict which hyperparameter setting will give the highest classification accuracy. For each learning algorithm, a meta-data set is

Table 9.6: The structure of the table for mapping learning algorithm hyperparameters among toolkits.

	DS meta features			toolkit	hyperparameters			
data set	numInst	numAttr	...	weka	LR	Mo	...	acc
iris	150	4	...	weka	0.71	0.61	...	96.80
iris	150	4	...	weka	0.11	0.25	...	97.04
⋮	⋮	⋮		⋮	⋮	⋮		⋮

provided that contains the data set meta-features, the toolkit that was used, the hyperparameter setting and the average accuracy for each unique tool kit/hyperparameter combination. The structure of the meta-data set for each learning algorithm is provided in Table 9.6. The accuracy ("acc") represents the average accuracy for all k-fold validation runs (i.e., multiple runs of the same learning algorithm with different random seeds to partition the folds). The toolkit is also provided to allow a user to compare toolkits or only do hyperparameter estimation for a single toolkit.

MLRR provides easy access for researchers and practitioners to a large and varying set of meta-data information as shown in the tables above. The provided meta-data sets are a snapshot of an underlying database that stores all of the previous experimental results that can be updated as more results are obtained. A revision history of the data sets is provided so that results can be compared even if the meta-data set has been updated.

9.3 Database Description

MLRR uses MongoDB as the database to store the results from machine learning experiments. MongoDB is a NoSQL database that allows for adding new features (such as new learning algorithms and/hyperparameters), thus, escaping the rigidity of the more traditional SQL databases. This allows for easily expanding the database with new learning algorithms and/or hyperparameters. Of course, this is theoretically also possible in a relational database, provided the database has been designed adequately. For example, one could certainly have, and that would indeed be following good design principles, one table for the algorithms and

one table for the hyper parameters with appropriate foreign keys. However, such design requires some amount of foresight. In traditional relational databases, the information that needs to be stored (and how) has to be planned for in advance. Otherwise, when new features are desired, a new schema needs to be created and then the database has to be migrated over to the new schema. With a NoSQL database, new learning algorithms/hyperparameters and other pieces of information can easily be added into the MLRR.

The data is stored as a document database as collections of key-value pairs. Each collection represents the experimental results on a particular data set. In each collection, the keys are LA_hyperparameterSetting. The value then is a JSON text document that stores the results of an experiment (e.g., the results of 10-fold cross-validation on the iris data set using C4.5). These documents also contain pointers to other documents that hold information about training/testing sets for each experiment. The data set/instance level meta-features are stored in separate documents in their respective data set collection. A separate collection stores information about the learning algorithms and their hyperparameters.

The best way to visualize the database is as a hierarchy of key-value pairs as shown in Figure 9.1. At the top-level, there are collections - these are the individual data sets in the database. Each of them holds a collection of documents that represent an output file, or experiment, named by its learning algorithm with two numbers that correspond to the random seed used to partition the data and the hyperparameter setting. In these documents, the predictions for each instance is stored. Collections for which instances were used for training hyperparameter settings are also included.

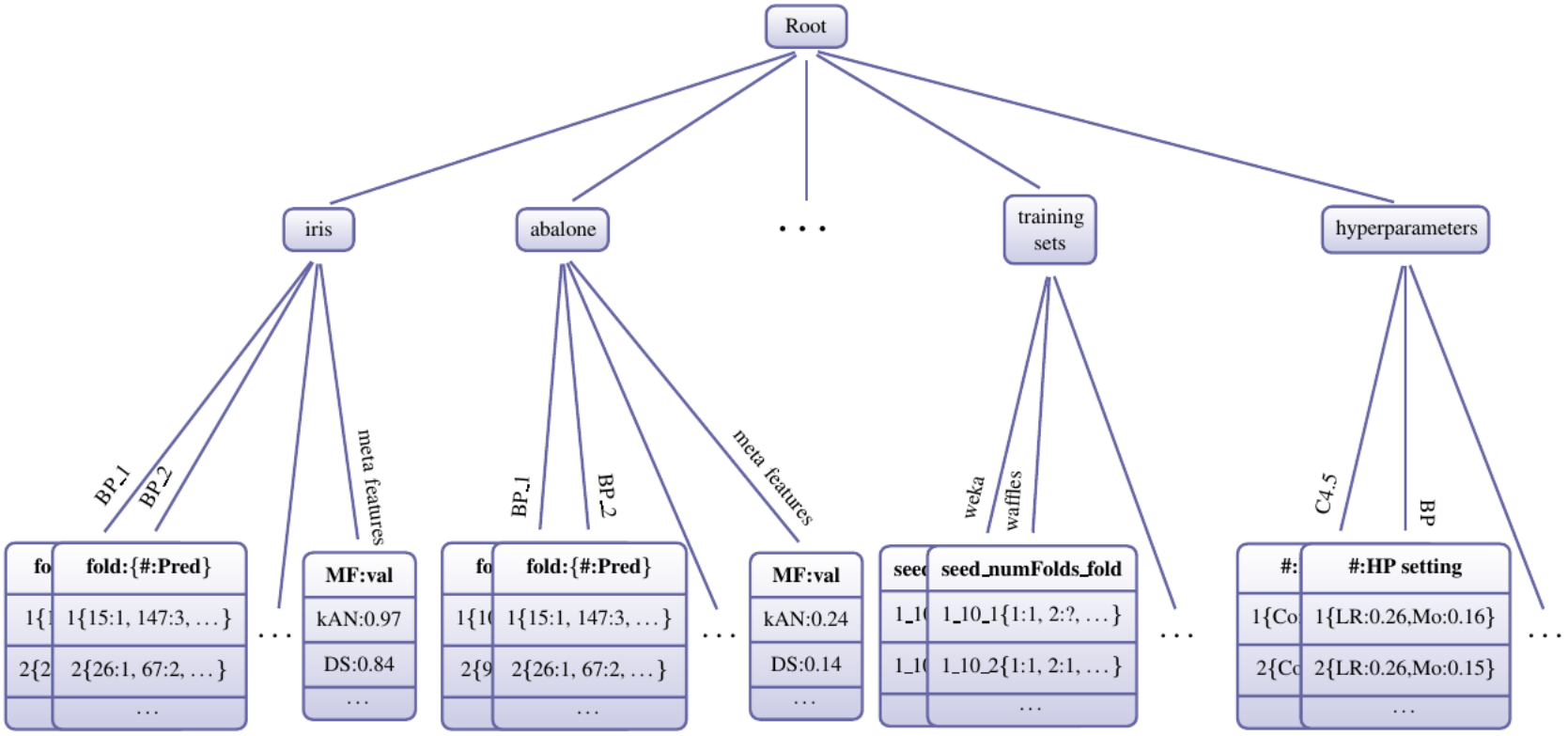

Figure 9.1: Hierarchical representation of how the results from machine learning experiments are stored in the NoSQL database for the MLRR. Each data set has a collection containing the predictions for each instance from a learning algorithm as well as its meta-features. A separate collection stores all of the information for the learning algorithms and which hyperparameters were used. Another collection stores the information for which instances were used for training.

9.4 Extending the Database

The data provided by MLRR only contains a snapshot of current machine learning results. To allow more machine learning results to be added and to allow the MLRR to evolve as the state of machine learning evolves, MLRR provides a method to upload new machine learning results. The MLRR also stores the original data sets to allow a user to add results from additional experiments on the current set of data sets. The results from experimentation on a new data set require that the new data set be uploaded as well as the experimental results. Scripts are provided to calculate the meta-features for the new data set. In the case where a data set is proprietary or has other privacy/licensing issues that prevent it from being posted, the meta-features can be calculated on the data set without storing the actual data set.

Currently, scripts are provided to upload the output from running Weka. This provides a simple way to upload experimental results from a commonly used toolkit. The file is slightly modified such that the first line provides which learning algorithm and hyperparameters were used. The database will have the ability to upload files generated by other toolkits in the future.

Of course, there are issues of data reliability. Currently, all of the results stored in the MLRR are from our experiments. To help with data reliability, we require that the script(s) and executable(s) required to reproduce the results are uploaded along with the results. This allows the results to be verified if their validity is questioned. If the results from an experiment are thought to be invalid, they can can be flagged, and inspected for possible removal from the MLRR.

9.5 Included Meta-features

In this section, we detail the meta-features that are included in the machine learning results repository (MLRR). We store a set of data set meta-features that have been commonly used

in previous meta-learning studies. Specifically, we used the meta-features from Brazdil et al. [5], Ho and Basu [10], Pfahringer et al. [17], and Smith et al. [23]. As the underlying database is a NoSQL database, additional meta-features can be easily added in the future. We now describe the meta-features from each study.

The study by Brazdil et al. [5] examined ranking learning algorithms using instance-based learning. The meta-features are designed to be quickly calculated and to represent properties that affect algorithm performance.

- *Number of examples.* This feature helps identify how scalable an algorithm is based on the size of its input.
- *Proportion of symbolic attributes.* This feature can be used to consider how well an algorithm deals with symbolic or numeric attributes.
- *Proportion of missing values.* This features can be used to consider how robust an algorithm is to incomplete data.
- *Proportion of attributes with outliers.* An attribute is considered to have an outlier if the ratio of variances of the mean value and the α-trimmed mean is smaller than 0.7 where $\alpha = 0.05$. This feature can be used to consider how robust an algorithm is to outlying numeric values.
- *Entropy of classes.* This feature measures one aspect of problem difficulty in the form of whether one class outnumbers another.

Ho and Basu [10] sought to measure the complexity of a data set to identify areas of the data set that contribute to its complexity focusing on the geometrical complexity of the class boundary.

- Measures of overlap of individual feature values:

– *The maximum Fisher's Discriminant ratio.* This is the Fisher's discriminant ratio for an attribute:

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2},$$

where μ_i and σ_i^2 represent the mean and variance for a class. The maximum Fisher's discriminant value over the attributes is used for this measure. For multiple classes, this measure is expanded to:

$$f = \frac{\sum_{i=1}^{C} \sum_{j=i+1}^{C} p_i p_j (\mu_i - \mu_j)^2}{\sum_{i=1}^{C} p_i \sigma_i^2}$$

where C is the number of classes and p_i is the proportion of instances that belong to the i^{th} class.

– *The overlap of the per-class bounding boxes.* This feature measures the overlap of the tails of the two class-conditional distributions. For data sets with more than 2 classes, the overlap of the per-class bounding boxes is computed for each pair of classes and the sum over all pairs of classes is returned.

– *The maximum (individual) feature efficiency.* This feature measures how discriminative a single feature is. For each attribute, the ratio of instances with differing classes that are not in the overlapping region is returned. The attribute that produces the largest ratio of instances is returned.

– *The collective feature efficiency.* This measure builds off of the previous one. The maximum ratio is first calculated as before. Then, the instances that can be discriminated are removed and the maximum (individual) feature efficiency is recalculated with the remaining instances. This process is repeated until no more instances can be removed. The ratio of instances that can be discriminated is returned.

- Measures of class separability:

- *The minimized sum of the error distance of a linear classifier.* This feature measures to what extent training data is linearly separable and returns the difference between a linear classifier and the actual class value.
- *The training error of a linear classifier.* This feature also measures to what extent the training data is linearly separable.
- *The fraction of points on the class boundary.* This feature estimates the length of the class boundary by constructing a minimum spanning tree over the entire data set and returning the ratio of the number of nodes in the spanning tree that are connected and belong to different classes to the number of instances in the data set.
- *The ratio of average intra/inter class nearest neighbor distance.* This measure compares the within class spread with the distances to the nearest neighbors of the other classes. For each instance, the distance to its nearest neighbor with the same class ($intraDist(x)$) and to its nearest neighbor with a different class ($interDist(x)$) is calculated. Then the measure returns:

$$\frac{\sum_i^N intraDist(x_i)}{\sum_i^N interDist(x_i)}$$

  where N is the number of instances in the data set.
- *The leave-one-out error rate of the one-nearest neighbor classifier.* This feature measures how close the examples of different classes are.

- Measures of geometry, topology, and density of manifolds
  - *The nonlinearity of a linear classifier.* Following Hoekstra and Duin [11], given a training set, a test set is created by linear interpolation with random coefficients between pairs of randomly selected instances of the same class. The error rate of a linear classifier trained with the original training set on the generated test set is returned.

- *The nonlinearity of the one-nearest neighbor classifier.* A test set is created as with the previous feature, but the error rate of a 1-nearest neighbor classifier is returned.
- *The fraction of maximum covering spheres.* A covering sphere is created by centering on an instance and growing as much as possible before touching an instance from another class. Only the largest spheres are considered. The measure returns the number of spheres divided by the number of instances in the data set and provides an indication of how much the instances are clustered in hyperspheres or distributed in thinner structures.
- *The average number of points per dimension.* This measure is the ratio of instances to attributes and roughly indicates how sparse a data set is.

Multi-class modifications are made according to the implementation of the data complexity library (DCoL) [15].

Pfahringer et al. [17] introduced the notion of using performance values (i.e., accuracy) of simple and fast classification algorithms as meta-features. The landmarkers that are included in the MLRR are listed below.

- *Linear discriminant learner.* Creates a linear classifier that finds a linear combination of the features to separate the classes.
- *One nearest neighbor learner.* Redundant with the leave-one-out error rate of the one-nearest neighbor classifier from Ho and Basu [10].
- *Decision node learning.* A decision stump that splits on the attribute that has the highest information gain. A decision stump is a decision tree with only one node.
- *Randomly chosen node learner.* A decision stump that splits on a randomly chosen attribute.
- *Worst node learner.* A decision stump that splits on the attribute that has the lowest information gain.

- *Average node learner.* A decision stump is created for each attribute and the average accuracy is returned.

The use of landmarkers has been shown to be competitive with the best performing meta-features with a significant decrease in computational effort [19].

Smith et al. [23] sought to identify and characterize instances that are difficult to classify correctly. The difficulty of an instance was determined based on how frequently it was misclassified. To characterize why some instances are more difficult than others to classify correctly, the authors used different hardness measures. They include:

- *k-Disagreeing Neighbors.* The percentage of k nearest neighbors that do not share the target class of an instance. This measures the local overlap of an instance in the original space of the task.
- *Disjunct size.* This feature indicates how tightly a learning algorithm has to divide the task space to correctly classify an instance. It is measured as the size of a disjunct that covers an instance divided by the largest disjunct produced, where the disjuncts are formed using the C4.5 learning algorithm.
- *Disjunct class percentage.* This features measure the overlap of an instance on a subset of the features. Using a pruned C4.5 tree, the disjunct class percentage is the number of instances in a disjunct that belong to the same class divided by the total number of instances in the disjunct.
- *Tree depth (pruned and unpruned).* Tree depth provides a way to estimate the description length, or Kolmogorov complexity, of an instance. It is the depth of the leaf node that classifies an instance in an induced tree.
- *Class likelihood.* This features provides a global measure of overlap and the likelihood of an instance belonging to the target class. It is calculated as:

$$\prod_{i}^{|x|} p(x_i|t(x))$$

where $|x|$ represents the number of attributes for the instance x and $t(x)$ is the target class of x.

- *Minority value.* This feature measures the skewness of the class that an instance belongs to. It is measured as the ratio of instances sharing the target class of an instance to the number of instances in the majority class.
- *Class balance.* This feature also measures the class skew. First, the ratio of the number of instances belonging the target class to the total number of instances is calculated. The difference of this ratio with the ratio of one over the number of possible classes is returned. If the class were completely balanced (i.e. all class had the same number of instances), a value of 0 would be returned for each instance.

The hardness measures are designed to capture the characteristics of why instances are hard to classify correctly. Data set measures can be generated by averaging the hardness measures over the instances in a data set.

9.6 Conclusions and Future Work

In this paper, we presented the *machine learning results repository* (MLRR) an easily accessible and extensible database for meta-learning. MLRR was designed with the main goals of providing an easily accessible data repository to facilitate meta-learning and providing benchmark meta-data sets to compare meta-learning experiments. To this end, the MLRR provides ready to download meta-data sets of previous experimental results. One of the important features of MLRR is that it provides meta-data at the instance level. Of course, the results could also be used as a means of comparing one's work with prior work as they are stored in the MLRR. The MLRR can be accessed at `http://axon.cs.byu.edu/mlrr`.

The MLRR allows for reproducible results as the data sets are stored on the server and as the class names and toolkits are provided. The ExpDB tends to be a lot more rigid in its design as it is based on relational databases and PMML (predictive model markup language),

thus exhibiting a relatively steep learning curve to import and extract data. The MLRR is less rigid in its design allowing for easier access to the data and more extensibility, with the trade-off of less formality.

One direction for future work is to integrate the API provided at OpenML[1] (an implementation of an experiment database) to incorporate their results with those that are in the MLRR. This will help provide easy access to the results that are already stored in OpenML without having to incur the learning cost associated with understanding the database schema.

Another open problem is how to store information about how a data set is preprocessed. Currently, the MLRR can store the instance level information resulting from preprocessing, but it lacks a mechanism to store the preprocessing process. Integrating this information in an efficient way is a direction of current research.

References

[1] Matti Aksela and Jorma Laaksonen. Using diversity of errors for selecting members of a committee classifier. *Pattern Recognition*, 39(4):608–623, 2006.

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

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

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[5] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003. doi: http://dx.doi.org/10.1023/A:1021713901879.

[1] www.openml.org

[6] Gavin Brown, Jeremy L. Wyatt, and Peter Tino. Managing diversity in regression ensembles. *Journal of Machine Learning Research*, 6:1621–1650, 2005.

[7] Michael S. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, MLOSS 12:2383–2387, July 2011. ISSN 1532-4435.

[8] Taciana A. F. Gomes, Ricardo Bastos Cavalcante Prudncio, Carlos Soares, Andr L. D. Rossi, and Andr C. P. L. F. Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[10] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:289–300, March 2002.

[11] Aarnoud Hoekstra and Robert P.W. Duin. On the nonlinearity of pattern classifiers. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 271–275, 1996.

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

[13] Jun Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[14] Jun Lee and Christophe Giraud-Carrier. Automatic selection of classification learning algorithms for data mining practitioners. *Intelligent Data Analysis*, 17(4):665–678, 2013.

[15] Albert Orriols-Puig, Núria Macià, Ester Bernadó-Mansilla, and Tin Kam Ho. Documentation for the data complexity library in c++. Technical Report 2009001, La Salle - Universitat Ramon Llull, April 2009.

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

[17] Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, pages 743–750, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2.

[18] Umaa Rebbapragada and Carla E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.

[19] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel. Automatic classifier selection for non-experts. *Pattern Analysis & Applications*, 17(1):83–96, 2014.

[20] Matthias Reif. A comprehensive dataset for evaluating approaches of various meta-learning tasks. In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pages 273–276. SciTePress, 2012.

[21] Michael R. Smith and Tony Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.

[22] Michael R. Smith and Tony Martinez. A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. *Computational Intelligence*, page to appear, 2014. URL `http://arxiv.org/pdf/1312.4986`.

[23] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.

[24] Joaquin Vanschoren, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. Experiment databases - a new way to share, organize and learn from experiments. *Machine Learning*, 87(2):127–158, 2012.

Chapter 10

Conclusions, Contributions, and Remaining Challenges

This dissertation addresses understanding the dependency between the training data used in machine learning problems, the learning algorithms, and their associated hyperparameter settings. The ultimate goal of using this knowledge would be to, given a data set, automatically 1) preprocess the data set, 2) select a learning algorithm, and 3) set the associated hyperparameters for the selected learning algorithm such that the induced hypothesis is most appropriate for the task. The most appropriate hypothesis is determined by the results from previous machine learning experiments that was used for meta-learning. This dissertation has established a foundation and tools upon which future research can be built to take a more principled approach to machine learning.

10.1 Summary and Contributions

The quality of an induced hypothesis in machine learning is dependent on 1) the quality of the training data, 2) the learning algorithm selected, and 3) the associated hyperparameters for the selected learning algorithm, as discussed in Chapter 1. Preprocessing the training data, selecting a learning algorithm, and setting its hyperparameters is a challenging task, even for machine learning experts and is often done heuristically. Most of the previous works have only considered learning algorithm selection or hyperparameter selection/optimization in isolation. This approach was shown to be beneficial, but was often criticized because the performance a learning algorithm is dependent on the hyperparameter settings and because some learning algorithms are better suited for certain tasks. An examination of the

quality of the data at the instance-level has received very little attention in previous work. Understanding the principles of how individual instances affect the induced hypothesis allows for a more principled approach for applying machine learning to a specific task.

Chapter 3 introduces instance hardness and the hardness measures, which identify and characterize instances that are hard to correctly classify. This is, to our knowledge, the first work that examines systematically the influence of each training instance on the induced model. The work on instance hardness confirms many of the reasons for an instance being misclassified, namely due to class overlap and dispels the magnitude of impact for class imbalance, which has a minor impact if there is no class overlap. This chapter lays the ground work for Chapters 4 through 6.

Chapters 4, 5, and 6 present methods for integrating the instance-level information into the learning process through filtering instances that should be misclassified, weighting the instances, and curriculum learning respectively. Chapter 7 shows the that incorporating classifier diversity in ensembles and filtering is more robust to noise.

Chapter 8 compares the potential and current state of filtering and hyperparameter optimization. In their current state, both techniques perform similarly. However, when examining their potential, filtering has a much greater potential than hyperparameter optimization. This is because most learning algorithms do not consider instances individually and they do a global optimization.

Chapter 9 presents the Machine Learning Results Repository, a repository for the machine learning community for storing the results of previous experiments. The repository addresses the problem of obtaining data for meta-learning problems. It also is the first repository to store results at the instance-level – allowing for instance-level meta-learning. Storing the results at the instance-level also allows for a wide variety of data set-level measures to be calculated.

10.2 Directions for Future Work

This dissertation lays the groundwork for facilitating a more principled approach to machine learning. This dissertation presents one method (instance hardness) for analyzing instances. One direction for future work includes examining how each instance affects the induced model. The brute force method would be to train on each subset of instances in the power set of the training data. A more feasible solution would be to induce hypotheses on overlapping subsets of the training data such that each instance is included in multiple subsets. The hypotheses can then be analyzed when each instance was used for training.

One area that this dissertation did not cover is feature selection/extraction. The features have a large impact on the quality of the training data. Thus, in addition to selecting which instances to use for training, the features should also be considered.

One disadvantage of the current approaches for integrating instance information into the learning process is that it requires a two-step process: 1) calculate the instance-level information, and then 2) integrate this information into the learning process. Developing a one-step process would be beneficial. However, detrimental instances have the greatest impact in the early stages of training. One possible solution would be to use the hardness measures to determine how to handle each instance. The action to be taken for the instances in a new data set could be learned using the results from previous experiments.

Of course understanding how the instances affect each other and which features are the most predictive involves a very large search space. Thus, in our estimation, the development of tools such as the machine learning results repository are of great worth moving forward. The repository represents a large collection of results that can be used to do meta-learning. This will facilitate more researchers with the resources for conducting research in meta-learning. Additionally, we hope that the repository will bring researchers together and facilitate collaboration. Of course, there are a number of issues revolving around the machine learning results repository that need to be addressed.

1. Easy access of data. Most researchers are busy and do not want to have to learn one more thing to access data. The repository was designed to be simple so that researchers do not have to learn how to use the web site.

2. Validity of the data. Having a community-based resource, there needs to be some way to determine the quality of the data and protect against faulty data. There are a number of approaches ranging from running the experiments on our own servers following our specified protocols to a community-based validation system when questionable results can be flagged.

3. Incentive to users. Using the repository is obviously optional. Thus, how can we motivate other researchers to upload their results to repository for the greater good of the community as a whole? Currently, we are looking at partnering with conferences to help in creating reproducible results that would require the use of the repository. We are also examining the development of plug-ins for popular machine learning tool kits.

4. Accommodating various formats. Each user will store their results in different formats. How can the repository be robust enough to handle a variety of results formats and provide a simple method for uploading results?

These are a few of the challenges facing the repository.

Once data is accumulated, meta-learning can be pursued more efficiently. Additional tools can be created specifically for meta-learning. One such work is the use of collaborative filtering techniques for meta-learning. The latent neural network (LNN) was developed as a means for addressing the issue of non-predictive meta-features. However, the LNN only *exploits* the learning algorithms and associated hyperparameters that it has information about. The LNN cannot *explore* or suggest to explore new options. Thus, a combination of exploitation and exploration can be examined. One starting point could be to provide

a learning mechanism to Bayesian optimization techniques to leverage the exploitation of previous results while also being able to explore new areas of the hypothesis space.

While meta-learning is still in its infancy, taking a more principled approach to machine learning is already starting to take hold. The importance of meta-learning will increase as more institutions look to use machine learning techniques to address their problems.