UTILIZING CLASS-SPECIFIC THRESHOLDS DISCOVERED BY OUTLIER DETECTION

A Thesis Submitted to the Committee on Graduate Studies in Partial Fulfillment of the Requirements for the Degree of Master of Science in the Faculty of Arts and Science

TRENT UNIVERSITY

Peterborough, Ontario, Canada

Applied Modelling and Quantitative Methods

M.Sc. Graduate Program

September, 2016

ABSTRACT

Utilizing Class-Specific Thresholds Discovered by Outlier Detection

Richard Arthur Conan Branch

We investigated if the performance of selected supervised machine-learning techniques could be improved by combining univariate outlier-detection techniques and machine-learning methods. We developed a framework to discover class-specific thresholds in class probability estimates using univariate outlier detection and proposed two novel techniques to utilize these class-specific thresholds. These proposed techniques were applied to various data sets and the results were evaluated. Our experimental results suggest that some of our techniques may improve recall in the base learner. Additional results suggest that one technique may produce higher accuracy and precision than AdaBoost.M1, while another may produce higher recall. Finally, our results

suggest that we can achieve higher accuracy, precision, or recall when AdaBoost.M1 fails to produce higher metric values than the base learner.

Keywords: Machine Learning, Classification, Outliers, Outlier Detection, Boosting, Probability Estimates, Class-Specific Thresholds, GenThresh, OutBoost, AdaBoost

Acknowledgments

This research has been a collaborative effort of many individuals. Without whose support I would not have been able to succeed. I would like to take this opportunity to thank these many individuals.

A very special thank you to Dr. Sabine McConnell and Dr. Richard Hurley. Both of whom have been an inspiration to me. I would not have been able to make this journey without their continued guidance, patience, and motivation. They have been a constant driving force, continuing to push me towards my goals and stretching my abilities.

Thank you to my examining committee for taking the time to participate in my defense. Thank you to my colleagues, mentors, and advisors. This journey would have been a difficult one without their company, advice, and encouragement. Thank you to the Trent community, faculty, and staff for assisting me in my journey.

A special thank you to NSERC and OGS for financially supporting my

research. Without their support I would not have been able to achieve my goals.

Thank you to my parents, siblings, and extended family. My parents raised me to be inquisitive and critical and these traits have served me well. Thank you to my brother and sister-in-law for having faith in my journey. They have been a constant source of encouragement.

Thank you to my second family: George, Karen, and Sonia. Their encouragement, affection, and personal support has been critical to my success. They have been a constant source of fun, joy, inspiration, and dessert.

Lastly, I would like to give an extra special thank you to Amanda. Amanda is a source of never ending love and encouragement. Without her confidence and support this would not have been possible. She had faith in me even when I did not have faith in myself. Thank you for being patient with my obsessive work habits and always being there for me in my times of need.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ANN - Artificial Neural Network

BBN - Bayesian Belief Network

CV - Cross Validation

DT - Decision Tree

FN - False Negatives

FP - False Positives

GNU - GNU's Not Unix!

KDD - Knowledge Discovery in Databases

LR - Logistic Regression

MAP - Maximum a Posteriori Estimate

NBC - Naive Bayes Classifier

SMO - Sequential Minimal Optimization

SVM - Support Vector Machine

TN - True Negatives

TP - True Positives

TPR - True Positive Rate

UCI - University of California, Irvine

WEKA - Waikato Environment for Knowledge Analysis

Chapter 1

Introduction

1.1 Preliminaries

We have entered an era of *Big Data* where it is estimated that we create 2.5 Exabytes of data each day [18–20]. We are called to increase our employment of automated methods of analyzing and discovering knowledge in data in order to handle this large volume of data [22]. *Machine learning* responds to this call by providing methods for automatically finding patterns in data, patterns that enable us to make decisions when dealing with uncertainty [22]. In essence, machine learning enables us to learn from data in order to make predictions regarding the future.

Outlier detection and *outlier analysis* explore the underlying abnormal

characteristics of data while playing an important role in our ability to gain insight into data [2]. *Outliers* in data often translate to significant and critical information for a wide variety of application domains in science and industry [4].

1.2 Thesis Statement

In this thesis, we ask if the performance of supervised machine-learning techniques can be improved by combining univariate outlier detection and traditional machine-learning methods. Specifically, we investigate if we can apply univariate outlier detection to estimated class probabilities in order to construct class-specific thresholds. We then investigate if we can utilize these thresholds to improve the performance of machine-learning methods.

1.3 Objectives and Goals

We defined the following three general objectives for this thesis:

1. To create a framework to define outliers in class probability estimates.

2. To propose two techniques to utilize the outliers defined in the class probability estimates in the form of class-specific thresholds.

3. To determine if these proposed techniques can improve the performance of a variety of classification techniques on a variety of data sets.

1.4 General Outline

We provide a general outline of our thesis in the following section and describe the contents of each chapter.

Chapter 1 briefly introduces machine learning and outlier detection, describing the inspiration for our research, our research question, our key objectives, and our research goals.

We give a brief background of the relevant concepts, algorithms, and terminology used in our thesis in Chapter 2. We focus primarily on supervised machine learning, classification, outliers, and detection of outliers. A brief introduction is provided for probability estimates, thresholds, and techniques for measuring performance in supervised machine-learning tasks.

Chapter 3 outlines and presents the methods used during this research. We present how we established a framework for utilizing estimated class probabilities in order to determine class-specific thresholds, then outline our proposed techniques for utilizing those thresholds. We describe how we implemented GenThresh, OutBoost, and their unique variations, and how we evaluated the proposed techniques, the base learners we used, and the data sets that we

applied the techniques to.

We present the results of our evaluation in Chapter 4. We compare the results produced by GenThresh to the results produced by the base learners and discuss the comparison. Additionally, we compare the results of OutBoost to AdaBoost.M1 and its associated base learner. Finally, we explore some of the behavior demonstrated with OutBoost by examining the results produced by OutBoost.B1Man.

Our thesis is concluded with a description of our conclusions in Chapter 5, including possibilities for future research.

Chapter 2

Background

The following chapter describes the background of our thesis and introduces our reader to key concepts, relevant terminology, and algorithms used in our thesis. We begin by describing machine learning, classification, and outlier detection, then outline probability estimates, thresholds, and methods of measuring performance.

2.1 Data Mining

Data Mining is the process of automatically discovering unexpected, non-trivial, and useful knowledge in data [28]. This definition is partially shared with the field of *Knowledge Discovery in Databases* (KDD) where the end goal of KDD is the acquisition of useful information [8]. Data mining is an essen-

tial part of the KDD process, which includes data selection, preprocessing, transformation, data mining, interpretation and evaluation [8]. The process of data mining can include multiple components such as exploration, prediction, description, and visualization of data [28]. Data mining relies on the combination of multiple disciplines, including statistics, machine learning, programming, and domain knowledge [28]. Specifically, data mining relies on different variations of machine-learning techniques in order to both make predictions and describe the data under examination [8].

2.2 Machine Learning

Machine Learning may be described as the set of methods that can be utilized to automatically detect patterns in data [22]. Once detected, those patterns can be leveraged for decision making when dealing with uncertainty [22]. Machine learning-techniques learn through experience, and that experience takes the form of data [9]. Machine-learning techniques are categorized as either *predictive* or *descriptive* in nature and are also referred to as *supervised* for the former and *unsupervised* for the later [22]. The goal of predictive learning is to map a set of inputs to a set of outputs given a predefined set of input-output pairs [9, 22]. Examples of predictive learning include classification and regression, where classification involves predicating values that are categorical,

while regression is focused on predicting real-valued results [9, 22]. The goal of descriptive learning is to find interesting patterns in data given a set of inputs. An example of a descriptive learning approach would be clustering [9, 22].

2.3 Classification

Classification is a predictive machine-learning task where the formal goal is to learn a mapping from inputs $\mathbf{x}$ to outputs y, given a labeled set of input-output pairs defined as $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^{N}$, where $\mathcal{D}$ is the training set, N is the number of training examples, y_i is the class for $\mathbf{x}_i$, $y \in \{0, 1, 2, \ldots, C-1\}$, and C is the number of classes [22]. When $C = 2$, this scenario is referred to as *binary classification* [22]. When $C > 2$, scenario is referred to as *multiclass classification* [22].

The inputs ($\mathbf{x}$) may be described as a collection of *instances* (observations, records, points, vectors, etc.) where each instance is described by *attributes* (characteristics, features, fields, dimensions) [4]. Attributes may be composed of many different types of data (such as categorical or continuous) and the input may be described by a single attribute (*univariate*) or multiple attributes (*multivariate*) [4].

The problem of classification may be formalized as *function approximation*, where it is assumed that $y = f(\mathbf{x})$ for some unknown function (f) [22]. The

goal is to learn an estimate of f given a labeled training set and use $\hat{y} = \hat{f}(\mathbf{x})$ to make predictions for novel inputs [22]. Numerous machine-learning techniques exist that can be used to estimate this function [9, 22, 28]. Regardless of how classification is formally defined, the process of classification is that of modelling, where a model is the final result [9].

The primary goal of classification is to train models that generalize well when making predictions for previously unseen inputs [22]. This concept is referred to as *generalization*, where the goal is to generalize beyond the training set [22]. If the model does not generalize well to novel inputs, this problem may be one of *overfitting*, where minor variations of the input are modeled that do not lend improvements to the predictive power of the model [22].

In certain scenarios, a training set is provided where an imbalance exists between the number of examples of one class compared to the number of examples of another class [28]. In such a scenario, there is a *minority class* and a *majority class* [28]. The problem of classification in the face of an imbalanced class distribution in a training set is defined as *imbalanced classification* [28]. Creating models that generalize well using an imbalanced training set is a challenging problem and numerous methods of performance evaluation have been proposed for this scenario (further discussed in Section 2.7.4) [28]. Additionally, the process of training classification models using imbalanced training

sets is closely related to the problem of outlier detection [2].

2.3.1 Decision Trees

Decision Trees (DTs) are a recursive machine-learning technique that are best used to approximate discrete-valued target functions [9, 21]. DTs have been popular in the past due to their *Divide-and-Conquer* nature [9, 21]. DTs are particularly useful for learning scenarios where the attributes of the training set are composed of a small number of discrete values, but are also easily adapted to continuous attributes [21].

In general, a DT is composed of *nodes* and *branches* where the first node in the tree is referred to as the *root* [28]. Each node uses a splitting criteria for a specific attribute and each branch extending from a node is a specific value or value range for that attribute [21]. Instances are classified by sorting the instances down the tree, from root to leaf [21].

Beginning at the root node, DTs are constructed in a top-down fashion [21]. For each node, DTs perform tests on the set of instances associated with the node in order to determine which attribute should be used as a splitting criteria [21]. A DT creates a branch for each value, set of values, or value range, corresponding to the chosen splitting criteria, and descendant nodes are attached to each branch [21]. The training examples are then allocated to

the descendant nodes, depending on their attribute values [21]. This process is repeated for each descendant node until a leaf node has been found [21].

Once a DT has been built, instances that require classification start at the root node, where the DT tests the instance's attribute value for the splitting criteria of that node [21]. The observation then moves down the branch that corresponds to its value [21]. This process is than repeated for each subtree [21]. This process ends when an instance reaches a leaf node, at that point, the DT assigns the instance to the class associated with the leaf node [28].

2.3.2 Naive Bayes Classifier

Naive Bayes Classifiers (NBCs) are a probabilistic method of classification where predictions for an observation are made by assigning the observation the most probable class given the attribute-value combination of the observation [21]. In order to determine the most probable class for an observation, the class-conditional distribution must be determined, thus making this a generative approach [22]. With NBCs, determining the class-conditional distribution of the training set is simplified by assuming that the features are conditionally independent given the class label [22].

In practical terms, we do not expect the features to be independent, even conditional on the class labels, so we consider this model to be naive [22].

However, even in situations where the conditional-independence assumption does not hold it has been observed that a NBC will still perform well [6]. This is most likely due to the simplicity of the model [22].

The particular form of the class-conditional distribution varies depending on the type of features present in the training set [22]. The models themselves are generally fitted by computing the *Maximum a Posteriori* (MAP) Estimate, combined with a Bayesian approach to compute the full posterior [22].

2.3.3 Bayesian Belief Networks

Bayesian Belief Networks (BBNs) are a flexible alternative to NBCs, as they allow the conditional-independence assumption of NBCs to be broken, by modelling conditional probabilities between attributes [21, 28]. BBNs work by describing the probability distributions, which govern a set of variables, through combining a set of conditional-independence assumptions with a set of conditional probabilities [21]. This allows conditional-independence assumptions to be specified for a specific subset of variables [21].

NBCs graphically represent the probabilistic relationships among the set of random variables with a directed acyclic graph and a table of conditional probabilities [21, 28]. The graphical representation is in the form of nodes and arcs, where the nodes are the variables being modeled [21].

The directed arcs in the graph represent the dependence relationships between variables, and the assertion that a variable is conditionally independent of its non-descendants given its parent [21]. Conditional-probability tables exist for each variable and describe the probability distribution of the variable given the parent node, associating each node to its immediate parent [21]. The set of local conditional probabilities in combination with the conditional-independence assumptions describe the complete joint-probability distribution of the network [21].

There are numerous ways to learn the structure of the network (including manually creating it) and numerous ways to make inferences using the network once the network structure has been learned [21, 28].

2.3.4 Support Vector Machines

Support Vector Machines (SVMs) are a popular machine-learning technique, which determine the largest margin between parallel *hyperplanes*, in order to find a decision boundary [28]. A hyperplane is a higher-dimensional counterpart to a plane in three-dimensional space [23]. Predictions using a SVM depend on a subset of the training data, where the decision boundary is represented by support vectors [22, 27].

In their basic form, SVMs are not probabilistic in nature and do not result

in probabilistic outputs, however, they can be modified to produce probabilities through the application of a *Logistic Regression* (LR) to the outputs of the SVM [9, 22]. A LR operates by passing a linear combination of its inputs through a *sigmoid* function (otherwise known as a *Logistic* or *Logit* function), where a sigmoid function is a function that maps the whole real-number line to $[0, 1]$ [22].

2.3.5 Artificial Neural Networks

An *Artificial Neural Network* (ANN) is a machine-learning technique that can learn real and discrete-valued functions [21]. The design of ANNs was inspired by biological learning systems, which are built of dense collections of neurons. ANNs are constructed from densely connected simple units, each producing a single value [21]. In order to adjust the network to fit the training set, *Backpropagation* can be used to tune the network parameters [21].

The *Perceptron* is the most basic form of threshold unit in a neural network [21]. Multiple inputs are provided to a perceptron where the inputs are combined in a linear fashion before being passed through an activation function [21]. The activation function produces an output of either a 0 or 1 depending upon whether linearly combining the input values result in a value that is greater or equal to a certain threshold [21]. In essence, each input is adjusted

to have a certain weight in terms of the final output [21].

In order to represent a wide variety of nonlinear functions, the general approach is to combine a number of threshold units together to form a multilayer network. The activation function is replaced with a nonlinear function of its inputs, which produces a continuous output [21]. A multilayer neural network is a combination of an input layer, intermediate layers (hidden layers), and an output layer [28]. Numerous network topologies and threshold units exist [21, 28].

Backpropagation learns the weights of a multilayer network and is a practical application of the chain rule for derivatives [21]. Backpropagation determines the gradient of an objective function with respect to the weights of the multilayer network [21]. Backpropagation is a two-step process where the inputs are propagated forward in the network, and the errors are propagated backwards [21].

2.3.6 Boosting

Boosting is a greedy method of combining numerous weak learners (or *base learners*) in order to produce a single ensemble classifier that theoretically exhibits high levels of accuracy providing that the weak learners exhibit performance that is better than the performance achieved by random guessing

[11, 12, 22]. A base learner is a learning method (e.g. DT, NBC, BBN, etc...) that is treated like a black box by the Boosting technique, where we are only concerned with the output produced by the base learner [26]. Boosting operates by adaptively modifying the distribution of the training examples used during classification, by adjusting the importance of instances that are difficult to classify. This allows the weak learners to place more emphasis on properly classifying difficult instances [28].

In order for Boosting to improve the performance of a weak learner, the training set must be modified in some way [26]. The key to Boosting is to force the base learner to use a new training set during each iteration in order to infer something new about the data [26]. Numerous iterations of Boosting methods exist, one of the commonly used Boosting methods is AdaBoost [11, 12, 26].

AdaBoost

The *AdaBoost* algorithm initially assigns each instance an equal weight of $\frac{1}{m}$, where m is the number of training instances [11, 12]. Next, the assigned weights are used as a sampling distribution to draw a new training set for classification [26]. The goal of AdaBoost is to use a base learner to determine a weak hypothesis that will minimalize the training error (see Section 2.7.3 for a definition of error) at each iteration [26].

At each iteration of AdaBoost, the error of the weak learner is utilized to increase the weights of misclassified instances while decreasing the weights of those instances that are easy to classify [26]. The combined weights for each iteration of the classifier are normalized to sum to 1, allowing for the weights to once again be used as a sampling distribution where the probability of selecting an instance at random is proportional to the weight of the instance [26]. In some cases, the weights assigned to the training set may be used directly by the base learner [26].

After multiple iterations, AdaBoost combines the different results of the base learners into a final classifier, using a weighted voting scheme where the weighted majority of the base learner's predicted class determines the final predicted class for each instance to be classified [26].

2.4 Outliers

Outliers are known by numerous names (anomalies, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants) [4]. The following section introduces some basics concepts regarding outliers.

2.4.1 Outlier Definition

The definition of an outlier varies depending upon the domain under study and the nature of the data under investigation [4]. However, in broad terms, an outlier is something that is considered surprising [3]. When speaking in terms of continuous data, this may be an unusually high or low value in comparison to other values under examination [3]. Although vague, this definition does illustrate the subjective and complicated nature of outliers [3]. It is usual to assume that the number of outliers does not outnumber the number of normal observations [3].

For the purposes of this thesis we define an outlier as a point or pattern found in data that does not conform to what is considered normal [4]. More formally, we can define an outlier as an observation that deviates so much from other observations that we find the observation suspicious [15]. The degree of deviation can be defined through the use of statistical techniques (parametric and non-parametric), machine-learning methods, or subjective observation [3, 4].

2.4.2 Applications Of Outlier Detection

Outlier detection is applicable to many different fields and is used in applications such as intrusion detection, fraud detection, medical and public health,

industrial damage, image processing, and sensor networks [4]. Some examples of the applications of outlier detection include:

- Detecting malicious activities on computer systems from both a host-based and network-based perspective [4].
- Identifying malicious and criminal activity for organizations such as banks, credit card companies, and insurance agencies [4].
- Medical diagnosis, such as electrocardiograms, and epidemiology [4].
- Industrial damage detection to detect faulty equipment or structural problems [4].
- Image processing such as motion detection [4].
- Detecting interesting events or faulty sensors in sensor networks [4].

2.4.3 Types of Outliers

Outliers may appear in data that is univariate or multivariate in nature [3]. In general, outliers can be broken down into three distinct types: *point*, *contextual*, and *collective* [4]. A point outlier is one where a specific instance is unusual when compared to the other observations under study [4]. An example of a point outlier might be an unusually high value in a collection of

measurements. Given a sample, there may be multiple observations that are considered point outliers; however, each point is considered an outlier in its own right rather than as a group [4]. A point outlier may be univariate or multivariate in nature [2].

A contextual outlier is an outlier where a specific observation is unusual when compared to the other observations under study, in relation to the specific context under which it is being examined [4]. Consider a sample of sensor readings of recorded daily temperatures in a specific region for an entire year [4]. A contextual outlier is an usual temperature reading given a specific month [4]. Contextual outliers are multivariate in nature as there must be a minimum of two attributes, a *contextual attribute* and a *behavioral attribute* [4]. The contextual attribute defines the context, which the observations are being examined under, and the behavior attribute defines the behavior of the observation under study [4]. In the case of the previous example, the contextual attribute is the month of the year and the behavioral attribute is the daily temperature [4]. Multiple contextual and behavioral attributes may exist for a given observation [4].

Collective outliers are groups of observations that are unusual compared to the other observations under study [4]. Collective outliers are considered anomalous when the observations are examined as a group, but not when ex-

amined on an individual basis [4]. They form a distinct pattern when observed in series, where this pattern is surprisingly distinct from the overall pattern exhibited by the entire series [4]. In the medical field, an example of a collective outlier is an unusually low value, for a long period of time, in the output of an electrocardiogram, where a low value on its own is not unusual [4].

2.4.4 Typical Response to Outliers

When an observation or group of observations are determined to be outliers, one of three typical responses can occur: *removal*, *accommodation*, or *explanation* [3]. Removal generally occurs when the source of the outliers is some sort of error, for example, if a value was entered incorrectly or a sensor malfunctioned in it's reading, resulting in spurious value [3]. Accommodation occurs when the goal is to work with the data in spite of the outliers [3]. Statistically robust methods of analyzing data focus more on accommodation rather than removal, where the techniques are robust to presence of the outliers [3]. An example of this would be using the median rather than the mean, as an estimate of central tendency [10]. Explanation occurs when we are more interested in the outlier itself and wish to know why it occurs [3].

2.4.5 Outliers in Data Mining

Within data mining, outliers are generally divided into two different categories: *noise* and *anomalies* [2]. Noise can be considered a form of weak outlier while an anomaly can be considered a strong outlier [2]. Anomaly detection, noise removal, and noise accommodation are treated as distinct and separate tasks within the data-mining community [4]. In such instances, noise is considered a phenomenon that is not interesting to the data analyst and may stymie the data analysis task [4]. The importance of anomaly detection cannot be understated as discovered anomalies can lead to significant and actionable knowledge discovery [4]. As an example, an anomalous traffic pattern on a computer network could reveal that a computer has been compromised and is transmitting sensitive information to an outside attacker [4].

2.4.6 Outliers in Univariate Statistics

Outliers and outlier detection have been studied in the statistical domain since as least as early as the late 19th century, where the studies focused primarily on univariate outlier detection [3, 7]. An outlier in univariate statistics is a point or group of points, which deviate so much from the other observations that we are suspicious regarding the generating mechanism of the observation [15]. The degree of the deviation, which defines an outlier, depends greatly

on the outlier detection technique and the underlying distribution of the data from which the sample containing the outlier was drawn [2–4].

2.4.7 Outliers in Multivariate Statistics

Outliers in multivariate statistics are more complicated to define and are not as well explored in statistical literature [3]. The underlying definition of an outlier holds, where an instance deviates so much from the other observations that we suspect the observation; however, this deviation is now from the perspective of an instance with multiple attributes [3]. The outlier itself may be apparent when examining a multivariate observation using its dimensions in combination, but may not be apparent when examining the observation in terms of its component parts [2, 3]. The task of detecting multivariate outliers increases in complexity as the number of dimensions increase [2].

2.4.8 Outlier Labels and Scores

When observations or groups of observations are examined, those observations that are considered outliers may be represented in one of two ways: labels or scores [4]. When labels are utilized, an observation or group of observations are labeled as either being an outlier or not [4]. When scores are used, an observation or group of observations are assigned a value that represents the

degree of anomalous character an observation demonstrates [4]. In the case of scores, either a threshold can be used to determine when a score indicates that an observation is or is not an outlier, or the top k number of observations can be identified as outliers [4].

2.4.9 Univariate Outlier Detection

Univariate outlier detection is generally the domain of statistics and has been extensively studied in terms of literature and research [3, 4]. Univariate outlier detection techniques can be broken down into parametric and non-parametric methods [4]. Parametric methods make some sort of assumption regarding the underlying distribution (Gaussian, Exponential, Beta, etc.) and usually take into account sample statistics or population parameters in order to determine if an observation is an outlier [3, 4]. Parametric outlier detection techniques may test for outliers individually or as a block [3]. If tested as a block, the number of suspected outliers may be included in the test [3]. Numerous tests have been developed, each making different assumptions regarding what defines an outlier, although many parametric outlier detection techniques developed are focused on observations that come from a normal distribution [2–4].

Non-parametric outlier detection techniques do not make any assumptions regarding the underlying distribution of the sample being tested [4]. An ex-

ample of a non-parametric outlier detection technique could include the use of a histogram, which is a non-parametric method of examining the distribution of a sample [4].

2.4.10 Walsh's Outlier Test

Walsh's Outlier Test is a non-parametric block outlier test that may be applied to samples that contain more than 60 observations [1, 29–31].

Let $X_1, X_2, X_3, ..., X_n$ be an ordered set of values. The lower r values can be considered outliers, with a significance level α, if the following inequality holds true [1, 29–31]

$$X_r - (1 + a)X_{r+1} + aX_k < 0 \tag{2.1}$$

Where

$$c = \left\lceil \sqrt{2n} \right\rceil$$

$$k = r + c$$

$$b^2 = \frac{1}{\alpha}$$

$$a = \frac{1 + b\sqrt{\frac{(c-b^2)}{c-1}}}{c - b^2 - 1}$$

and the significance level $\alpha = .10$ if $60 < n <= 220$ or $\alpha = .05$ if $220 < n$ [1, 29–31].

The r upper values can be considered outliers, with a significance level α, if the following inequality holds true [1, 29–31]

$$X_{n+1-r} - (1+a)X_{n-r} + aX_{n+1-k} > 0 \tag{2.2}$$

Both the r upper and lower values can be considered outliers, with a significance level α, if both Inequality 2.1 and Inequality 2.2 hold true [1, 29–31].

In essence, Walsh's Outlier Test compares the distance between specific observations in an ordered sample in order to determine if the uppermost value in the lower r values being tested (X_r) is far enough away from its neighboring observation (X_{r+1}) to be declared outlying, in comparison to the distance between the neighboring observation and another observation higher in the sample (X_k) [1, 29–31]. X_k is chosen in order to consider an observation that is c observations away from the r observations being tested, where c specifies an observation that is a fair distance away from the uppermost value in the r lower observations, but not too far away as the sample size (n) grows [1, 29–31].

The parameter b^2 is predetermined by the significance level required for

the number of observations examined, and a is a product of the pre-specified significance level and the sample size being examined, in order to allow the inequality to say with some significance that the r lower observations are outliers [1, 29–31]. As the sample size increases $a \to 1$, which means the distance between X_r and X_{r+1} must be greater in comparison to the distance between X_{r+1} and X_k. As the sample size decreases a increases in size, allowing for a smaller spread between the afore mentioned observations.

2.5 Probability Estimates

The probability distribution over possible classes, given the input vector $\mathbf{x}$ and training set $\mathcal{D}$ is denoted as $P(y|\mathbf{x}, \mathcal{D})$ and represents a vector of length C [9]. In a classification task, $P(y = 0|\mathbf{x}, \mathcal{D})$ is the estimated probability of Class 0 given an instance ($\mathbf{x}$) and a data set ($\mathcal{D}$) [9]. Probability estimates range in value from $[0, 1]$, and in a binary classification scenario $P(y = 0|\mathbf{x}, \mathcal{D})$ is equal to one minus the probability of an observation being of the other class $P(y = 1|\mathbf{x}, \mathcal{D}) = 1 - P(y = 0|\mathbf{x}, \mathcal{D})$, and vice versa [9]. The methods used to determine the probability estimate vary depending on the classification technique and its implementation.

In certain cases, a probability estimate may be a direct result of the classifier being utilized (in the case of NBCs or BBNs) or may be interpreted from

scores or ranks assigned to each observation [9]. In other cases, the classification technique may not naturally create useful probability estimates and a LR may need be applied to the output of the classifier (such as the case of a SVM) [22]. DTs tend to assign a raw training frequency to each leaf and may utilize these values as the probability estimate [22]. In the case of many ANNs, the classifier may naturally produce good probability estimates [22].

2.6 Thresholds

A *threshold* (τ) is a boundary (or value) on a real-number line where some decision is made when a given value (a) is smaller or equal to a certain threshold ($a \leq \tau$), or when that value is larger then a certain threshold ($a > \tau$). A threshold may be used to define whether an instance is an outlier or not, or may be used to define whether an instance is of one class or another.

2.7 Measuring Performance

The following section describes common performance measures and techniques employed for estimating the generalization performance of a model or method.

2.7.1 Test Set

Good generalization is an important goal for models, so a method of evaluating how well the models generalize is required [22]. The performance of the model could be computed using the training set but this would only indicate how well the model performs on the training set, not how well it generalizes to novel inputs [22]. Therefor, once a classification model ($\hat{f}$) has been learned from a training set ($\mathcal{D}$) a test set ($\mathcal{T}$) may be used to estimate the generalization performance of the model, where the test set and the training set should be disjoint sets [21]. The test set ($\mathcal{T}$) is defined as $\mathcal{T} = (\mathbf{x}_i, y_i)_{i=1}^{N}$, where N is the number of instances in the test set, y_i is the class for $\mathbf{x}_i$, $y \in \{0, 1, 2, \ldots, C-1\}$, and C is the number of classes [22]. It is assumed that the training set and the test set are both drawn from the same unknown distribution [21].

2.7.2 The Confusion Matrix

The performance of a learned model may be represented by a confusion matrix (or contingency table) (See Table 2.1) [9, 28]. Each row in a confusion matrix represents the actual class of the observations, while each column represents the predicted class [9, 28]. In a binary classification problem, it is common practice to denote the minority class as the positive class (+) and the majority class as the negative class (−) [28].

Table 2.1: Definition of a confusion matrix [28].

	Predicted (+)	Predicted (−)
Actual (+)	TP	FN
Actual (−)	FP	TN

In terms of the confusion matrix, *True Positives* (TP) are observations that belong to the positive class and are predicted to belong to the positive class [28]. *True Negatives* (TN) are observations that belong to the negative class and are predicted to belong to the negative class [28]. *False Negatives* (FN) are observations that belong to positive class but are falsely predicted to belong to the negative class [28]. *False Positives* (FP) are observations that belong to the negative class but are falsely predicted to belong to the positive class [28].

2.7.3 Accuracy and Error

Given a learned model ($\hat{f}$), the accuracy of a model can be defined as the fraction of instances in the test set ($\mathcal{T}$) that were properly classified, and may be denoted as

$$acc_{\mathcal{T}}(\hat{f}) \equiv \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{T}} \delta(f(\mathbf{x}), \hat{f}(\mathbf{x})) \tag{2.3}$$

such that

$$\delta(f(\mathbf{x}), \hat{f}(\mathbf{x})) = \begin{cases} 1 & \text{if } f(\mathbf{x}) = \hat{f}(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where N is the number of instances in the test set [21].

Accuracy may also be defined in terms of the confusion matrix and denoted as [28]

$$acc_{\mathcal{T}} = \frac{TP + TN}{TP + TN + FN + FP} = \frac{TP + TN}{N} \tag{2.5}$$

Given a learned model ($\hat{f}$), the error of a model is defined as the fraction of instances in the test set ($\mathcal{T}$) that were not properly classified, and may be defined as

$$err_{\mathcal{T}}(\hat{f}) \equiv \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{T}} \delta(f(\mathbf{x}), \hat{f}(\mathbf{x})) \tag{2.6}$$

such that

$$\delta(f(\mathbf{x}), \hat{f}(\mathbf{x})) = \begin{cases} 1 & \text{if } f(\mathbf{x}) \neq \hat{f}(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

where N is the number of instances in the test set [21].

Error may also be defined in terms of the confusion matrix and be denoted as [28]

$$err_{\mathcal{T}} = \frac{FP + FN}{TP + TN + FN + FP} = \frac{FP + FN}{N} \tag{2.8}$$

Additionally, error can be defined in terms of accuracy such that [9]

$$err_{\mathcal{T}} = 1 - acc_{\mathcal{T}} \tag{2.9}$$

Accuracy may also be interpreted as an estimate of the probability of a random observation ($\mathbf{x}$) being classified properly $P_{\mathbf{x}}(\hat{f}(\mathbf{x}) = f(\mathbf{x}))$ [9]. Conversely, error may be interpreted as an estimate of the probability that a random observation will be classified improperly $P_{\mathbf{x}}(\hat{f}(\mathbf{x}) \neq f(\mathbf{x}))$ [9].

2.7.4 Alternative Performance Measures

Other metrics may be more appropriate to evaluate the performance of a classifier in an imbalanced classification scenario [28]. Accuracy and error are not always appropriate evaluation measures when the class imbalance is large, as both measures will be biased towards proper classification of the majority class over the minority class [9]. To give an extreme example, consider a classification task for a sample where 98% of the instances belong to the majority class and 2% of the instances belong to the minority class. A learned model could classify every instance in the sample as the majority class and still have an accuracy of 98%, or conversely an error rate of 2%. As such, accuracy and error are not always proper performance measures if the minority class is of

interest [9].

In order to address the problem of majority bias in accuracy and error measures, additional evaluation measures such as *Precision* and *Recall* may be adopted [5, 16, 24].

2.7.5 Precision and Recall

Precision for the minority class is the fraction of the classified examples that are of the minority class [28],

$$p = \frac{TP}{TP + FP} \tag{2.10}$$

Recall for the minority class is the fraction of the minority class that we properly classified [28],

$$r = \frac{TP}{TP + FN} \tag{2.11}$$

Additionally, recall may be considered the probability of a predicting that a randomly selected observation of the positive class is the positive class [9]. Recall is also known in the literature as the *True Positive Rate (TPR)* [9].

2.7.6 Cross Validation

The generalization performance of a model can be approximated by applying it to a large independent test set ($\mathcal{T}$), which was not used during the training of the model [22]. However, in practice this test set may not be available [22]. Instead, the training set can be split into two disjoint sets of different sizes, where the first set is used for training the model and the second set is used to estimate the future performance [22]. The performance of each model can then be compared to determine which one performs better in terms of generalization performance [22].

In k-fold *Cross Validation* (CV), the training set of size m is split into k partitions (otherwise denoted as k-folds) where each fold is approximately of size $\frac{m}{k}$ [21, 28]. The process of training and testing is iterated k times and for each iteration, one of the k folds is held back in a round-robin fashion to be used as the test set, while the other $k-1$ folds are used for training [22]. Every instance in the original training set is used the same number of times for training and only once for testing [22, 28]. A proxy for the generalization performance can be determined by combining the results and averaging the error (or other performance measure) over the folds [28]. It is common to use k=5 or k=10, where the former is refereed to as 5-fold CV and the later is referred to as 10-fold CV [22, 28]. If the process of k-fold CV is repeated, and

the folds are re-sampled for each iteration, the results of the repeated k-fold CV can be averaged in order to produce a performance estimate [32].

Stratified k-fold CV is a slight modification to the above technique and operates identically to k-fold CV except for one difference [9]. In stratified k-fold CV each fold keeps the same class distribution as the class distribution in the overall data set [9]. This ensures that in the case of an imbalanced class distribution the minority and majority class will be represented in each fold with equal frequency as in the original data set [9].

2.8 Chapter Summary

In this chapter we introduced the reader to key concepts, relevant terminology, and algorithms used in our thesis. We specifically focused on data mining, machine learning, boosting, outlier detection, evaluation measures, and methods of evaluating machine learning techniques. The reader was also introduced to key concepts regarding probability estimates, which will play an important role in the proposed techniques discussed in the following chapter.

Chapter 3

Methods

This chapter describes the methodologies used in our thesis. We begin by outlining a framework that utilizes univariate outlier detection to find class-specific thresholds in probability estimates. We then illustrate the different scenarios that may arise once we have found the class-specific thresholds, propose two methods of using these class-specific thresholds for classification tasks, and discuss how we implemented and evaluated them.

In general terms, class-specific thresholds were selected by applying a univariate outlier detection technique to the ordered set of probability estimates determined for each class. We placed particular emphasis on finding the lower outliers present in the probability estimates. Two methods were proposed to utilize the thresholds, which we refer to as GenThresh and OutBoost.

3.1 Refined Goals

We fashioned our approach with restrictions in mind in order to limit the breadth of this exploration. Specifically, our methods were formulated to work with a binary classification scenario. As such, we chose data sets with two classes in order to evaluate our techniques.

We refined our third general objective (introduced in Section 1.3) to include the following goals for GenThresh:

1. To determine if GenThresh could improve the accuracy achieved by its base learner.

2. Failing this, to determine if GenThresh could exceed the precision or recall of its base learner for the minority class of imbalanced data sets.

Additionally, we refined our third general objective to include the following goals for OutBoost:

1. To determine if OutBoost could exceed the accuracy of AdaBoost.M1.

2. Failing this, to determine if OutBoost could exceed the precision or recall of AdaBoost.M1 for the minority class of imbalanced data sets.

3. Finally, to determine if OutBoost could exceed the performance of AdaBoost.M1 when AdaBoost.M1 failed to exceed the performance of its

base learner.

3.2 Thresholds

The following section defines class-specific thresholds and introduces the concept of a cutoff.

3.2.1 Finding Thresholds

S is the set of m training examples $(\mathbf{x_i}, y_1), \ldots, (\mathbf{x_m}, y_m)$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \{0, 1\}$. We define P_{ij} as the estimated probability ($Pr(y = j|\mathbf{x}_i, S)$) of class j given training example $\mathbf{x}_i$ and the training set S. A threshold (τ) is a point on a real number line that delineates normal training examples from outlying training examples. We specify a threshold for each class where τ_j is the class-specific threshold for Class j. The real number line contains values that range from 0 to 1, such that $0 \leq \tau_j \leq 1$. Any observation with an estimated probability ($Pr(y = j|\mathbf{x}_i, S)$) of belonging to class j that fell on or below τ_j ($P_{ij} \leq \tau_j$) was designated as an outlier for Class j, while any observation with an estimated probability that fell above τ_j ($P_{ij} > \tau_j$) was designated as normal for Class j.

E_j is the ordered set of probability estimates for the m training examples,

where each probability estimate is the estimated probability of class j given the training example. The process of determining the class-specific threshold (τ_j) for each Class j proceeded as follows. We applied Walsh's outlier test to the probability estimates for Class j (E_j) starting with the $\lfloor \frac{m}{2} \rfloor$ lower probability estimates, where the $\lfloor \frac{m}{2} \rfloor$ probability estimates are the lower half of the probability estimates. Next, we applied Walsh's outlier test to the $\lfloor \frac{m}{2} \rfloor - 1, \lfloor \frac{m}{2} \rfloor - 2, ..., 1$ lower probability estimates, decreasing the number of lower probability estimates tested until we found lower outliers or until all of the lower probability estimates had been tested. If outliers were discovered during this procedure and the i lower probability estimates were deemed outliers, then the value of the highest probability estimate of the i lower probability estimates became the threshold, such that $\tau_j = E_{ji}$, where E_{ji} was deemed an outlier.

3.2.2 Cutoffs

Given the above, it is possible to encounter a scenario where one could not detect τ_j but still wished to define τ_j. Additionally, one could want to pretrim the probability estimates provided to the outlier detection test in order to subjectively define outliers. We introduced the concept of the cutoff α in order to meet these requirements.

We define the cutoff α as a predefined parameter that pre-trims the probability estimates E_j for each Class j before applying a univariate outlier detection test to E_j. Unlike the thresholds defined in Section 3.2, we only specify a single cutoff (α) that we used for all classes. The default choice for the cutoff was 0, in order to use all of the probability estimates to find the class-specific thresholds.

E_s is a subset of E ($E_s \subset E$) where E_s is the set of probability estimates found in E that are greater or equal to the cutoff. Given the ordered set of probability estimates (E_{s_j}, where $\left\{E_{s_{j1}}, E_{s_{j2}}, \ldots, E_{s_j}\right\} \in E_s$), we define E_{s_j} as the ordered set of probability estimates for Class j where $E_{ji} \geq \alpha$. Given that $\forall E_{s_{ji}} \in E_{s_j} \quad \alpha \leq E_{s_{ji}} \leq 1$, it was observed that E_s contains the same values as E ($E_s = E$) when the pre-defined cutoff is the default value.

Given our introduction of α, we substituted E_s for E when searching for the class-specific thresholds. The cutoff was also used as the default threshold for each class if the threshold was not detected via a univariate outlier test. We labeled any training example with an estimated probability ($E_{s_{ji}}$) that was equal to the cutoff (α) an outlier, even if they were not detected as outliers by an outlier test. Given the choice of α, any probability estimate that was smaller than the cutoff ($E_{s_{ji}} < \alpha$) was considered an outlier or simply part of another distribution.

3.2.3 Threshold Scenarios

Once a threshold (τ_j) was found for each Class j, we defined a set of thresholds (τ) that resulted in training examples falling in different positions relative to each threshold, depending upon the estimated probability of class j given the training example. Given a training example ($\mathbf{x}_i$) with an estimated probability of belonging to class j given $\mathbf{x}_i$ (P_{ij}), three different situations could occur:

a. The training example may fall below a class-specific threshold ($P_{ij} < \tau_j$)

b. The training example may fall above a class-specific threshold ($P_{ij} > \tau_j$)

c. The training example may fall on a class-specific threshold ($P_{ij} = \tau_j$)

A training example that fell on or below a class-specific threshold (τ_j) was designated an outlier for Class j ($P_{ij} \leq \tau_j$), and a training example that fell above a class-specific threshold (τ_j) was designated as normal for Class j ($P_{ij} > \tau_j$). We find two class-specific thresholds (τ_0 and τ_1) given a binary classification scenario, where P_{i0} is the estimated probability of belonging to Class 0 given $\mathbf{x}_i$, and P_{i1} is the estimated probability of belonging to Class 1 given $\mathbf{x}_i$. This leaves us with four specific possibilities for $\mathbf{x}_i$ given the values of the probability estimates for $\mathbf{x}_i$ and the class-specific thresholds:

1. The training example may fall below two class-specific thresholds ($(P_{i0} \leq \tau_0) \wedge (P_{i1} \leq \tau_1)$)

2. The training example may fall above two class-specific thresholds ($(P_{i0} > \tau_0) \wedge (P_{i1} > \tau_1)$)

3. The training example may fall above the first class-specific threshold and below the second class-specific threshold ($(P_{i0} > \tau_0) \wedge (P_{i1} \leq \tau_1)$)

4. The training example may fall below the first class-specific threshold and above the second class-specific threshold ($(P_{i0} \leq \tau_0) \wedge (P_{i1} > \tau_1)$)

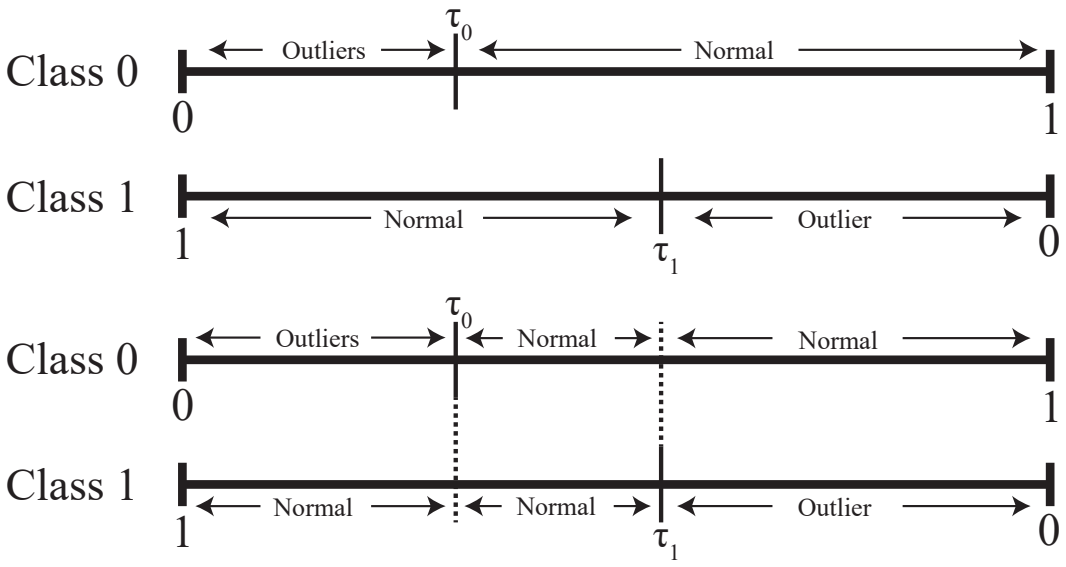
This was translated directly into outlier terms, where we detail each of the following mutually exclusive scenarios for binary classification:

1. A training example may be considered an outlier for both classes

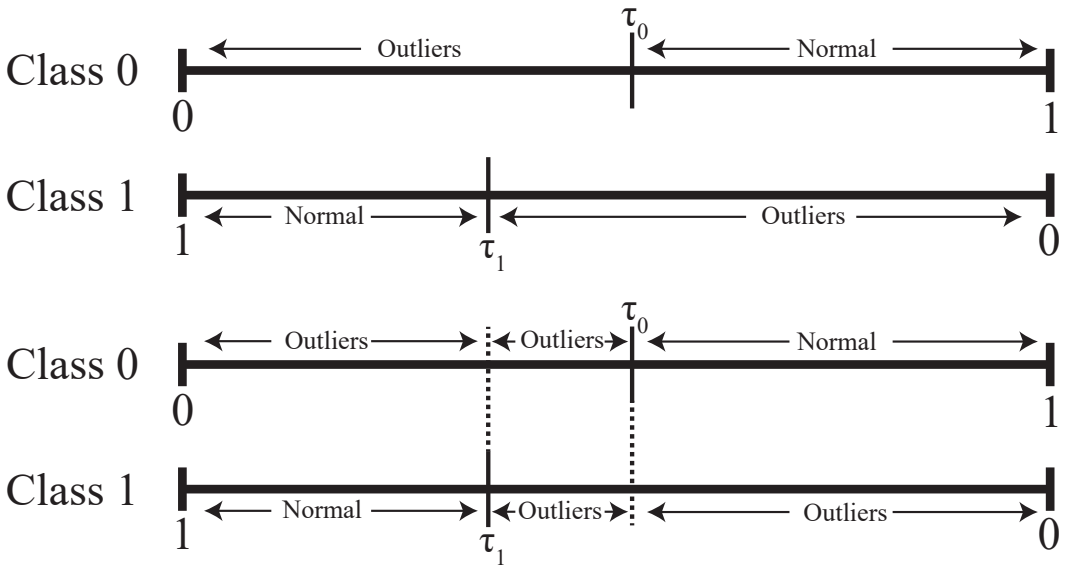
2. A training example may be considered normal for both classes

3. A training example may be considered normal for the one class but an outlier for the other

We illustrate an example of class-specific thresholds in Figure 3.1, as well as the different regions of probability estimates where the above three scenarios could occur relative to the class-specific thresholds. Figure 3.1 (a) demonstrates the relative positions of the class-specific thresholds (τ_0 and τ_1) required for a region of normal probability estimates to exist between the thresholds. Figure 3.1 (b) demonstrates a similar concept, showing the relative positions



(a) Region of Normal Probability Estimates between Class-Specific Thresholds



(b) Region of Outlying Probability Estimates between Class-Specific Thresholds

Figure 3.1: Examples of different class-specific thresholds and the different regions of probability estimates where Scenario 1, 2, and 3 may occur. (a) All of the training instances would be considered an outlier for both classes or considered an outlier for one class but normal for the other. (b) All of the training instances would be considered normal for both classes or considered an outlier for one class but normal for the other.

required for a region of outlying probability estimates to exist between the thresholds.

As $P(y = 0|\mathbf{x}, \mathcal{S}) = 1 - P(y = 1|\mathbf{x}, \mathcal{S})$, we visualize the potential values of the probability estimates for Class 0 in ascending order and Class 1 in descending order. Both (a) and (b) demonstrate the class-specific thresholds on their own and relative to the other class (indicated by the dashed line). To illustrate Scenario 3, consider Figure 3.1 (a) and a training example with a probability estimate for Class 0, which is smaller than τ_0, and a probability estimate for Class 1, which is larger than τ_1. The probability estimate for Class 1 must be larger than $1 - \tau_0$. Based on Figure 3.1 (a), this hypothetical training example would be an outlier for Class 0 and normal for Class 1, falling into Scenario 3.

3.2.4 Implementation

Our framework for discovering class-specific thresholds was implemented in Find-Thresholds (Algorithm 1) and Find-Threshold (Algorithm 2). We used the notation introduced in Section 3.2.1 and Section 3.2.2 and employed Walsh's Outlier Test to determine if a probability estimate and the probability estimates below it, were outliers.

Algorithm 1 Find-Thresholds operates by determining the class-specific thresholds (τ) given the estimated probabilities (P) of a training set. Find-Thresholds uses the probability estimates (E_j) to determine the class-specific threshold (τ_j) for each class (j), where C is the number of classes. Find-Thresholds sorts the probability estimates for Class j in ascending order and finds the class-specific threshold for Class j by calling Find-Threshold (Algorithm 2)

```
1: function FIND-THRESHOLDS(P, α)
2:     for j = 1 to C do
3:         for i = 1 to m do
4:             E_ji = P_ij
5:         end for
6:         SORT(E_j)
7:         τ_j = FIND-THRESHOLD(E_j, α)
8:     end for
9:     return τ
10: end function
```

3.3 GenThresh

In the following section, we propose and outline the GenThresh (short for general threshold) algorithm and its variations. We devised GenThresh to take advantage of the class-specific thresholds, cutoffs, and threshold scenarios which we defined in the previous sections.

GenThresh was fashioned to be part of a class of machine-learning techniques that assumes the availability of a base learner, where the goal is to improve the performance of the base learner. GenThresh treats the base learner much like a black box, where GenThresh ignores the internal workings of the base learner and only uses the results. In order for GenThresh to improve the

Algorithm 2 Find-Threshold operates by finding the class-specific threshold (τ) given a set of sorted probability estimates (P). Find-Threshold begins by ignoring any probability estimates that are less then the cutoff (α). Is-Outlier is a place holder for any outlier test. In this case, Is-Outlier is configured for a block outlier test that checks the j lower probability estimates to determine if they are outliers. The class-specific threshold is defined as the highest lower outlier found in the pre-trimmed probability estimates (E_s).

```
1: function FIND-THRESHOLD(P, α)
2:     for i = 1 to P.length do
3:         if P_i ≥ α then
4:             E_{s_i} = P_i
5:         end if
6:     end for
7:     for j = ⌊E_s.length / 2⌋ downto 1 do
8:         if IS-OUTLIER(E_s, j) then
9:             τ = E_{s_j}
10:            return τ
11:        end if
12:    end for
13:    τ = α
14:    return τ                          ▷ The threshold is the cutoff
15: end function
```

performance of the base learner, GenThresh has to use the results of the base learner and modify them in some way. In general terms, GenThresh utilizes the thresholds generated via the application of FindThresholds (Algorithm 1) and combines them in order to create a general threshold. GenThresh then shifts the probability estimates based on the general threshold, as outlined in the following sections.

We proposed two variations of the GenThresh technique, which we named GenThresh.B1 and GenThresh.B2. The "B" in the name indicates that the algorithm applied to binary classification only, while the "1" and "2" simply indicates the variations of similar methods.

3.3.1 GenThresh.B1

As previously mentioned, GenThresh.B1 (Algorithm 3) was based on a class of machine-learning techniques that assumes the availability of a base learner. We designed GenThresh.B1 to operate solely on a binary classification problem. Given a training set (S) composed of m examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in \mathcal{X}$ and $y_i \in \{0, 1\}$, GenThresh.B1 begins execution by applying a base learner to a training set (S) in order to determine a hypothesis, distinguished by h.

Once GenThresh.B1 has the hypothesis, it finds the estimated probability

Algorithm 3 GenThresh.B1 operates by training a base learner in order to determine the probability estimates for each class. GenThresh.B1 generates class-specific thresholds for each class and utilizes these thresholds to create a general threshold (τ_g), which is the average of $(1 - \tau_1)$ and τ_0. GenThresh.B1 finds the final classifier by determining the MAP estimate of the newly calibrated probability estimates, which is the most likely class given the input.

function GENTHRESH.B1
    TRAIN-LEARNER(S)
    Get hypothesis $h \colon \mathcal{X} \to \{0, 1\}$
    for $i = 1$ **to** m **do**
        $P_i = Pr(y|\mathbf{x}_i, S)$
    end for
    $\tau = \text{FIND-THRESHOLDS}(P)$
    $\tau_g = \frac{\tau_0 + (1-\tau_1)}{2}$
    $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \text{CALIBRATE}(Pr(y|\mathbf{x}, S, h), \tau_g, y)$
end function

(P_i) of each class (y) for each training example ($\mathbf{x}_i$). We define P_i as a vector of length C, where C is the number of classes. GenThresh.B1 finds the class-specific thresholds (τ_0 and τ_1) by applying Find-Thresholds (Algorithm 1) to the estimated probabilities (P). After which, GenThresh.B1 determines a general threshold (τ_g) (see Figure 3.2) and finds the final classifier by determining the MAP estimate of the newly calibrated probability estimates, which is simply the most likely class given the input.

The goal behind the development of GenThresh.B1 was to improve the performance of the base classifier by shifting the underlying estimated probability distribution for each class. We created Calibrate (Algorithm 4) with this goal in mind. Calibrate takes the general threshold (τ_g) and shifts the estimated probability of class y given observation $\mathbf{x}_i$, either to the left or the

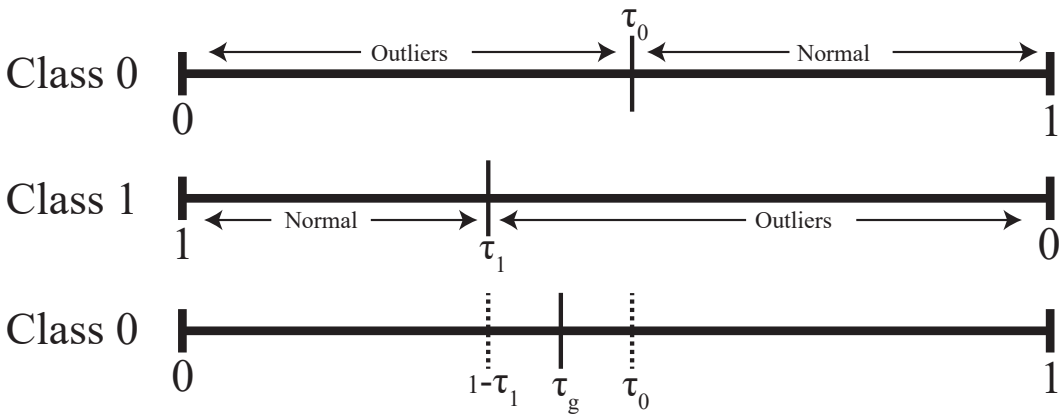


Figure 3.2: Visualization of how GenThresh.B1 (Algorithm 3) creates the general threshold (τ_g), using the class-specific threshold for Class 0 (τ_0) and Class 1 (τ_1), where the general threshold is the average of $(1 - \tau_1)$ and τ_0.

right, depending on the value of the general threshold. In essence, Calibrate uses τ_g as a new decision threshold for the probability estimates, but instead of changing the original decision threshold (τ_d) and modifying the MAP estimate to accommodate τ_g, Calibrate modifies the estimated probability landscape for each class.

Calibrate shifts the estimated probability Pr of class y given $\mathbf{x}_i$, based on the original class decision threshold (τ_d) and the new general threshold (τ_g). Although the MAP estimate does not explicitly use a decision threshold to choose the class of a specific observation, given a binary classification scenario there is an implicit decision threshold of $\tau_d = 0.5$. To demonstrate this, consider that $Pr(y = 0|\mathbf{x}, S) = 1 - Pr(y = 1|\mathbf{x}, S)$. If we found the MAP estimate for observation $\mathbf{x}$ with the $Pr(y = 0|\mathbf{x}, S) \geq .5$, then the MAP estimate would

Algorithm 4 Calibrate shifts the estimated probability (Pr) of class y given an observation, based on the original class decision threshold (τ_d) and the new general threshold (τ_g.)

```
1: function CALIBRATE(Pr, τ_g, y)
2:     if y = 0 then
3:         Pr = Pr − (τ_g − τ_d)
4:     else
5:         Pr = Pr − ((1 − τ_g) − τ_d)
6:     end if
7:     if Pr < 0 then
8:         Pr = 0
9:     end if
10:    if Pr > 1 then
11:        Pr = 1
12:    end if
13:    return Pr
14: end function
```

label observation $\mathbf{x}$ as Class 0.

Calibrate operates differently depending upon which class the probability estimate belongs. The new probability estimate (Pr) is the difference between the original probability estimate and the difference between the general threshold (τ_g) and the original decision threshold (τ_d), if the probability estimate is for Class 0. Calibrate shifts the probability estimate to the right if $\tau_g < \tau_d$ or shifts the probability estimate to the left if $\tau_g > \tau_d$. The new probability estimate (Pr) is the difference between the original probability estimate and the difference between $1 - \tau_g$ and the original decision threshold (τ_d), if the probability estimate is for Class 1. Calibrate shifts the probability estimate to the right if $(1 - \tau_g) < \tau_d$ or shifts the probability estimate to the left if $(1 - \tau_g) > \tau_d$.

The goal behind this shift was to modify the performance of a base learner by shifting potentially misclassified training examples, which may be close to the original decision threshold, across the original decision threshold. Essentially, GenThresh.B1 classifies any observations as Class 0 if its original probability estimate for Class 0 fell to the right of τ_g and classifies any observations as Class 1 if its original probability estimate for Class 0 fell to the left of τ_g.

3.3.2 GenThresh.B2

We constructed GenThresh.B2 (Algorithm 5) to operate almost the same as GenThresh.B1. The only difference between GenThresh.B1 and GenThresh.B2 is how the general threshold (τ_g) is determined. Instead of taking the average of the two class-specific thresholds (τ_0 and $(1 - \tau_1)$), GenThresh.B2 computes the average probability estimate (in terms of Class 0) of the training examples that fall between τ_0 and $(1 - \tau_1)$ (Algorithm 6).

We consider two different cases, depending upon the value of the thresholds (τ). If outlying observations may fall between the thresholds, then the general threshold (τ_g) is the average of the estimated probabilities of the outlying observations on and between the thresholds. If normal observations may fall between the thresholds then the general threshold (τ_g) is the average of the

Algorithm 5 GenThresh.B2 operates by training a base learner in order to determine the probability estimates for each class. GenThresh.B2 generates class-specific thresholds for each class and utilizes these thresholds to create a general threshold (τ_g), which is the average probability estimate (in terms of Class 0) of the training examples that fall between τ_0 and $(1 - \tau_1)$. GenThresh.B2 finds the final classifier by determining the MAP estimate of the newly calibrated probability estimates, which is simply the most likely class given the input.

function GenThresh.B2
    Train-Learner(S)
    Get hypothesis h: $\mathcal{X} \rightarrow \{0, 1\}$
    for $i = 1$ **to** m **do**
        $P_i = Pr(y|\mathbf{x}_i, S, h)$
    end for
    τ = Find-Thresholds(P)
    τ_g = AvgPrBetweenThresholds(P, τ)
    $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}}$Callibrate($Pr(y|\mathbf{x}, S, h), \tau_g$)
end function

Algorithm 6 AvgPrBetweenThresholds creates a general threshold (τ_g) by finding the average of the probability estimates that fall between τ_0 and $(1-\tau_1)$.

function AvgPrBetweenThresholds(P, τ)
    for $i = 1$ **to** m **do**
        if $\tau_0 - (1 - \tau_1) \geq 0$ **then** ▷ outlier between thresholds
            if $P_i(y = 0) \leq \tau_0$ and $P_i(y = 0) \geq (1 - \tau_1)$ **then**
                $sum+ = P_i(y = 0)$
                $count++$
            end if
        else ▷ normal between thresholds
            if $P_i(y = 0) > \tau_0$ and $P_i(y = 0) < (1 - \tau_1)$ **then**
                $sum+ = P_i(y = 0)$
                $count++$
            end if
        end if
    end for
    if $count \neq 0$ **then**
        $\tau_g = \frac{sum}{count}$
    else
        $\tau_g = \frac{\tau_0 + (1-\tau_1)}{2}$
    end if
    return τ_g
end function

estimated probabilities of the normal observations falling between the thresholds. Essentially, we were only interested in the estimated probabilities of those training examples that fell into Scenario 1 or 2 (See Section 3.2.3).

3.4 OutBoost

In the following section, we propose and outline OutBoost (short for Outlier Boosting), which is based heavily on the AdaBoost.M1 algorithm. We created OutBoost by modifying AdaBoost.M1[1] to take advantage of the thresholds, cutoffs, and threshold scenarios outlined previously, in order to increase the weight of those observations that were misclassified and fell into the different threshold scenarios.

Like GenThresh, OutBoost was created to be part of a class of machine-learning techniques that assumes the availability of a base learner. This class of machine-learning technique treats the base learner like a black box that can be called numerous times in order to classify the training examples [26]. The goal is to improve the performance of the base learner over the numerous

[1] One of the primary benefits of Boosting is the theoretical guarantee that it will improve the performance of a weak learner for any weak learner that arbitrarily exceeds 50% in accuracy [25]. These theoretical guarantees may not extend to the proposed OutBoost techniques, as it is possible that the sampling distribution of the training examples may stay the same, depending upon the value of the cutoff, the class-specific thresholds, and the probability estimates generated by the base learner.

Table 3.1: Comparison of OutBoost Techniques

OutBoost Method	Thresholds	Details
OutBoost.B1	Automatic	Increases the weight of those misclassified training instances that are outliers for one class
OutBoost.B1Man	Manual	Increases the weight of those misclassified training instances that are outliers for one class
OutBoost.B2	Automatic	Increases the weight of those misclassified training instances that are normal for both classes or outliers for both classes

iterations, and combine the results of the numerous base learners in order to produce a single classifier with improved performance [26]. In order to improve the performance of the base learner over each iteration, the machine-learning technique must force the base learner to produce new results that are different than the previous results [26]. In general terms, OutBoost utilizes the threshold scenarios illustrated in Section 3.2.3 in order to place more importance on those training examples that are improperly classified during each iteration, conditional on where they fall relative to the different threshold scenarios, in terms of their estimated probabilities.

We proposed three variations of the OutBoost technique that we denominated OutBoost.B1Man, OutBoost.B1, and OutBoost.B2. The "B" in the name again indicates that the algorithm applies to binary classification only, while the "1" and "2" simply indicate the variations on similar methods. "Man" implies that we manually specify the class-specific thresholds. Table 3.1 describes the differences between the three techniques.

3.4.1 AdaBoost.M1

We begin by describing AdaBoost.M1 in further detail in order to properly discuss the different variations of OutBoost.

Algorithm 7 AdaBoost.M1 operates by training a base learner, where the goal of the base learner is to minimalize the weighted error [12]. AdaBoost.M1 decreases the weight of properly classified observations in order to focus the base leaner on the misclassified observations during the next iteration of AdaBoost.M1 [12]. The weights of all the observations are normalized to sum to one and used as a sampling distribution for the next iteration of the base learner [12]. AdaBoost.M1 combines the results of each iteration with a weighted voting scheme in order to build a final classifier [12].

1: **function** AdaBoost.M1
2: **for** $i = 1$ **to** m **do**
3: $D_1(i) = \frac{1}{m}$
4: **end for**
5: **for** $t = 1$ **to** T **do**
6: TrainWeakLearner(D_t)
7: Get weak hypothesis $h_t \colon \mathcal{X} \to \mathcal{Y}$
8: $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq \mathbf{y}_i} D_t(i)$
9: **if** $\epsilon_t \geq \frac{1}{2}$ **then**
10: $T = t - 1$
11: **Exit Loop**
12: **end if**
13: $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
14: **for** $i = 1$ **to** m **do**
15: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) = \mathbf{y}_i \\ 1 & \text{otherwise} \end{cases}$
16: **end for**
17: **end for**
18: $H(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = \mathbf{y}} log\frac{1}{\beta_t}$
19: **end function**

AdaBoost.M1 (Algorithm 7) operates in a multiclass classification scenario, hence the M [12, 26]. Given a training set (S) of m examples $(x_1, y_1), \ldots, (x_m, y_m)$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, the distribution D_t is the distribution over the training

set (S) for iteration t and $D_t(i)$ is the weight assigned to example $\mathbf{x}_i$ [12]. AdaBoost.M1 assigns each example an initial weight ($D_1(i)$) of $\frac{1}{m}$ before the first iteration. For each iteration (t from $1 \ldots T$) AdaBoost.M1 trains a base learner using the training set (S) given the distribution D_t and obtains hypothesis h_t via the base learner [12]. The goal of the base learner is to find a hypothesis that minimalizes the training error (ϵ_t), which is the error of h_t given the distribution D_t [12].

AdaBoost.M1 calculates β_t, where $\beta_t \in [0, 1)$ and uses β_t as a weighting factor for each example ($\mathbf{x}_i$) that is classified properly by h_t ($h_t(\mathbf{x}_i) = y_i$) [12]. Next, AdaBoost.M1 determines the weight of each example for the following iteration ($D_{t+1}(i)$) by multiplying the current weight of the training example ($D_t(i)$) by β_t if the training example was classified properly ($h_t(\mathbf{x}_i) = y_i$) [12]. Z_t is the normalization factor for iteration t so that D_{t+1} may be used as a distribution over S [12]. This process repeats for T iterations, or until $\epsilon_t \geq .5$ [26].

In essence, those examples that are properly classified are assigned a lower weight, while those observations that are harder to classify are assigned a higher weight [12]. This enables the base learner to focus on those examples that are hard to classify [12]. A final classifier is then formed as a weighted vote of the base hypotheses [12].

WEKA's implementation of AdaBoost.M1 (Algorithm 8) is a slight modification of the Adaboost.M1 algorithm [14]. It differs in assigning β_t a value where $\beta_t \in (1, 100)$ and determines the weight of each example for the next iteration $D_{t+1}(i)$ by multiplying the current weight of the training example $D_t(i)$ by β_t if the training example was incorrectly classified [14]. WEKA's implementation is also modified so that the weighted vote uses $\log \beta_t$ as the weight for each base learner instead of $\log \frac{1}{\beta_t}$ [14]. We modified WEKA's implementation of AdaBoost.M1 in order to create OutBoost.

Algorithm 8 AdaBoost.M1 (WEKA Implementation) operates almost identically to AdaBoost.M1 but instead increases the weight of misclassified observations. WEKA's implementation also modifies how β_t is calculated and used for reweighing observations, and how β_t is used in the final vote. [12, 14].

function AdaBoost.M1
  for $i = 1$ **to** m **do**
    $D_1(i) = \frac{1}{m}$
  end for
  for $t = 1$ **to** T **do**
    TrainWeakLearner(D_t)
    Get weak hypothesis $h_t \colon \mathcal{X} \to \mathcal{Y}$
    $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq \mathbf{y}_i} D_t(i)$
    if $\epsilon_t \geq \frac{1}{2}$ **then**
      $T = t - 1$
      Exit Loop
    end if
    $\beta_t = \frac{1-\epsilon_t}{\epsilon_t}$
    for $i = 1$ **to** m **do**
      $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) \neq y_i \\ 1 & \text{otherwise} \end{cases}$
    end for
  end for
  $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = y} log \beta_t$
end function

3.4.2 OutBoost.B1Man

Although we developed OutBoost.B1Man (Algorithm 9) as a final OutBoost variation, we describe it here first given its description helps form the basic building blocks for the remaining methods.

OutBoost.B1Man is very similar to AdaBoost.M1 (Algorithm 8), however, there are a few key differences. OutBoost.B1Man operates on data sets possessing only two classes while AdaBoost.M1 operates on data sets with multiple classes. Additionally, OutBoost.B1Man uses class-specific thresholds τ, where τ_0 is the class-specific threshold for Class 0 and τ_1 is the class-specific thresholds for Class 1. Finally, OutBoost.B1Man was created so that we could manually specify the class-specific thresholds, which then remains the same for each iteration.

As with AdaBoost.M1, OutBoost.B1Man trains the base learner using the distribution D_t over the training examples S, determines the base hypothesis h_t, and calculates the error (ϵ_t) and β_t. OutBoost.B1Man differs by checking if outliers or normal training examples could fall between the class-specific thresholds before OutBoost.B1Man updates the weights of the training examples, creating distribution D_{t+1}.

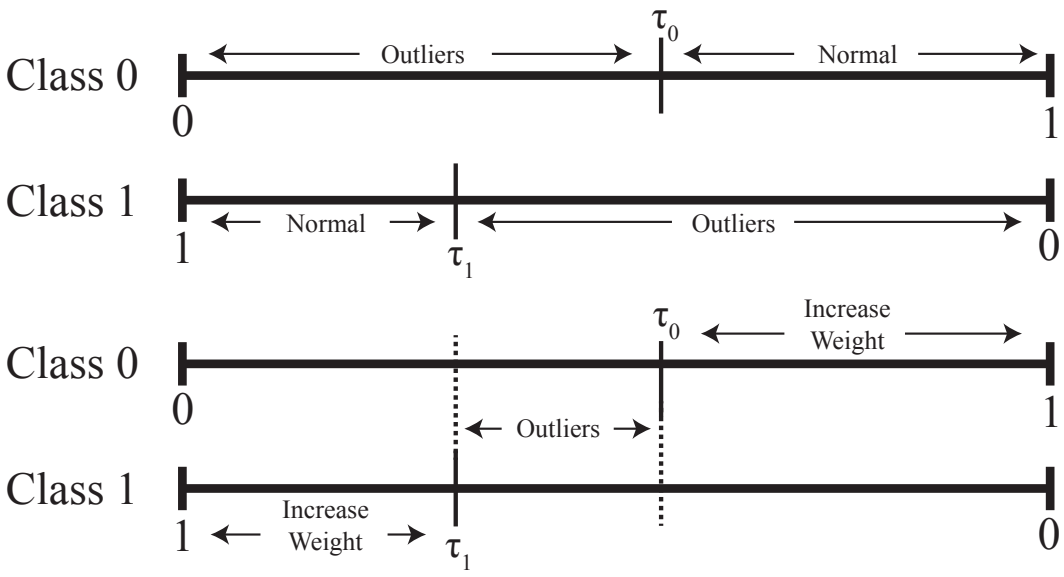
If OutBoost.B1Man discovers that there may be normal training examples between both class-specific thresholds (τ) we know that Scenario 1 or 3 (see

Algorithm 9 OutBoost.B1Man is a modification of AdaBoost.M1 that operates on data sets with two classes. OutBoost.B1 operates by training a base learner, where the goal of the base learner is to minimalize the weighted error. OutBoost.B1Man enables the user to manually specify class-specific thresholds. OutBoost.B1Man increases the weight of an observation that is misclassified and considered an outlier for one class and normal for the other class given the class-specific thresholds. The weights of all the observations are normalized to sum to one and used as a sampling distribution for the next iteration of the base learner. OutBoost.B1Man combines the results of each iteration with a weighted voting scheme in order to build a final classifier.

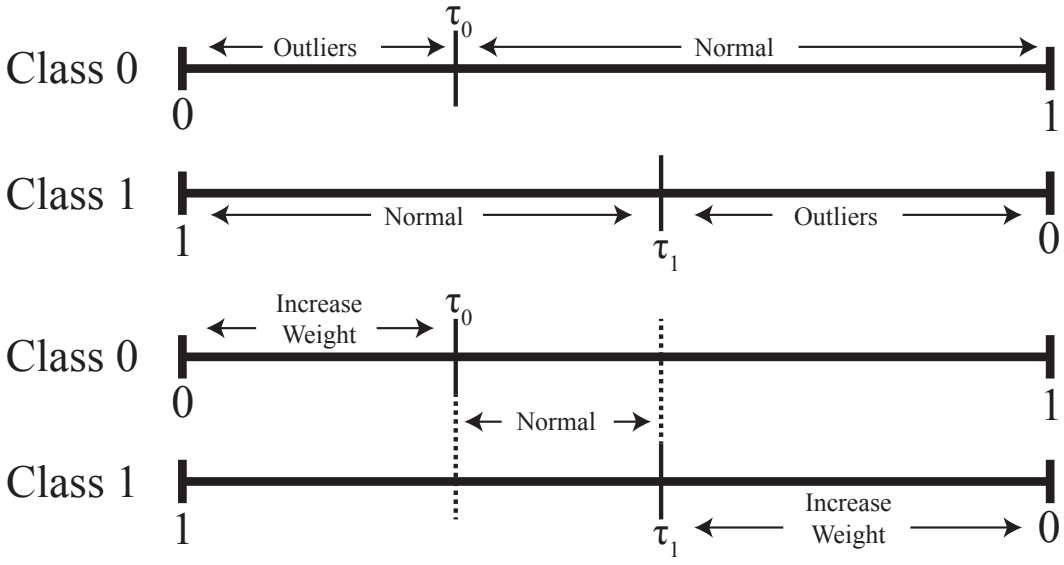
function OUTBOOST.B1MAN
  for $i = 1$ **to** n **do**
    $D_1(i) = \frac{1}{n}$
  end for
  for $t = 1$ **to** T **do**
    TRAINBASELEARNER(D_t)
    Get base hypothesis $h_t \colon \mathcal{X} \to \{0, 1\}$
    $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq y_i} D_t(i)$
    if $\epsilon_t \geq \frac{1}{2}$ **then** ▷ error is too large
      $T = t - 1$
      Exit Loop
    end if
    $\beta_t = \frac{1-\epsilon_t}{\epsilon_t}$
    for $i = 1$ **to** n **do**
      if $\tau_0 - (1 - \tau_1) \geq 0$ **then** ▷ outlier between thresholds
        $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) \neq y_i \text{ and} \\ & ((Pr(y = \hat{y}_i | \mathbf{x}_i, D_t) > \tau_{\hat{y}_i}) \\ 1 & \text{otherwise} \end{cases}$$
      else ▷ normal between thresholds
        $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } (h_t(\mathbf{x}_i) \neq y_i) \text{ and} \\ & ((Pr(y = y_i | \mathbf{x}_i, D_t)) \leq \tau_{y_i} \\ 1 & \text{otherwise} \end{cases}$$
      end if
    end for
  end for
  $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = y} log \beta_t$
end function

Section 3.2.3) is in effect for all training examples. OutBoost.B1Man is not concerned with those training examples that are considered normal for both classes. If OutBoost.B1Man finds that there may be outlying training examples between the thresholds, this indicates that Scenario 1 or 3 (see Section 3.2.3) is in effect for all training examples. Similarly, OutBoostM1.Man is not concerned with those training examples that are considered outliers by both classes. OutBoostM1.Man was tailored to only increase the weights of those observations that are affected by Scenario 3. In other words, it increases the weights of observations that are misclassified and an outlier for one class but not the other.

If OutBoost.B1Man finds that normal training examples may be present between the thresholds, OutBoost.B1Man determines the weight of each example for the next iteration ($D_{t+1}(i)$) by multiplying the current weight of the training example ($D_t(i)$) by β_t if the training example was misclassified ($h_t(\mathbf{x}_i) \neq y_i$) and if the estimated probability of being the true class is lower than or equal to the class threshold of the true class ($Pr(y_i|\mathbf{x}_i, D_t) \leq \tau_{y_i}$). Like AdaBoost.M1, OutBoost.B1 divides the weights by Z_t, which is the normalization factor for iteration t, so that D_{t+1} may be used as a distribution over S. In effect, OutBoost.B1Man increases the weights of those observations that are misclassified and considered an outlier for one class but not the other, and



(a) Region of Outlying Probability Estimates between Class-Specific Thresholds



(b) Region of Normal Probability Estimates between Class-Specific Thresholds

Figure 3.3: Visualizes the regions of probability estimates where OutBoost.B1Man and OutBoost.B1 will increase the weight of a misclassified training example, given the class-specific thresholds and the probability estimate produced by the base learner for the training instance.

decreases the weights of the other observations.

As a final case, if OutBoost.B1Man discovers that outlying training examples may be present between the thresholds, OutBoost.B1Man instead multiplies the current weight of the training example ($D_t(i)$) by β_t if the training example was misclassified ($h_t(\mathbf{x}_i) \neq y_i$) and if the estimated probability of being the estimated class is greater than the class threshold of the estimated class ($Pr(\hat{y}_i|\mathbf{x}_i, D_t) > \tau_{\hat{y}_i}$). Again, OutBoost.B1Man increases the weights of those observations that are misclassified and considered an outlier for one class but not the other and decreases the weights of the other observations.

We illustrate both of these cases in Figure 3.3 by visualizing the regions where a training example must fall relative to the class-specific thresholds in terms of its probability estimates, in order for the weight of the observation to be increased if it is misclassified. Where (a) demonstrates the first case, where observations may be present that are considered outliers by both classes, and (b) shows the second case, where observations may be present that are considered normal by both classes.

In essence, OutBoostM1.Man forces the base learner to focus on those training examples that are misclassified and have probabilities estimates that are considered outliers for one class but normal for the other. In other words, OutBoostM1.Man forces the base learner to place more importance on those

examples that are especially difficult to classify, similar to AdaBoost.M1. As with AdaBoost.M1 this process is repeated for T iterations, or until $\epsilon_t \geq 0.5$. After which, a final classifier is formed as a weighted vote of the base hypotheses.

3.4.3 OutBoost.B1

OutBoost.B1 (Algorithm 10) operates almost identically to OutBoost.B1Man except for one key difference: instead of manually specifying the thresholds (τ) before execution, OutBoost.B1 determines class-specific thresholds by applying a univariate outlier detection technique to the probability estimates generated by the base learner. OutBoost.B1 finds the estimated probabilities for each training example (P_i) and automatically generates a class-specific threshold for each class during each iteration by applying Find-Thresholds (Algorithm 1) to the estimated probabilities (P) of the training examples for iteration t.

3.4.4 OutBoost.B2

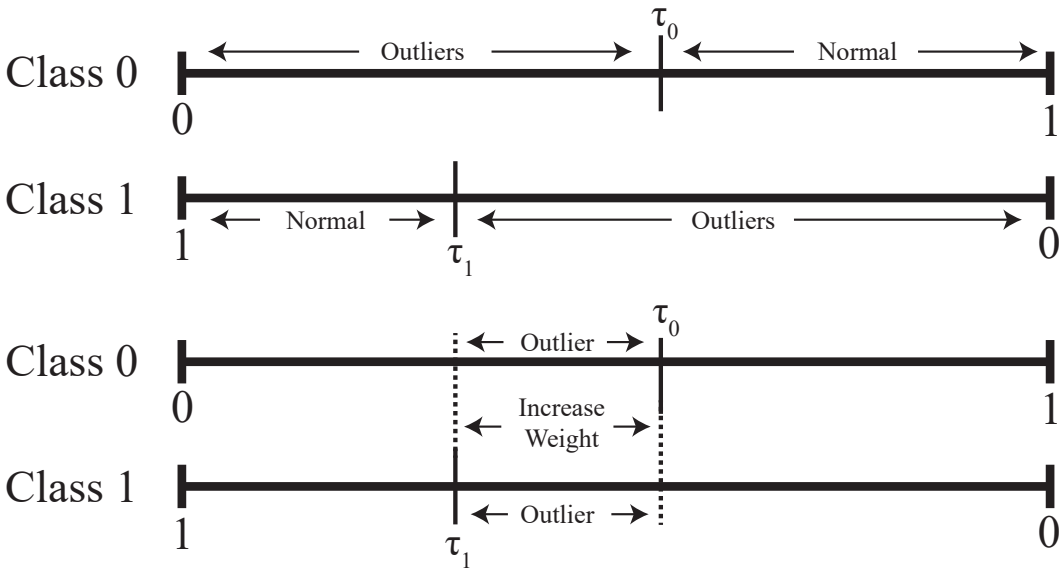
OutBoost.B2 (Algorithm 11) is similar to OutBoost.B1 except for another key difference: OutBoost.B2 increases the weight of those observations that are considered normal or outliers by both classes instead of increasing the weight of those observations that are misclassified and considered outliers for

Algorithm 10 OutBoost.B1 is a modification of AdaBoost.M1 that requires two classes. OutBoost.B1 operates by training a base learner, where the goal of the base learner is to minimalize the weighted error. OutBoost.B1 determines class-specific thresholds by applying a univariate outlier detection technique to the probability estimates generated by the base learner. OutBoost.B1 increases the weight of an observation that is misclassified and considered an outlier for one class and normal for the other class, given the class-specific thresholds. The weights of all the observations are normalized to sum to one and used as a sampling distribution for the next iteration of the base learner. OutBoost.B1 combines the results of each iteration with a weighted voting scheme in order to build a final classifier.

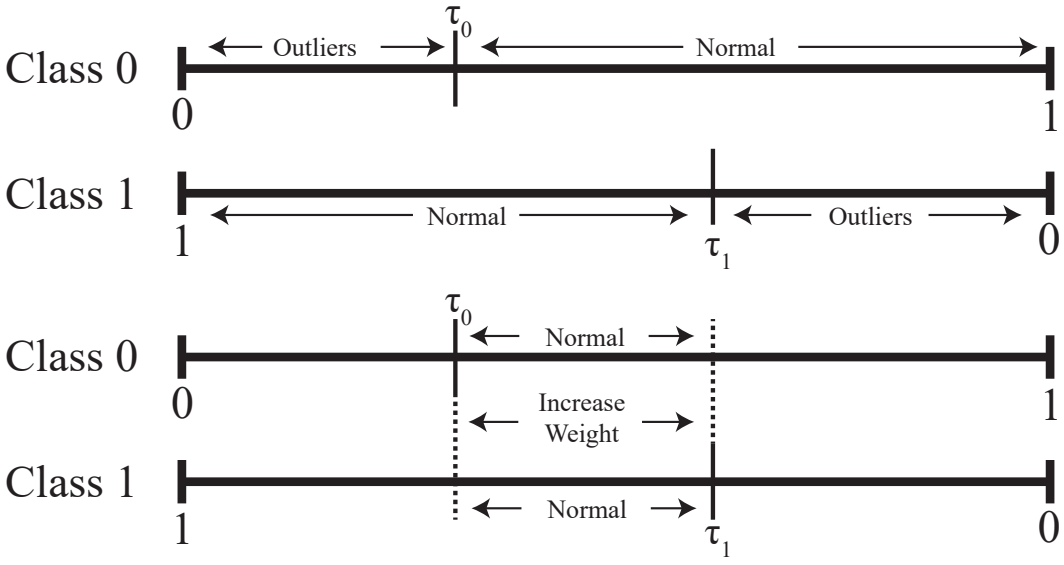
function OUTBOOST.B1
  for $i = 1$ **to** n **do**
    $D_1(i) = \frac{1}{n}$
  end for
  for $t = 1$ **to** T **do**
    TRAINBASELEARNER(D_t)
    Get base hypothesis $h_t \colon \mathcal{X} \to \{0, 1\}$
    $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq y_i} D_t(i)$
    if $\epsilon_t \geq \frac{1}{2}$ **then** ▷ error is too large
      $T = t - 1$
      Exit Loop
    end if
    $\beta_t = \frac{1-\epsilon_t}{\epsilon_t}$
    for $i = 1$ **to** n **do**
      $P_i = Pr(y|\mathbf{x}_i, D_t)$
    end for
    $\tau = \text{FINDTHRESHOLDS}(P)$
    for $i = 1$ **to** n **do**
      if $\tau_0 - (1 - \tau_1) \geq 0$ **then** ▷ outlier between thresholds
        $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) \neq y_i \text{ and} \\ & ((Pr(y = \hat{y}_i|\mathbf{x}_i, D_t) > \tau_{\hat{y}_i}) \\ 1 & \text{otherwise} \end{cases}$$
      else ▷ normal between thresholds
        $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } (h_t(\mathbf{x}_i) \neq y_i) \text{ and} \\ & ((Pr(y = y_i|\mathbf{x}_i, D_t) \leq \tau_{y_i} \\ 1 & \text{otherwise} \end{cases}$$
      end if
    end for
  end for
  $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = y} log\beta_t$
end function

Algorithm 11 OutBoost.B2 is a modification of AdaBoost.M1 and OutBoost.B1 [12]. OutBoost.B2 operates by training a base learner, where the goal of the base learner is to minimalize the weighted error. OutBoost.B2 determines class-specific thresholds by applying a univariate outlier detection technique to the probability estimates generated by the base learner. OutBoost.B2 increases the weight of an observation that is misclassified and considered an outlier or normal by both classes, given the class-specific thresholds. The weights of all the observations are normalized to sum to one and used as a sampling distribution for the next iteration of the base learner. OutBoost.B2 combines the results of each iteration with a weighted voting scheme in order to build a final classifier.

1: **function** OUTBOOST.B2
2: **for** $i = 1$ **to** n **do**
3: $D_1(i) = \frac{1}{n}$
4: **end for**
5: **for** $t = 1$ **to** T **do**
6: TRAINBASELEARNER(D_t)
7: Get base hypothesis $h_t \colon \mathcal{X} \to \{0, 1\}$
8: $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq y_i} D_t(i)$
9: **if** $\epsilon_t \geq \frac{1}{2}$ **then** ▷ error is too large
10: $T = t - 1$
11: **Exit Loop**
12: **end if**
13: $\beta_t = \frac{1-\epsilon_t}{\epsilon_t}$
14: **for** $i = 1$ **to** n **do**
15: $P_i = Pr(y|\mathbf{x}_i, D_t)$
16: **end for**
17: $\tau = \text{FINDTHRESHOLDS}(P)$
18: **for** $i = 1$ **to** n **do**
19: **if** $\tau_0 - (1 - \tau_1) \geq 0$ **then** ▷ outlier between thresholds
20: $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } (h_t(\mathbf{x}_i) \neq y_i) \text{ and} \\ & ((Pr(y = y_i|\mathbf{x}_i, D_t)) \leq \tau_{y_i} \\ 1 & \text{otherwise} \end{cases}$$
21: **else** ▷ normal between thresholds
22: $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) \neq y_i \text{ and} \\ & ((Pr(y = \hat{y}_i|\mathbf{x}_i, D_t) > \tau_{\hat{y}_i}) \\ 1 & \text{otherwise} \end{cases}$$
23: **end if**
24: **end for**
25: **end for**
26: $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = y} log\beta_t$
27: **end function**



(a) Region of Outlying Probability Estimates between Class-Specific Thresholds



(b) Region of Normal Probability Estimates between Class-Specific Thresholds

Figure 3.4: Visualizes the regions of probability estimates where OutBoost.B2 will increase the weight of a misclassified training example, given the class-specific thresholds and the probability estimate produced by the base learner for the training instance.

one class and normal for the other class. Both cases are illustrated in Figure 3.4, which shows the regions where a training example must fall relative to the class-specific thresholds in terms of its probability estimates, in order for the weight of the observation to be increased. In essence, OutBoost.B2 treats those observations that are especially difficult to classify the same as it treats observations that were properly classified.

3.5 Implementation

We implemented and evaluated our proposed machine-learning techniques with the *WEKA 3.7.12 Software Workbench* [14]. Developed using the Java programming language, WEKA is licensed under the GNU General Public License and as such is open source [14]. WEKA is a collection of machine-learning algorithms, which includes tools for classification, clustering, and visualization [14]. We chose to implement our techniques in WEKA due to it being open source and it offering a wide variety of pre-programmed classes that facilitated the creation and evaluation of classification models [14].

3.5.1 Proposed Techniques

We implemented GenThresh.B1 and GenThresh.B2 using the SingleClassifierEnhancer superclass provided by the WEKA 3.7.12 software workbench. OutBoost.B1, OutBoost.B2, and OutBoost.B1Man were implemented by modifying WEKA's version of AdaBoost.M1, which used the RandomizableSingleClassifierEnhancer superclass, to include the changes outlined in Section 3.4.

3.5.2 Base Learners

In order to explore the performance of our proposed machine-learning techniques we tested a wide range of supervised machine-learning methods as base learners. Table 3.2 details the machine-learning methods selected for our evaluation and the names of WEKA's implementations for the chosen techniques. No changes were made to WEKA's implementations and they were utilized with their default configuration. The general concepts of these base learners were described in Section 2.3.

3.5.3 AdaBoost.M1

During our evaluation, we used WEKA's pre-implemented version of AdaBoost.M1 [14].

Table 3.2: List of supervised machine-learning algorithms used as base learners, which were used to explore the performance of GenThresh and OutBoost. All base learners are available pre-implemented in the WEKA 3.7.12 Software Workbench [14].

Machine-Learning Technique	WEKA Implementation
DT	J48
NBC	NiaveBayes
BBN	BayesNet
SVM	SMO
SVM + LR	SMO with LR
ANN	MultilayerPerceptron

3.6 Data

We selected five different data sets to use as benchmarks for our proposed techniques. Details regarding the data sets can be found in Table 3.3, while Table 3.4 outlines the class distribution and number of features. We acquired all but the Corot data set from the UCI Machine Learning Repository and created two new data sets from the Disease and Madelon data sets to use as benchmarks for GenThresh.B1 and GenThresh.B2 [17]. The new data sets were created by taking each of the original data sets and randomly sampling a subset of the minority class without replacement in order to create an imbalanced class distribution for each data set. The new sample of the minority class was combined with the observations of the majority class in order to create two new data sets, where the minority class of the observations represented approximately 25% of the observations in the data set.

Table 3.3: List of data sets that were used as benchmarks for the evaluation of our proposed techniques.

Data	Description
Adult	The Adult data set is composed of features and observations extracted from the 1994 United States Census database and may be found on the UCI Machine Learning Repository [17]. We are tasked with predicting whether or not an individual makes over $50,000 a year [17].
Cancer	Dr. William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian created the Breast Cancer Wisconsin (Diagnostic) Data Set data set found on the UCI Machine Learning Repository [17]. We are tasked with predicting if an observation is malignant or benign [17].
Corot	The Corot Problem 1 Magnitude Limited Data Set is an unpublished data set where we are given the task of predicting which observations are planetary candidates.
disease	Robert Detrano, M.D., Ph.D. of the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation created the Heart Disease data set found on the UCI Machine Learning Repository [17]. We are tasked with predicting the presence or absence of heart disease [17].
Madelon	Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, and Gideon Dror created the Madelon Data Set found on the Machine Learning Repository [13, 17]. A number of features are included that have no predictive power [17]. We are tasked with separating the observations into two classes [17].
Disease Sample	We created an imbalanced sample of the Heart Disease data set by randomly sampling (without replacement) the minority class of the Heart Disease data set.
Madelon Sample	We created an imbalanced sample of the Madelon data set by randomly sampling (without replacement) the minority class of the Madelon data set.

Table 3.4: Class distribution for the selected data sets. This includes the number of features, number of instances for each class (count and percentage), and total number of instances.

Data	Features	Class 1	Class 2	Instances
Adult	15	7841 (24.08%)	24720 (75.92%)	32561
Cancer	10	241 (34.48%)	458 (65.52%)	699
Corot	5	225 (1.45%)	15317 (98.55%)	15542
Disease	14	139 (45.87%)	164 (54.13%)	303
Madelon	501	999 (49.97%)	1000 (50.03%)	1999
Disease Sample	14	55 (25.11%)	164 (74.89%)	219
Madelon Sample	501	334 (25.04%)	1000 (74.96%)	1334

The order of the classes were switched in all of data sets where the minority class was not the positive class. This was a necessary step to acquire minority class performance measures from WEKA, such as precision and recall.

3.7 Evaluation

The following section details how we achieved the performance assessment of our proposed machine-learning techniques. It describes how the performance of the base learners and proposed techniques were determined, as well as how they were compared.

Our proposed techniques were assessed in the WEKA software workbench in order to take advantage of WEKA's built in evaluation tools and the Experimenter platform [14]. Each proposed technique was evaluated using a variety of base learners (detailed in Table 3.2) and data sets (detailed in Table 3.3),

where the performance results were computed.

3.7.1 Baseline Creation

In order to evaluate the performance of our proposed techniques we created a performance baseline for the selected base learners and AdaBoost.M1. The base learners (detailed in Table 3.2) and the base learners in conjunction with AdaBoost.M1, were applied to all of the data sets listed in Table 3.3 (aside from the sampled data sets, which AdaBoost.M1 was not applied). We executed AdaBoost.M1 for 1000 iterations when using a DT, NBC, and BBN as a base learner and five iterations when using a SVM, SVM+LR, and ANN. Additionally, 10 iterations of AdaBoost.M1 were performed on the Cancer and Disease data sets using all of the base learners.

3.7.2 Proposed Techniques

We applied GenThresh.B1 and GenThresh.B2 in conjunction with all of the base learners, on the same data sets used to establish the base line for the base learners. Additionally, OutBoost.B1 and OutBoost.B2 were executed using the same base learners, data sets, and iterations as AdaBoost.M1 (aside from 10 iterations, which we did not repeat for OutBoost.B1 and OutBoost.B2).

Each combination of base learner and proposed technique was configured

with both a cutoff (α) of 0.0 ($\alpha = 0.0$) and 0.5 ($\alpha = 0.5$). A cutoff of 0.0 was selected so that all of the probability estimates could be used to find the class-specific thresholds. A cutoff of 0.5 was chosen with the assumption that any training example with an estimated probability of belonging to a class given the training example, could be an outlier for that class or part of another distribution (i.e. the other class) if the estimated probability for that class was smaller or equal to 0.5.

OutBoost.B1Man was executed for 10 iterations in combination with all of the base learners on both the Cancer and Disease data sets with identical class-specific thresholds ranging from 0.00 to 0.50 at .05 intervals. Finally, OutBoost.B1Man was applied to the Cancer data set using a BBN base learner for 5, 10, 15 and 20 iterations with the same class thresholds.

3.7.3 Chosen Performance Measures

Accuracy was chosen as a performance measure for the balanced data sets as we wished to use a commonly used measure that summarized the performance between both classes. Precision and recall was chosen for the imbalanced data sets as we were particularly interested in examining the effect of our techniques on these metrics. Other performance measures were examined but we restricted our evaluation to these three metrics in order to focus our evalua-

tion. Additionally, accuracy, precision, and recall are fairly intuitive metrics in their interpretation.

3.7.4 Estimating Generalization Performance

The mean estimated generalization performance for the base learners, AdaBoost.M1, and the proposed techniques were collected by applying repeated stratified k-fold CV to each data set, the basics of which are described in Section 2.7.6. Ten folds were utilized for the stratified CV and the process of stratified CV was repeated five times.[2] The mean performance results were calculated by averaging the results of the folds for each iteration, then averaging the results of the iterations. We also computed the sample standard deviation for the iterations. Generalization accuracy, precision, and recall were selected as performance metrics, where the generalization accuracy was calculated for all of the data sets and generalization precision and recall was calculated for the imbalanced data sets.

[2] Using k-fold CV in combination with Walsh's Outlier Test, adds additional conditions that must be observed in order for Walsh's Outlier Test to be effective. Specifically, Walsh's Outlier Test is limited to a minimum of more then 60 observations for a significance level of 0.1. Given that k-fold CV partitions all the training instances (where N is the number of instances) into k-folds and only utilizes $k-1$ of the folds for training, the inequality $\frac{N(k-1)}{k} > 60$ must hold true. If this inequality does not hold true the significance level must then be reduced in order to compensate for the smaller number of observations. All of our chosen data sets met this condition for $k = 10$.

3.7.5 Comparing Performance

The mean performance of AdaBoost.M1, GenThresh.B1, and GenThresh.B2 was compared to the mean performance of its associated base learner when applied to the same data set, and it was noted when they exceeded their base learner's performance for the chosen metrics.[3]

We also compared the performance of OutBoost.B1Man, OutBoost.B1, and OutBoost.B2 to the performance of AdaBoost.M1 when both the proposed technique and AdaBoost.M1 had the same combination of iterations, base learner, and data set. We paid special attention to when the proposed techniques outperformed AdaBoost.M1, AdaBoost.M1 and the its associated base learner, and when AdaBoost.M1 failed to outperform its base learner when the proposed technique did. The error bars in all of visualizations used to make comparisons are one standard deviation.

3.8 Chapter Summary

In this chapter we constructed a framework for discovering class-specific threshold by applying Walsh's Outlier Test to the probability estimates produced by

[3] Tests of significance were not utilized during the comparisons, due to the lack of statistical tests of significance that were applicable to repeated stratified k-fold CV. It is recommended in [32] to use the corrected resampled t-test, however, we were not convinced that it was applicable given that k-fold cross validation was repeated.

machine learning techniques, referred to as base learners. Two methods of utilizing these class-specific thresholds were proposed and illustrated, which we referred to as GenThresh and OutBoost, where OutBoost is a modification of the popular boosting technique referred to as AdaBoost. Details were provided regarding our approach to evaluating and comparing these proposed techniques to AdaBoost.M1 (with OutBoost) and the selected base learners, where each technique was applied to a selected group of benchmark data sets and the results of which are presented and analyzed in the following chapter.

Chapter 4

Results and Discussion

The following chapter presents and discuses our results as well as the potential implications of these results. We begin by describing and examining those results collected for the base learners and AdaBoost.M1 before moving on to investigating the results produced by GenThresh and OutBoost.

4.1 Baseline

The following section communicates the performance results collected for the base learners and AdaBoost.M1, where the performance results were used as a baseline to draw comparisons between the base learners, AdaBoost.M1, and the proposed techniques. The base learners in Table 3.2 and AdaBoost.M1 in conjunction with the base learners, were applied to the selected data sets

listed in Table 3.3 as described in Section 3.7 of our Methods.

4.1.1 Base Learners

The mean and sample standard deviation of the accuracy of the base learners applied to all of the data sets, and the mean and sample standard deviation of the precision and recall of the minority class of the imbalanced data sets are shown in Table 4.1. These results served as a base line for the performance of AdaBoost.M1 and our proposed techniques, given the same base learners.

Referring to Table 4.1, we discerned a few trends regarding the performance of the base learners for the given data sets. When the base learners were applied to the Corot data set we witnessed accuracy of 98.55% for almost all of the base learners. Given that the majority class represents 98.55% of the training examples (the class distribution in Table 3.4), these results imply that the base learners were not able to distinguish between the minority class and the majority class, instead classifying all of the examples as the majority class. Further supporting this hypothesis, we recognized that none of the observations that were classified as the minority class were actually of the minority class for any of the base learners, given that the recall and precision of the base learners were zero. This poor performance likely due to the severe imbalance of the training examples.

Table 4.1: Mean and sample standard deviation of the selected performance metrics for all of the base learners.

Base Learners: Accuracy (Percent)

Data	DT	NBC	BBN	SVM	SVM+LR	ANN
Disease	79.36 ± 0.97	83.50 ± 0.44	82.51 ± 0.64	83.63 ± 0.63	83.37 ± 0.28	78.55 ± 0.74
Madelon	68.80 ± 1.32	58.66 ± 0.34	61.16 ± 0.42	54.75 ± 0.60	54.79 ± 0.62	59.20 ± 0.36
Cancer	95.02 ± 0.36	96.05 ± 0.08	97.20 ± 0.16	96.71 ± 0.20	96.54 ± 0.16	95.34 ± 0.57
Adult	86.18 ± 0.04	83.48 ± 0.02	83.83 ± 0.02	85.05 ± 0.02	85.06 ± 0.04	82.40 ± 0.65
Corot	98.55 ± 0.00	98.53 ± 0.02	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
Disease Sample	80.09 ± 1.41	86.47 ± 0.26	85.20 ± 0.75	86.58 ± 0.54	86.48 ± 0.26	79.19 ± 1.89
Madelon Sample	69.76 ± 1.12	68.55 ± 0.35	65.37 ± 0.34	65.20 ± 0.50	64.68 ± 0.66	67.92 ± 1.18

Base Learners: Precision

Data	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.9260 ± 0.0054	0.9184 ± 0.0021	0.9407 ± 0.0020	0.9492 ± 0.0047	0.9504 ± 0.0027	0.9370 ± 0.0098
Adult	0.7541 ± 0.0022	0.7153 ± 0.0007	0.6296 ± 0.0005	0.7440 ± 0.0004	0.7380 ± 0.0012	0.6937 ± 0.0236
Corot	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.6605 ± 0.0502	0.7578 ± 0.0108	0.7392 ± 0.0175	0.8339 ± 0.0312	0.8116 ± 0.0168	0.6151 ± 0.0631
Madelon Sample	0.3942 ± 0.0277	0.3607 ± 0.0046	0.3571 ± 0.0062	0.2877 ± 0.0108	0.2855 ± 0.0139	0.3190 ± 0.0262

Base Learners: Recall

Data	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.9337 ± 0.0107	0.9744 ± 0.0019	0.9834 ± 0.0042	0.9586 ± 0.0031	0.9519 ± 0.0046	0.9303 ± 0.0091
Adult	0.6327 ± 0.0024	0.5217 ± 0.0007	0.7985 ± 0.0009	0.5783 ± 0.0012	0.5888 ± 0.0008	0.5765 ± 0.0624
Corot	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.5240 ± 0.0196	0.7200 ± 0.0135	0.6633 ± 0.0085	0.6033 ± 0.0122	0.6213 ± 0.0194	0.5860 ± 0.0345
Madelon Sample	0.3863 ± 0.0487	0.3246 ± 0.0061	0.4741 ± 0.0126	0.2661 ± 0.0101	0.2739 ± 0.0144	0.2391 ± 0.0167

Additionally, we noted that the performance of the base learners on the Cancer data set left little room for improvement, given that the performance ranged from 95.02% to 97.20% depending on the base learner. These results imply that if we see increased performance for any of our proposed methods, it will be smaller in scale than what could be seen with a data set where the base learners did not perform as highly, such as the Disease or Adult data set.

At first glance, the increased accuracy for the Disease Sample and Madelon Sample data sets compared to the accuracy of the base learners on the original data sets, may appear interesting. However, we suggest this was likely due to the new class imbalance and the majority bias of the accuracy measure, rather than an increased ability to distinguish between classes. These thoughts are further supported when examining the recall of the minority class, which was fairly low for both sampled data sets. Indicating that few examples of the minority class were classified as the minority class.

Referring to Table 4.1, we observed that the accuracy of the base learners on the Madelon data set ranged from 54.75% to 68.60% in accuracy, in some cases indicating that the base learner performed little better than chance when attempting to distinguish between the classes. The results suggest that the Madelon data set is an especially difficult classification problem, most likely due to the numerous features without predictive powers [17].

Examining the results in Table 4.1 revealed that using an ANN as a base learner did not produce exceptional results. Although it performed relativity well when compared to the other base learners, it did not perform the best for any of the metrics examined for any of the data sets. Although this may seem discouraging, this gave us an opportunity to observe the effect of combining OutBoost with a weaker classifier. This is important to evaluate, given the purpose of AdaBoost is to combine weak classifiers together to create a stronger classifier [12].

As a final note, the Adult, Corot, and Madelon Sample data sets displayed the greatest potential for increased recall and precision, given the low metric values produced across all of the base learners. If we are to see increased recall or precision in our proposed techniques it is more likely to appear in these three data sets.

4.1.2 AdaBoost.M1

The computed mean accuracy and sample standard deviation for AdaBoost.M1 having been applied to the selected data sets in combination with the different base learners, are shown in Table 4.2. Also displayed in Table 4.2 are the mean and sample standard deviation of the precision and recall of the minority class given the same base learners. The instances marked in bold demonstrated

higher performance than the associated base learner for the given metric.

We observed nine instances of AdaBoost.M1 exhibiting higher accuracy than the associated base learner, where three of the nine occurrences were more than 1.00% greater, implying a substantial difference in accuracy for those three examples. Given the potentially low number of consequential improvements in accuracy, our results indicate that AdaBoost.M1 does not perform better overall in terms of accuracy, given the chosen base learners and data sets.

This implication was further reinforced when we plotted the accuracy of AdaBoost.M1 against its associated base learner, as shown in Figure 4.1. In order to interpret Figure 4.1, note that each point on each scatter plot indicates the accuracy of AdaBoost.M1 vs. the accuracy of the base learner, where points that fall above the line $y = x$ demonstrate improvements in accuracy over the base learner, and points that fall below the line indicate lower performance than the base learner.

The scatter plots are grouped in such a way that each plot in the figure show the same observations, but with additional details. The top left plot displays the original comparison, while the top right plot displays error bars for the same points. The horizontal error bars are the standard deviation of the base learner and the vertical error bar are the standard deviation of

Table 4.2: Mean and sample standard deviation of the selected performance metrics for AdaBoost.M1.

AdaBoost.M1: Accuracy (Percent)

Data	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Disease	$\mathbf{80.11 \pm 0.82}$	81.98 ± 0.77	82.45 ± 0.44	83.63 ± 0.63	83.30 ± 0.37	$\mathbf{79.75 \pm 1.11}$
Madelon	63.62 ± 0.59	54.60 ± 0.87	$\mathbf{61.46 \pm 0.50}$	53.01 ± 0.91	53.46 ± 1.05	59.20 ± 0.36
Cancer	$\mathbf{96.65 \pm 0.36}$	95.54 ± 0.26	95.94 ± 0.28	96.62 ± 0.13	$\mathbf{96.57 \pm 0.14}$	$\mathbf{95.59 \pm 0.67}$
Adult	84.38 ± 0.16	83.48 ± 0.02	$\mathbf{85.28 \pm 0.09}$	$\mathbf{85.07 \pm 0.02}$	85.05 ± 0.03	$\mathbf{82.55 \pm 0.61}$
Corot	98.55 ± 0.02	98.53 ± 0.02	98.09 ± 0.05	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00

AdaBoost.M1: Precision

Data	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	$\mathbf{0.9407 \pm 0.0057}$	$\mathbf{0.9379 \pm 0.0048}$	$\mathbf{0.9453 \pm 0.0041}$	0.9489 ± 0.0044	$\mathbf{0.9510 \pm 0.0038}$	$\mathbf{0.9408 \pm 0.0061}$
Adult	0.6913 ± 0.0042	0.7153 ± 0.0007	$\mathbf{0.6771 \pm 0.0023}$	$\mathbf{0.7450 \pm 0.0006}$	0.7376 ± 0.0007	$\mathbf{0.7039 \pm 0.0198}$
Corot	$\mathbf{0.4117 \pm 0.1023}$	0.0000 ± 0.0000	$\mathbf{0.0897 \pm 0.0135}$	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

AdaBoost.M1: Recall

Data	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	$\mathbf{0.9668 \pm 0.0065}$	0.9345 ± 0.0058	0.9394 ± 0.0096	0.9561 ± 0.0036	0.9519 ± 0.0046	$\mathbf{0.9336 \pm 0.0156}$
Adult	$\mathbf{0.6357 \pm 0.0025}$	0.5217 ± 0.0007	0.7433 ± 0.0027	0.5778 ± 0.0017	$\mathbf{0.5891 \pm 0.0013}$	0.5456 ± 0.0543
Corot	$\mathbf{0.0338 \pm 0.0041}$	0.0000 ± 0.0000	$\mathbf{0.0357 \pm 0.0033}$	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

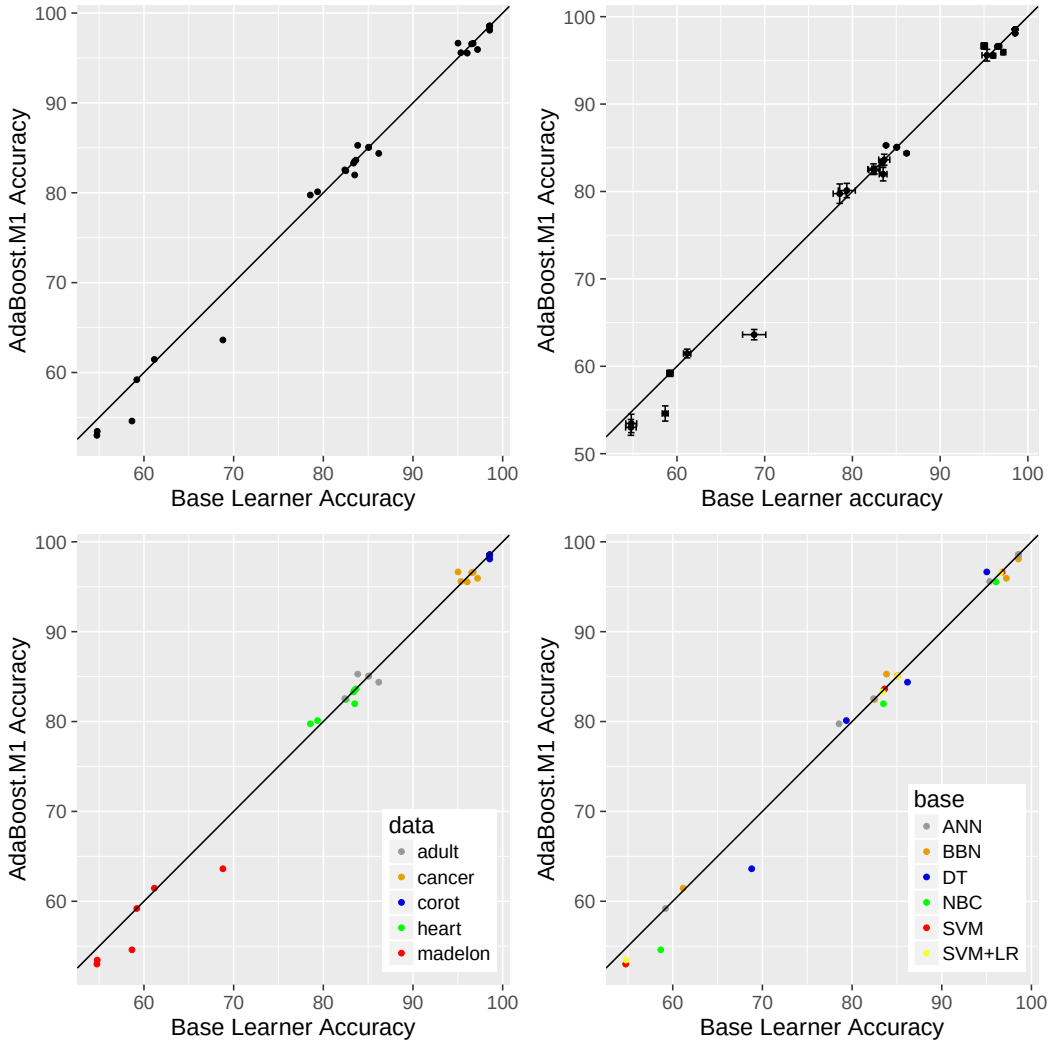


Figure 4.1: Comparison of the mean accuracy of AdaBoost.M1 and its associated base learner. Mixed results were observed, with AdaBoost.M1 performing both better and worse than its base learner, while the difference in performance is somewhat minimal.

AdaBoost.M1. The bottom left and bottom right plots color code the same points, where the former are color coded given the data set and the latter are color coded given the base learner.

Figure 4.1 reveals a few examples where AdaBoost.M1 produced higher accuracy than its associated base learner, where we see points that fall above the line. However, numerous examples appear on the diagonal line indicating the same performance, and below the line that suggest poorer performance. This may be due to the nature of AdaBoost.M1, which was conceived to boost the performance of a weak learner, however many of the base learners used for AdaBoost.M1 were strong learners. Overall, these results reinforce our initial proposition that AdaBoost.M1 produced mixed results in terms of accuracy, which may lead to similar mixed results in our evaluation of OutBoost, given that OutBoost is a modification of AdaBoost.M1.

Referring to Table 4.2, we note 10 instances where AdaBoost.M1 produced higher precision than the associated base learner on the imbalanced data sets, with a difference greater than 0.01 for six of the 10 instances, suggesting a fairly substantial difference for the six instances. These results suggest that AdaBoost.M1 may be very beneficial for the precision of those training examples classified as the minority class. Figure 4.2 reinforces this suggestion, where we see that most of the points fall on or above the diagonal line, indicating

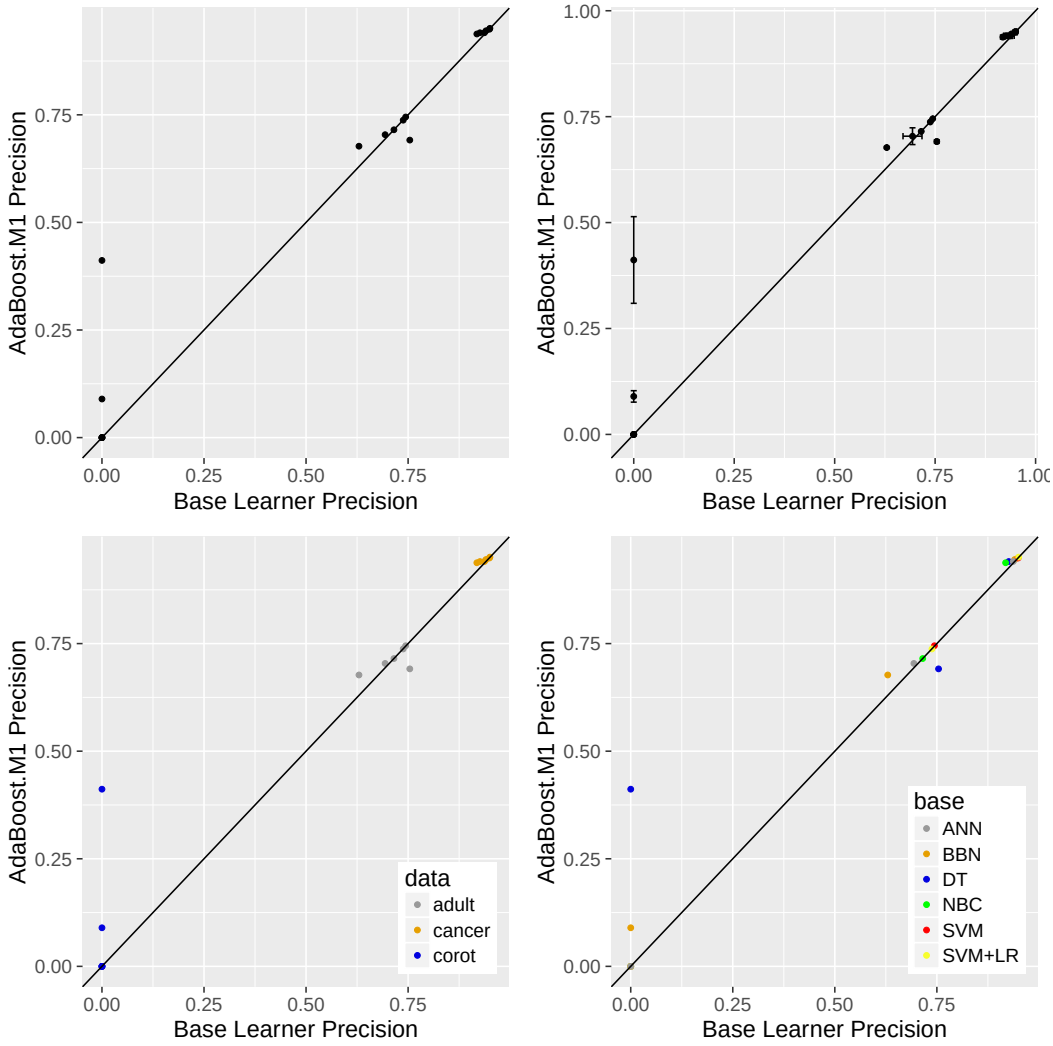


Figure 4.2: Comparison of the mean precision of AdaBoost.M1 and its associated base learner. Positive results were observed, with many examples of AdaBoost.M1 performing the same or better than its base learner.

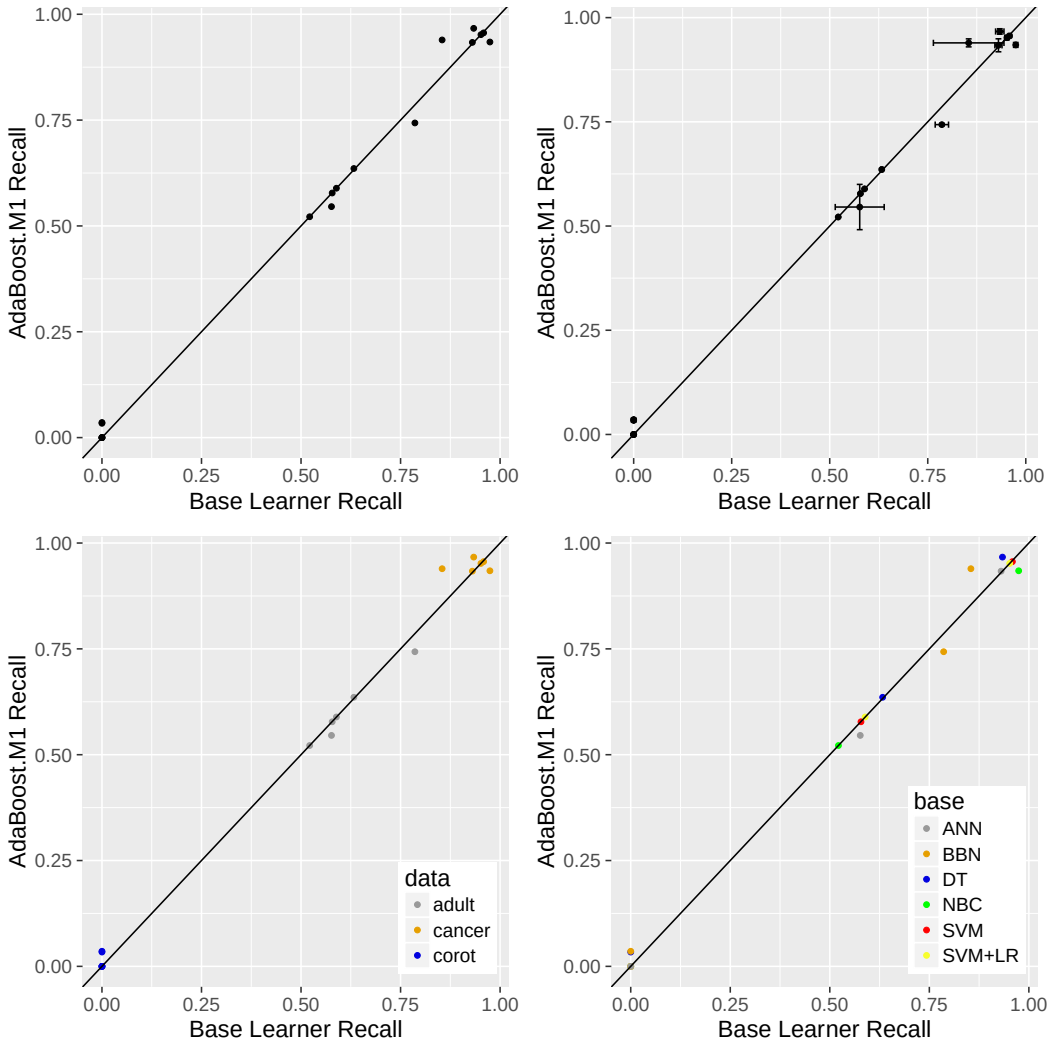


Figure 4.3: Comparison of the mean recall of AdaBoost.M1 and its associated base learner. Mixed results were observed, with examples of AdaBoost.M1 performing both better and worse than its base learner.

the same or better precision than the base learner.

Finally, we observed six instances of AdaBoost.M1 exhibiting higher recall than the associated base learner on the imbalanced data sets in Table 4.2. We also observed a greater than 0.01 change for three of those instances, suggesting a consequential difference in recall for those three examples. Figure 4.3 also suggests mixed performance for the recall measure, where numerous points fall above and below the line, suggesting that although AdaBoost.M1 may increase the recall of the minority class for a base learner, it may also decrease the recall.

Increased recall or precision in the minority class is somewhat expected, as AdaBoost.M1 focuses the base learners attention on training examples that are difficult to classify, such as those observations that are of the minority class but are classified as the majority class. Given the results produced by AdaBoost.M1, we could infer that if we see an improvement in the performance of OutBoost compared to AdaBoost.M1, we may see further improvements in precision or recall.

Overall, these results served as a base line for the performance of OutBoost. Given that OutBoost.B1 and OutBoost.B2 are modifications of AdaBoost.M1, these results suggest that we may also see mixed results for our evaluation of OutBoost.B1 and OutBoost.B2. The results further suggest that we may

expect to see only a few examples of increased performance across the observed metrics, and even fewer instances demonstrating a large difference.

4.2 GenThresh

The following section presents our evaluation of GenThresh.B1 (Algorithm 3) and GenThresh.B2 (Algorithm 5) by describing and discussing the performance results collected during the evaluation. GenThresh.B1 and GenThresh.B2 in combination with the base learners listed in Table 3.2, were applied to the selected data sets listed in Table 3.3 as described in Section 3.7 of the Methods. Comparisons were drawn between the proposed techniques and the base learners by determining the mean accuracy for all of the selected data sets and the mean precision and recall of the minority class for the imbalanced data sets.

4.2.1 GenThresh.B1

Table 4.3 displays the mean and sample standard deviation of the accuracy of GenThresh.B1 combined with the base learners, applied to all of the data sets. Also shown in Table 4.3, are the mean and sample standard deviation of the precision and recall of the minority class of the imbalanced data sets. As with

the evaluation of AdaBoost.M1, the observations marked in bold demonstrated better performance than the associated base learner for the given metric.

Examining the results in Table 4.3 revealed 19 instances where GenThresh.-B1 exhibited higher recall than the associated base learner on the imbalanced data sets, where a greater than 0.01 increase was observed for 16 of those examples, indicating a considerable improvement in the recall of the minority class over that which was produced by the base learner in Table 4.1. Our results suggest that GenThresh.B1 may be very beneficial in improving the recall of the minority class.

These potentially positive ramifications are further reinforced by Figure 4.4 and Figure 4.5 where the recall of GenThresh.B1 is plotted against its associated base learner. Figure 4.4 visualizes the results produced when a cutoff of 0.0 ($\alpha = 0.00$) was utilized and Figure 4.5 shows the results for a cutoff of 0.5 ($\alpha = 0.50$). From the data shown in both figures, we observed that the points fall on or above the line in almost all cases, suggesting that combining GenThresh.B1 and a base learner may produce higher recall in the minority class than a given base learner, or at least it may have no negative effect on the recall.

We also noted that choosing a cutoff of 0.0 rather than 0.5 produced higher recall for more examples (fourteen vs. five), which can be easily observed when

Table 4.3: Mean and sample standard deviation of the selected performance metrics for GenThresh.B1.

GenThresh.B1: Accuracy (Percent)

Data	α	DT	NBC	BBN	SMO	SVM+LR	ANN
Disease	0.00	78.37 ± 0.42	83.50 ± 0.44	82.38 ± 0.69	83.63 ± 0.63	83.37 ± 0.28	78.22 ± 0.42
	0.50	$\mathbf{79.43 \pm 0.89}$	$\mathbf{83.56 \pm 0.46}$	81.92 ± 0.61	83.63 ± 0.63	83.30 ± 0.37	78.29 ± 1.11
Madelon	0.00	68.79 ± 1.32	58.66 ± 0.34	$\mathbf{61.44 \pm 0.63}$	54.75 ± 0.60	54.68 ± 0.77	$\mathbf{59.23 \pm 0.42}$
	0.50	68.80 ± 1.32	58.66 ± 0.34	60.91 ± 0.44	54.75 ± 0.60	54.63 ± 0.90	59.12 ± 0.48
Cancer	0.00	91.76 ± 0.54	95.73 ± 0.34	96.22 ± 0.22	96.71 ± 0.20	96.54 ± 0.16	$\mathbf{95.42 \pm 0.92}$
	0.50	95.02 ± 0.36	96.05 ± 0.08	$\mathbf{97.25 \pm 0.12}$	96.71 ± 0.20	96.54 ± 0.16	95.31 ± 0.62
Adult	0.00	67.31 ± 1.37	83.48 ± 0.02	83.71 ± 0.16	85.05 ± 0.02	85.06 ± 0.04	82.40 ± 0.65
	0.50	86.14 ± 0.06	83.48 ± 0.02	$\mathbf{83.95 \pm 0.04}$	85.05 ± 0.02	85.06 ± 0.04	82.40 ± 0.65
Corot	0.00	98.55 ± 0.00	98.53 ± 0.02	98.55 ± 0.00	98.55 ± 0.00	96.62 ± 4.31	97.84 ± 0.43
	0.50	98.55 ± 0.00	98.53 ± 0.02	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
Disease Sample	0.00	65.34 ± 4.05	86.47 ± 0.26	71.70 ± 1.52	86.58 ± 0.54	86.48 ± 0.26	78.37 ± 1.06
	0.50	79.81 ± 1.33	86.47 ± 0.26	80.63 ± 1.06	86.58 ± 0.54	86.21 ± 0.49	79.10 ± 2.07
Madelon Sample	0.00	60.00 ± 1.54	68.55 ± 0.35	58.76 ± 0.91	65.20 ± 0.50	55.58 ± 1.78	67.87 ± 1.17
	0.50	69.76 ± 1.12	68.55 ± 0.35	64.05 ± 0.38	65.20 ± 0.50	64.20 ± 0.96	$\mathbf{67.93 \pm 1.18}$

GenThresh.B1: Precision

Data	α	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.00	0.8311 ± 0.0108	0.8982 ± 0.0080	0.9085 ± 0.0059	0.9492 ± 0.0047	0.9504 ± 0.0027	0.9255 ± 0.0201
	0.50	0.9260 ± 0.0054	0.9184 ± 0.0021	0.9407 ± 0.0020	0.9492 ± 0.0047	0.9504 ± 0.0027	0.9362 ± 0.0109
Adult	0.00	0.4210 ± 0.0116	0.7153 ± 0.0007	0.6272 ± 0.0034	0.7440 ± 0.0004	0.7380 ± 0.0012	0.6937 ± 0.0236
	0.50	0.7456 ± 0.0028	0.7153 ± 0.0007	$\mathbf{0.6336 \pm 0.0009}$	0.7440 ± 0.0004	0.7380 ± 0.0012	0.6937 ± 0.0236
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	$\mathbf{0.0003 \pm 0.0006}$	$\mathbf{0.0017 \pm 0.0014}$
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.00	0.4689 ± 0.0587	0.7578 ± 0.0108	0.4951 ± 0.0322	0.8339 ± 0.0312	0.8116 ± 0.0168	0.5985 ± 0.0439
	0.50	0.6563 ± 0.0489	0.7578 ± 0.0108	0.6203 ± 0.0184	0.8339 ± 0.0312	0.8056 ± 0.0078	0.6118 ± 0.0680
Madelon Sample	0.00	0.3254 ± 0.0099	0.3607 ± 0.0046	0.3398 ± 0.0038	0.2877 ± 0.0108	0.2781 ± 0.0090	0.3182 ± 0.0259
	0.50	0.3942 ± 0.0277	0.3607 ± 0.0046	0.3500 ± 0.0056	0.2877 ± 0.0108	0.2795 ± 0.0127	$\mathbf{0.3194 \pm 0.0262}$

GenThresh.B1: Recall

Data	α	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.00	$\mathbf{0.9627 \pm 0.0094}$	$\mathbf{0.9918 \pm 0.0030}$	$\mathbf{0.9951 \pm 0.0019}$	0.9586 ± 0.0031	0.9519 ± 0.0046	$\mathbf{0.9485 \pm 0.0082}$
	0.50	0.9337 ± 0.0107	0.9744 ± 0.0019	$\mathbf{0.9851 \pm 0.0038}$	0.9586 ± 0.0031	0.9519 ± 0.0046	0.9303 ± 0.0091
Adult	0.00	$\mathbf{0.9285 \pm 0.0048}$	0.5217 ± 0.0007	$\mathbf{0.8014 \pm 0.0045}$	0.5783 ± 0.0012	0.5888 ± 0.0008	0.5765 ± 0.0624
	0.50	$\mathbf{0.6451 \pm 0.0028}$	0.5217 ± 0.0007	0.7916 ± 0.0006	0.5783 ± 0.0012	0.5888 ± 0.0008	0.5765 ± 0.0624
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	$\mathbf{0.0200 \pm 0.0447}$	$\mathbf{0.0105 \pm 0.0095}$
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.00	$\mathbf{0.7380 \pm 0.0440}$	0.7200 ± 0.0135	$\mathbf{0.7947 \pm 0.0240}$	0.6033 ± 0.0122	0.6213 ± 0.0194	$\mathbf{0.6187 \pm 0.0334}$
	0.50	0.5240 ± 0.0196	0.7200 ± 0.0135	$\mathbf{0.7113 \pm 0.0174}$	0.6033 ± 0.0122	0.6213 ± 0.0194	$\mathbf{0.5900 \pm 0.0367}$
Madelon Sample	0.00	$\mathbf{0.5364 \pm 0.0537}$	0.3246 ± 0.0061	$\mathbf{0.6833 \pm 0.0273}$	0.2661 ± 0.0101	$\mathbf{0.4907 \pm 0.0259}$	0.2391 ± 0.0167
	0.50	0.3863 ± 0.0487	0.3246 ± 0.0061	$\mathbf{0.5101 \pm 0.0115}$	0.2661 ± 0.0101	0.2739 ± 0.0104	0.2391 ± 0.0167

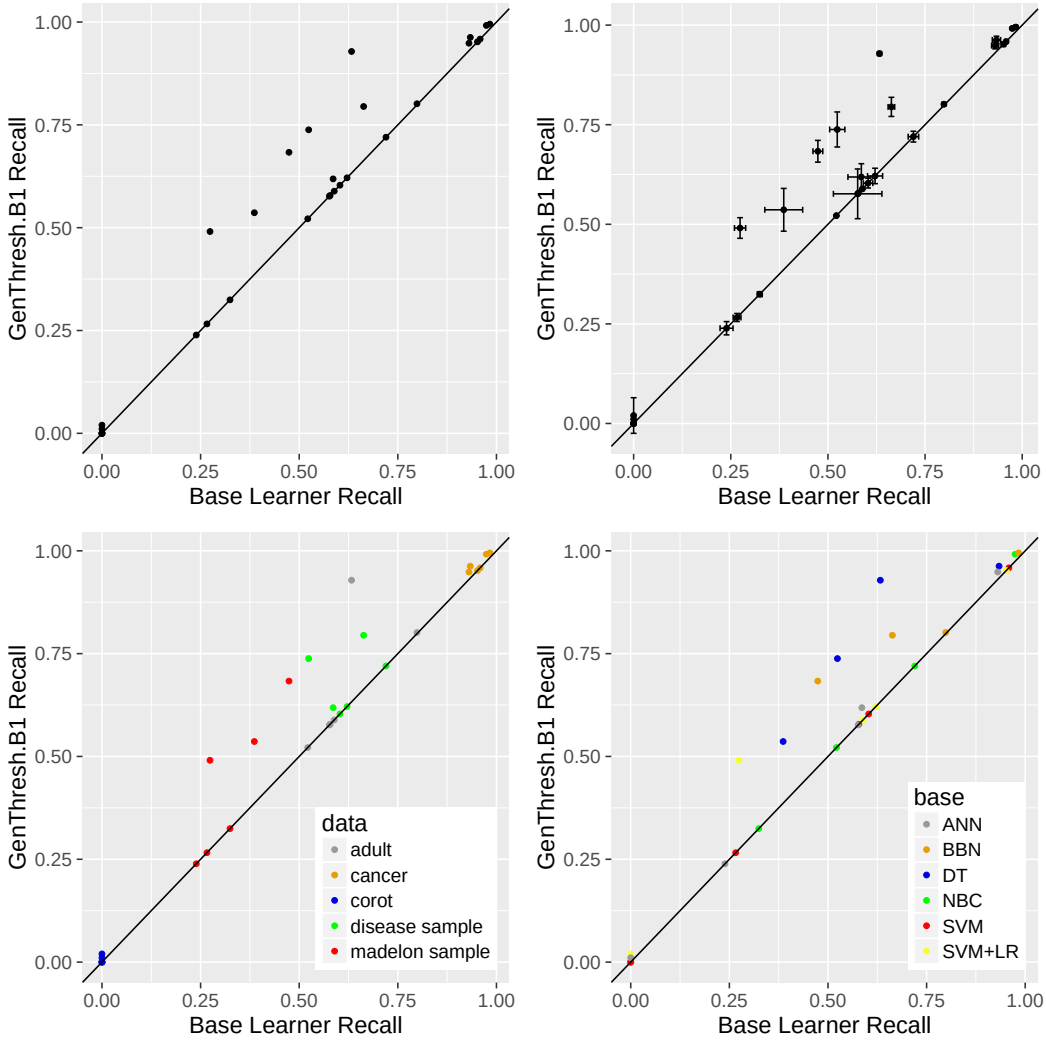


Figure 4.4: Comparison of the mean recall of GenThresh.B1 its associated base learner, where $\alpha = 0.0$. Numerous examples were observed where GenThresh.B1 produced higher recall, indicating that GenThresh.B1 may be beneficial in increasing the recall of the base learner.

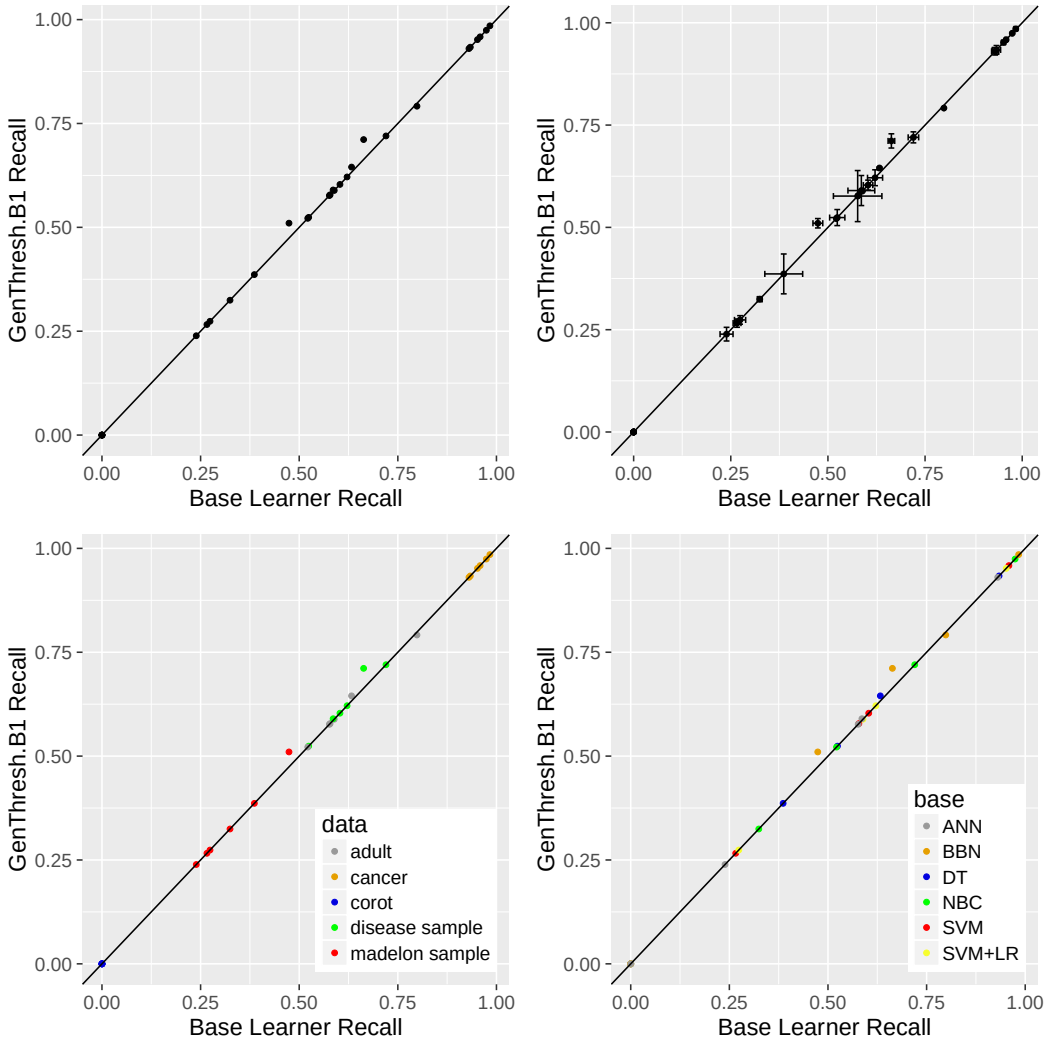


Figure 4.5: Comparison of the mean recall of GenThresh.B1 its associated base learner, where $\alpha = 0.5$. Examples of increased recall were observed, where the difference in recall was not as dramatic as when a cutoff of $\alpha = 0.0$ was specified. Suggesting that a cutoff of $\alpha = 0.5$ may not be the preferred choice, if dramatic changes in recall is the objective.

comparing Figure 4.4 and Figure 4.4. This suggests that allowing Walsh's Outlier Test to use all of the probability estimates produced during training is more beneficial for the recall of the minority class.

Overall, these results suggest that GenThesh.B1 may be useful for an imbalanced classification scenario, where we emphasize increasing recall over increasing accuracy or precision. Particularly, GenThesh.B1 may be useful in scenarios where TP are heavily rewarded and FP are not penalized. Consequently, GenThesh.B1 may be useful when faced with an outlier detection scenario where finding the highest number of outliers possible, within reason, is more important than confirming that all of the discovered outliers are true outliers.

The importance of a scenario where ignoring the penalties produced by FP is made obvious when examining the precision observed for GenThresh.B1. We noted four instances where GenThresh.B1 produced higher precision than the associated base learner on the imbalanced data sets, but there was less than a 0.01 difference for all of these observations, indicating they are of little consequence. This implies that GenThresh.B1 may produce more false positives, as the recall increased but the precision did not.

Additionally, we observed eight instances of GenThresh.B1 exhibiting higher accuracy (in Table 4.3) than the associated base learner. However, less than

a 1.00% difference existed between GenThresh.B1 and the associated base learner for all of these observations. Suggesting that GenThresh.B1 does not improve the accuracy of the base learner.

4.2.2 GenThresh.B2

The results for GenThresh.B2 are presented in Table 4.4, where the mean and sample standard deviation of the accuracy of GenThresh.B2 are provided for each base learner and data set combination. Also included are the mean and sample standard deviation for the precision and recall of the minority class. As with the evaluation of GenThresh.B1, the observations marked in bold demonstrated higher performance than the associated base learner in the given metric.

Comparing the results produced by GenThresh.B2 in Table 4.4 to the results produced by GenThresh.B1 in Table 4.3 revealed that both techniques produced identical results for accuracy, precision, and recall. Therefore, the discussion of the individual results found in Section 4.2.1 applies to GenThresh.B2.

The identical results suggest that using the average of the two thresholds vs. the average probability estimate of those observations that fall between the two thresholds, makes little difference to the application of the general

threshold in GenThresh. Additionally, our results imply that it may be better to chose GenThresh.B1 over GenThresh.B2, as the complexity of determining the general threshold τ_g in OutBoost.B1 is reduced in comparison to GenThresh.B2.

4.3 OutBoost

The following section describes and discusses the performance results collected during our evaluation of OutBoost.B1 (Algorithm 10) and OutBoost.B2 (Algorithm 11). OutBoost.B1 and OutBoost.B2 were applied to the selected data sets listed in Table 3.3 in combination with the base learners in Table 3.2, as described in Section 3.7 of the Methods. Comparisons were drawn between the proposed techniques, AdaBoost.M1, and the base learners, by determining the mean accuracy for all of the selected data sets and the mean precision and recall of the minority class for the imbalanced data sets. Additionally, we noted when OutBoost.B1 and OutBoost.B2 produced higher performance metrics than AdaBoost.M1 when AdaBoost.M1 failed to produce higher performance than its associated base learner.

Table 4.4: Mean and sample standard deviation of the selected performance metrics for GenThresh.B2.

GenThresh.B2: Accuracy (Percent)

Data	α	DT	NBC	BBN	SVM	SVM+LR	ANN
Disease	0.00	78.37 ± 0.42	83.50 ± 0.44	82.38 ± 0.69	83.63 ± 0.63	83.37 ± 0.28	78.22 ± 0.42
	0.50	$\mathbf{79.43 \pm 0.89}$	$\mathbf{83.56 \pm 0.46}$	81.92 ± 0.61	83.63 ± 0.63	83.30 ± 0.37	78.29 ± 1.11
Madelon	0.00	68.79 ± 1.32	58.66 ± 0.34	$\mathbf{61.44 \pm 0.63}$	54.75 ± 0.60	54.68 ± 0.77	$\mathbf{59.23 \pm 0.42}$
	0.50	68.80 ± 1.32	58.66 ± 0.34	60.91 ± 0.44	54.75 ± 0.60	54.63 ± 0.90	59.12 ± 0.48
Cancer	0.00	91.76 ± 0.54	95.73 ± 0.34	96.22 ± 0.22	96.71 ± 0.20	96.54 ± 0.16	$\mathbf{95.42 \pm 0.92}$
	0.50	95.02 ± 0.36	96.05 ± 0.08	$\mathbf{97.25 \pm 0.12}$	96.71 ± 0.20	96.54 ± 0.16	95.31 ± 0.62
Adult	0.00	67.31 ± 1.37	83.48 ± 0.02	83.71 ± 0.16	85.05 ± 0.02	85.06 ± 0.04	82.40 ± 0.65
	0.50	86.14 ± 0.06	83.48 ± 0.02	$\mathbf{83.95 \pm 0.04}$	85.05 ± 0.02	85.06 ± 0.04	82.40 ± 0.65
Corot	0.00	98.55 ± 0.00	98.53 ± 0.02	98.55 ± 0.00	98.55 ± 0.00	96.62 ± 4.31	97.84 ± 0.43
	0.50	98.55 ± 0.00	98.53 ± 0.02	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
Disease Sample	0.00	65.34 ± 4.05	86.47 ± 0.26	71.70 ± 1.52	86.58 ± 0.54	86.48 ± 0.26	78.37 ± 1.06
	0.50	79.81 ± 1.33	86.47 ± 0.26	80.63 ± 1.06	86.58 ± 0.54	86.21 ± 0.49	79.10 ± 2.07
Madelon Sample	0.00	60.00 ± 1.54	68.55 ± 0.35	58.76 ± 0.91	65.20 ± 0.50	55.58 ± 1.78	67.87 ± 1.17
	0.50	69.76 ± 1.12	68.55 ± 0.35	64.05 ± 0.38	65.20 ± 0.50	64.20 ± 0.96	$\mathbf{67.93 \pm 1.18}$

GenThresh.B2: Precision

Data	α	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.00	0.8311 ± 0.0108	0.8982 ± 0.0080	0.9085 ± 0.0059	0.9492 ± 0.0047	0.9504 ± 0.0027	0.9255 ± 0.0201
	0.50	0.9260 ± 0.0054	0.9184 ± 0.0021	0.9407 ± 0.0020	0.9492 ± 0.0047	0.9504 ± 0.0027	0.9362 ± 0.0109
Adult	0.00	0.4210 ± 0.0116	0.7153 ± 0.0007	0.6272 ± 0.0034	0.7440 ± 0.0004	0.7380 ± 0.0012	0.6937 ± 0.0236
	0.50	0.7456 ± 0.0028	0.7153 ± 0.0007	$\mathbf{0.6336 \pm 0.0009}$	0.7440 ± 0.0004	0.7380 ± 0.0012	0.6937 ± 0.0236
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	$\mathbf{0.0003 \pm 0.0006}$	$\mathbf{0.0017 \pm 0.0014}$
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.00	0.4689 ± 0.0587	0.7578 ± 0.0108	0.4951 ± 0.0322	0.8339 ± 0.0312	0.8116 ± 0.0168	0.5985 ± 0.0439
	0.50	0.6563 ± 0.0489	0.7578 ± 0.0108	0.6203 ± 0.0184	0.8339 ± 0.0312	0.8056 ± 0.0078	0.6118 ± 0.0680
Madelon Sample	0.00	0.3254 ± 0.0099	0.3607 ± 0.0046	0.3398 ± 0.0038	0.2877 ± 0.0108	0.2781 ± 0.0090	0.3182 ± 0.0259
	0.50	0.3942 ± 0.0277	0.3607 ± 0.0046	0.3500 ± 0.0056	0.2877 ± 0.0108	0.2795 ± 0.0127	$\mathbf{0.3194 \pm 0.0262}$

GenThresh.B2: Recall

Data	α	DT	NBC	BBN	SVM	SVM+LR	ANN
Cancer	0.00	$\mathbf{0.9627 \pm 0.0094}$	$\mathbf{0.9918 \pm 0.0030}$	$\mathbf{0.9951 \pm 0.0019}$	0.9586 ± 0.0031	0.9519 ± 0.0046	$\mathbf{0.9485 \pm 0.0082}$
	0.50	0.9337 ± 0.0107	0.9744 ± 0.0019	$\mathbf{0.9851 \pm 0.0038}$	0.9586 ± 0.0031	0.9519 ± 0.0046	0.9303 ± 0.0091
Adult	0.00	$\mathbf{0.9285 \pm 0.0048}$	0.5217 ± 0.0007	$\mathbf{0.8014 \pm 0.0045}$	0.5783 ± 0.0012	0.5888 ± 0.0008	0.5765 ± 0.0624
	0.50	$\mathbf{0.6451 \pm 0.0028}$	0.5217 ± 0.0007	0.7916 ± 0.0006	0.5783 ± 0.0012	0.5888 ± 0.0008	0.5765 ± 0.0624
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	$\mathbf{0.0200 \pm 0.0447}$	$\mathbf{0.0105 \pm 0.0095}$
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Disease Sample	0.00	$\mathbf{0.7380 \pm 0.0440}$	0.7200 ± 0.0135	$\mathbf{0.7947 \pm 0.0240}$	0.6033 ± 0.0122	0.6213 ± 0.0194	$\mathbf{0.6187 \pm 0.0334}$
	0.50	0.5240 ± 0.0196	0.7200 ± 0.0135	$\mathbf{0.7113 \pm 0.0174}$	0.6033 ± 0.0122	0.6213 ± 0.0194	$\mathbf{0.5900 \pm 0.0367}$
Madelon Sample	0.00	$\mathbf{0.5364 \pm 0.0537}$	0.3246 ± 0.0061	$\mathbf{0.6833 \pm 0.0273}$	0.2661 ± 0.0101	$\mathbf{0.4907 \pm 0.0259}$	0.2391 ± 0.0167
	0.50	0.3863 ± 0.0487	0.3246 ± 0.0061	$\mathbf{0.5101 \pm 0.0115}$	0.2661 ± 0.0101	0.2739 ± 0.0104	0.2391 ± 0.0167

4.3.1 OutBoost.B1

The mean and sample standard deviation of the accuracy of each OutBoost.B1 and base learner combination applied to each of the selected data sets, are presented in Table 4.5. Additionally, the mean and sample standard deviation of the precision and recall for the minority class are contained in the same table.

The instances in bold demonstrated higher performance than AdaBoost.M1 and the associated base learner, while the italicized instances demonstrated higher performance than AdaBoost.M1 but not the associated base learner. The instances in bold with an asterisk exhibited higher performance than AdaBoost.M1 and its associated base learner when AdaBoost.M1 failed to produce higher performance metrics than its base learner.

Table 4.6 reveals eight instances of OutBoost.B1 exhibiting higher precision than AdaBoost.M1 and its associated base learner, where we observed a greater than 0.01 difference between OutBoost.B1 and AdaBoost.M1 for four of the instances, indicating a substantial difference. Figure 4.6 compares the precision of OutBoost.B1, AdaBoost.M1, and its associated base learners and reinforces this suggestion. Of particular interest, we noted that there were numerous cases where OutBoost.B1 was applied to the Cancer data set and produced precision for the minority class that was higher than any of the evaluated base

Table 4.5: Mean and sample standard deviation of the selected performance metrics for OutBoost.B1.

OutBoost.B1: Accuracy (Percent)

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
heart	0.00	75.99 ± 2.72	*83.50 ± 0.44*	81.65 ± 2.02	83.63 ± 0.63	*83.37 ± 0.28*	**81.07 ± 0.77**
	0.50	68.76 ± 2.85	81.98 ± 0.77	82.06 ± 0.72	83.63 ± 0.63	83.30 ± 0.37	78.76 ± 1.17
Madelon	0.00	*63.66 ± 0.47*	*58.66 ± 0.34*	50.24 ± 0.60	53.01 ± 0.91	53.46 ± 0.95	59.20 ± 0.36
	0.50	**68.86 ± 1.13***	54.60 ± 0.87	**61.74 ± 0.70**	53.01 ± 0.91	53.40 ± 0.45	59.20 ± 0.36
Cancer	0.00	**96.68 ± 0.27**	94.25 ± 0.41	69.59 ± 6.14	96.62 ± 0.13	96.54 ± 0.16	94.42 ± 0.83
	0.50	92.08 ± 0.98	95.42 ± 0.30	92.65 ± 1.00	96.62 ± 0.13	96.54 ± 0.16	**95.62 ± 0.58**
Adult	0.00	78.14 ± 2.02	83.48 ± 0.02	83.87 ± 0.15	85.07 ± 0.02	*85.06 ± 0.04*	**83.98 ± 0.14**
	0.50	82.43 ± 0.39	83.48 ± 0.02	**85.45 ± 0.17**	85.07 ± 0.02	85.05 ± 0.03	82.55 ± 0.61
Corot	0.00	98.55 ± 0.00	98.53 ± 0.02	*98.55 ± 0.01*	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
	0.50	97.98 ± 0.15	98.53 ± 0.02	98.07 ± 0.04	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00

OutBoost.B1: Precision

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	0.00	**0.9488 ± 0.0061**	**0.9615 ± 0.0044**	0.6527 ± 0.0983	0.9489 ± 0.0044	0.9504 ± 0.0027	**0.9619 ± 0.0055**
	0.50	0.9010 ± 0.0109	**0.9489 ± 0.0048**	**0.9507 ± 0.0047**	0.9489 ± 0.0044	0.9504 ± 0.0027	**0.9420 ± 0.0086**
Adult	0.00	0.6590 ± 0.1230	0.7153 ± 0.0007	0.6364 ± 0.0095	0.7450 ± 0.0006	*0.7380 ± 0.0012*	0.6951 ± 0.0119
	0.50	0.6714 ± 0.0097	0.7153 ± 0.0007	**0.6893 ± 0.0097**	0.7450 ± 0.0006	0.7376 ± 0.0007	0.7039 ± 0.0198
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
	0.50	0.1304 ± 0.0901	0.0000 ± 0.0000	**0.0907 ± 0.0226**	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

OutBoost.B1: Recall

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	0.00	0.9577 ± 0.0017	0.8697 ± 0.0096	0.8544 ± 0.0900	0.9561 ± 0.0036	0.9519 ± 0.0046	0.8746 ± 0.0209
	0.50	0.8681 ± 0.0197	0.9187 ± 0.0081	0.8306 ± 0.0295	0.9561 ± 0.0036	0.9519 ± 0.0046	0.9336 ± 0.0126
Adult	0.00	0.3999 ± 0.0711	0.5217 ± 0.0007	*0.7860 ± 0.0170*	0.5778 ± 0.0017	0.5888 ± 0.0008	**0.6140 ± 0.0213**
	0.50	0.5304 ± 0.0098	0.5217 ± 0.0007	0.7287 ± 0.0162	0.5778 ± 0.0017	0.5891 ± 0.0013	0.5456 ± 0.0543
Corot	0.00	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
	0.50	0.0144 ± 0.0097	0.0000 ± 0.0000	0.0357 ± 0.0033	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

learners or AdaBoost.M1. Not only do these results imply that OutBoost.B1 can produce higher precision than AdaBoost.M1 when AdaBoost.M1 produces higher precision than its base learner, it can also produce higher precision then all of the base learners and AdaBoost.M1 and base learner combinations.

Our results suggest that OutBoost.B1 may be useful for an imbalanced classification scenario where we emphasize increasing the precision over increasing the recall. Particularly, OutBoost.B1 may be useful in machine-learning scenarios where FP are heavily punished and TP are heavily rewarded. Consequently, OutBoost.B1 may be useful when faced with an outlier detection scenario where finding outliers that we are certain are outliers is more important than finding all of the possible outliers.

Additionally, it was observed that when the precision increased for both cutoffs, a cutoff of 0.0 produced higher precision when compared to a cutoff of 0.5. However, we also observed several instances where a cutoff of 0.5 demonstrated higher precision than AdaBoost.M1, when a cutoff $\alpha = 0.00$ did not. Regardless, these results suggest that a cutoff of 0.0 may be recommended.

Comparing the accuracy of OutBoost.B1 to the accuracy produced by AdaBoost.M1, revealed seven instances of OutBoost.B1 exhibiting higher accuracy than AdaBoost.M1 and its associated base learner. Of these seven observations we observed a greater than 1.00% difference for three of the instances,

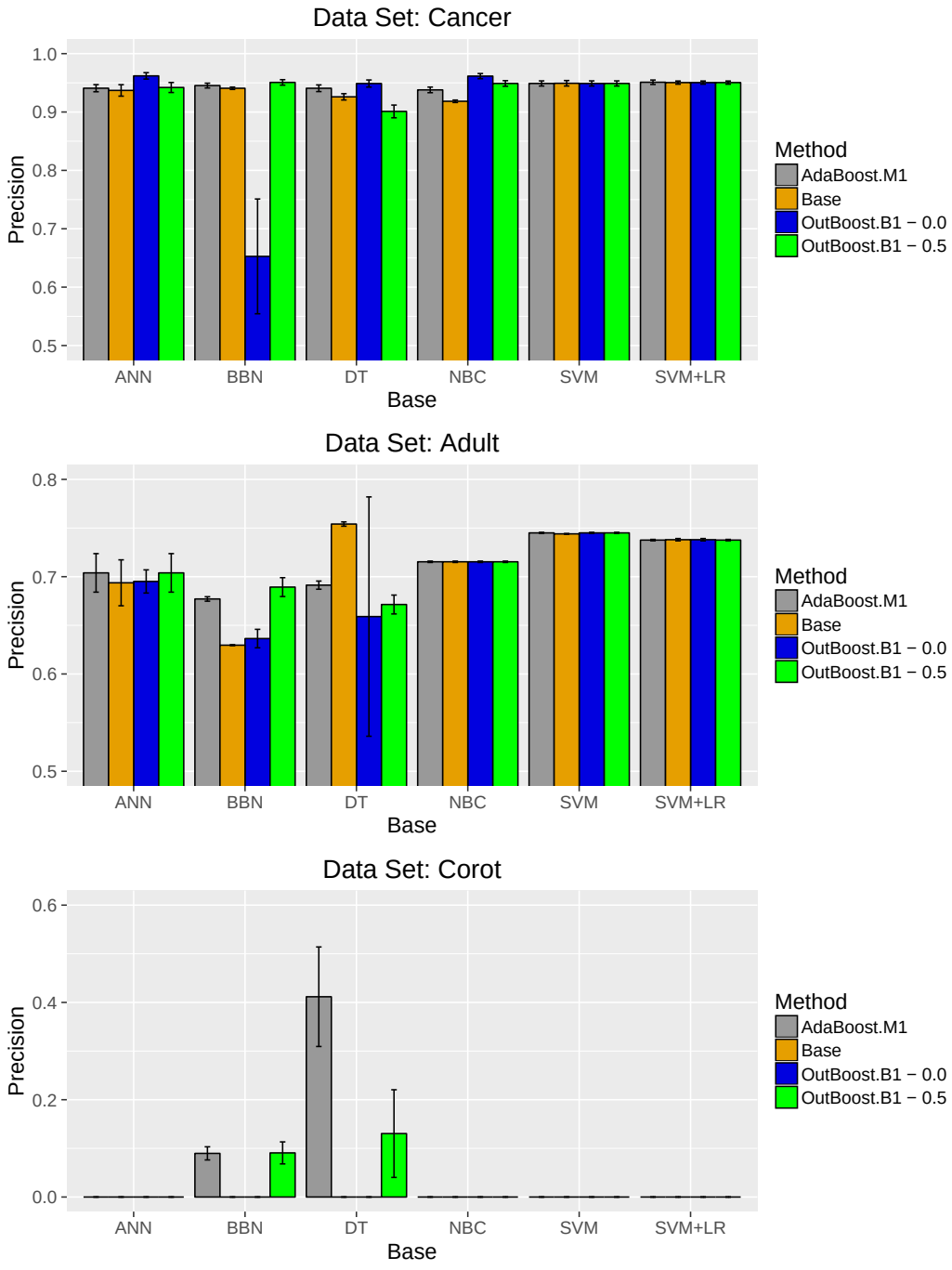


Figure 4.6: Comparison of the mean precision of OutBoost.B1, AdaBoost.M1, and its associated base learner for each data set. The number of examples where OutBoost.B1 outperforms the base learner and AdaBoost.M1 imply that OutBoost.B1 is beneficial in increasing the recall of the base learner.

indicating a considerable difference in the accuracy for at least three of the examples. Not only do these results suggest that OutBoost.B1 may produce higher accuracy than AdaBoost.M1 using the same base learner, but that OutBoost.B1 may be more accurate than AdaBoost.M1 when AdaBoost.M1 is more accurate than its associated base learner.

Figure 4.7 compares the accuracy of OutBoost.B1, AdaBoost.M1 and the associated base learner, where both cutoffs of 0.0 and 0.5 are represented. The number of examples where OutBoost.B1 produced an improvement in accuracy over AdaBoost.M1 and its associated base learner, is similar to the number of examples where AdaBoost.M1 produced higher accuracy than its base learner. Even though OutBoost.B1 produced mixed results in terms of accuracy, it produced similar mixed results to AdaBoost.M1, while at the same time exceeding the performance of AdaBoost in certain situations, which bodes well for the applications of OutBoost.B1.

The results displayed in Figure 4.7 also suggest that OutBoost.B1 may be more robust in certain situations where AdaBoost.M1 suffers in terms of accuracy, with OutBoost.B1 producing accuracy that was greater than the accuracy of AdaBoost.M1 when there was a noted decline in the accuracy of AdaBoost.M1. Seven examples of OutBoost.B1 producing higher accuracy than AdaBoost.M1 when AdaBoost.M1 failed to produce higher accuracy than

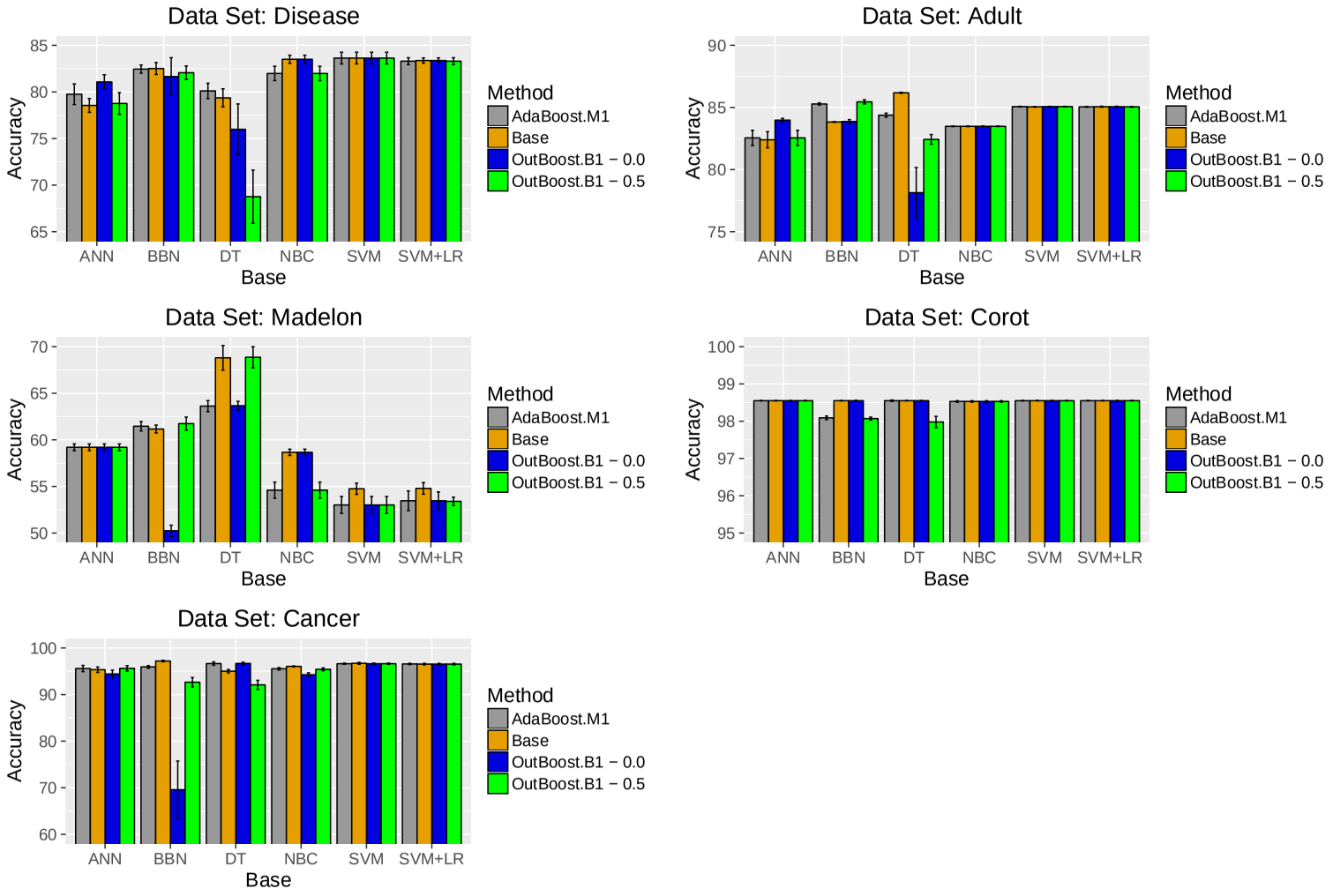


Figure 4.7: Comparison of the mean accuracy of OutBoost.B1, AdaBoost.M1 and its associated base learner for each data set, where the number of examples where OutBoost.B1 outperforms the base learner and AdaBoost.M1, imply that OutBoost.B1 is beneficial in increasing the accuracy of the base learner.

the associated base learner were observed, where we witnessed a greater than 1.00% difference in three of the instances, indicating a substantial difference. Additionally, we observed many cases where a cutoff of 0.0 produced identical accuracy as the base learner. We explore this specific scenario and the possible mechanism behind it in Section 4.3.2, during our evaluation of OutBoost.B1Man.

One example of OutBoost.B1 produced higher accuracy than AdaBoost.M1 and the base learner when AdaBoost.M1 failed to produce higher accuracy than its base learner. Although it produced a greater than 1.00% difference between OutBoost.B1 and AdaBoost.M1 we also observed a less than 1.00% difference between OutBoost.B1 and the base learner. This indicates that although increased accuracy is observed, it may not be consequential.

We also observed that using a DT as a base learner in combination with OutBoost.B1 produced low accuracy for six instances, compared to the accuracy of AdaBoost.M1 and its associated base learner. Additionally, we observed that using a BBN as a base learner with OutBoost.B1 produced a similar decline in accuracy for three examples. We noted a similar trend in the precision of OutBoost.B1 when using a BBN or DT as a base learner. This is most likely correlated to the drop in accuracy, as the dip in precision occurs where the dip in accuracy was observed with the same base learners. This de-

cline in accuracy and the potential reasons behind it are further investigated in Section 4.3.2.

4.3.2 OutBoost.B1Man

OutBoost.B1Man was applied to the Disease and Cancer data sets with varying identical thresholds ($\tau_0 = \tau_1$), in combination with the base learners listed in Table 3.2. Threshold values ranging from 0.00 to 0.50 at 0.05 intervals were chosen in order to explore the effect of varying the threshold on the mean accuracy for both data sets, and the mean precision and recall of the minority class for the imbalanced data sets. The tabular results presented in this section follow the same formating conventions introduced in Section 4.3.1.

In order to make comparisons, the mean and sample standard deviation of the accuracy was computed for each AdaBoost.M1 and base learner combination that was applied to the Disease and Cancer data sets, where the results are available in Table 1 of the Appendix. Reported on the same table are the mean and sample standard deviation of the precision and recall of the minority class for the Cancer data set.

The mean and sample standard deviation of the accuracy of OutBoost.B1-Man for the Disease data set are presented in Table 4.6 and the mean and sample standard deviation of accuracy, precision and recall for the Cancer data

Table 4.6: Mean and sample standard deviation of the accuracy for OutBoost.B1Man on the Disease data set. Various identical class-specific thresholds were selected, ranging from 0.00 to 0.50, at .05 intervals.

OutBoost.B1Man: Heart Disease Accuracy

τ	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
0.00	79.36 ± 0.97	*83.50 ± 0.44*	*82.51 ± 0.64*	83.57 ± 0.62	*83.37 ± 0.28*	**80.62 ± 0.99**
0.05	77.65 ± 1.09	81.79 ± 1.25	79.76 ± 1.62	83.57 ± 0.62	**84.03 ± 0.68***	78.29 ± 1.22
0.10	78.40 ± 1.04	81.19 ± 0.93	75.41 ± 1.93	83.57 ± 0.62	83.11 ± 0.93	79.35 ± 2.17
0.15	78.90 ± 1.37	81.52 ± 0.60	72.68 ± 2.88	83.57 ± 0.62	78.31 ± 0.65	78.70 ± 1.96
0.20	78.70 ± 1.66	80.99 ± 0.98	69.07 ± 2.30	83.57 ± 0.62	77.62 ± 0.69	78.71 ± 1.60
0.25	**79.90 ± 1.21**	*82.18 ± 0.66*	73.07 ± 1.85	83.57 ± 0.62	78.22 ± 1.09	78.57 ± 0.79
0.30	78.24 ± 1.69	81.72 ± 1.11	73.21 ± 2.01	83.57 ± 0.62	79.69 ± 1.69	78.63 ± 1.34
0.35	78.38 ± 1.26	81.19 ± 1.34	78.41 ± 1.25	83.57 ± 0.62	82.19 ± 1.23	79.35 ± 0.65
0.40	79.04 ± 1.76	80.46 ± 1.45	78.53 ± 1.61	83.57 ± 0.62	82.91 ± 0.35	**79.42 ± 0.37**
0.45	78.84 ± 1.80	*82.53 ± 0.98*	80.73 ± 0.70	83.57 ± 0.62	*83.30 ± 0.36*	79.09 ± 1.43
0.50	79.37 ± 1.08	82.05 ± 0.57	82.45 ± 0.44	83.57 ± 0.62	83.04 ± 0.18	79.36 ± 1.51

Table 4.7: Mean and sample standard deviation of the accuracy for OutBoost.B1Man on the breast Cancer data set. Various identical class-specific thresholds were selected, ranging from 0.00 to 0.50, at .05 intervals.

OutBoost.B1Man: Breast Cancer Accuracy (Percent)

τ	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
0.00	95.02 ± 0.36	*96.05 ± 0.08*	*97.20 ± 0.16*	96.65 ± 0.13	96.54 ± 0.16	95.62 ± 0.69
0.05	95.79 ± 0.48	95.34 ± 0.63	92.08 ± 0.39	96.65 ± 0.13	93.74 ± 0.31	95.39 ± 0.67
0.10	**96.08 ± 0.33**	*95.59 ± 0.27*	93.65 ± 0.96	96.65 ± 0.13	91.59 ± 1.11	**95.65 ± 0.72**
0.15	**96.11 ± 0.31**	*95.68 ± 0.12*	94.25 ± 0.71	96.65 ± 0.13	93.36 ± 0.72	95.51 ± 0.67
0.20	95.88 ± 0.29	*95.71 ± 0.42*	95.34 ± 0.48	96.65 ± 0.13	92.53 ± 0.77	95.57 ± 0.61
0.25	95.80 ± 0.30	*95.94 ± 0.16*	95.71 ± 0.47	96.65 ± 0.13	94.77 ± 0.77	95.45 ± 0.51
0.30	95.74 ± 0.37	*95.94 ± 0.28*	*95.88 ± 0.43*	96.65 ± 0.13	95.94 ± 0.60	95.62 ± 0.65
0.35	95.80 ± 0.24	*95.76 ± 0.45*	95.74 ± 0.49	96.65 ± 0.13	96.42 ± 0.18	95.57 ± 0.65
0.40	95.88 ± 0.37	*95.77 ± 0.36*	95.85 ± 0.34	96.65 ± 0.13	96.40 ± 0.28	**95.82 ± 0.58**
0.45	**95.97 ± 0.31**	*95.91 ± 0.16*	*96.14 ± 0.23*	96.65 ± 0.13	96.51 ± 0.13	95.62 ± 0.53
0.50	95.94 ± 0.26	95.51 ± 0.30	95.85 ± 0.29	96.65 ± 0.13	96.54 ± 0.16	95.62 ± 0.68

OutBoost.B1Man: Breast Cancer Precision

τ	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
0.00	0.9260 ± 0.0054	0.9184 ± 0.0021	0.9407 ± 0.0020	0.9490 ± 0.0045	0.9504 ± 0.0027	0.9374 ± 0.0092
0.05	0.9348 ± 0.0089	**0.9471 ± 0.0016**	0.9021 ± 0.0040	0.9490 ± 0.0045	**0.9700 ± 0.0027***	**0.9436 ± 0.0099**
0.10	0.9382 ± 0.0052	0.9307 ± 0.0062	0.8946 ± 0.0187	0.9490 ± 0.0045	**0.9694 ± 0.0122***	**0.9420 ± 0.0109**
0.15	**0.9418 ± 0.0079**	0.9315 ± 0.0047	0.9149 ± 0.0143	0.9490 ± 0.0045	**0.9703 ± 0.0098***	0.9382 ± 0.0083
0.20	0.9389 ± 0.0066	0.9271 ± 0.0046	0.9296 ± 0.0036	0.9490 ± 0.0045	**0.9550 ± 0.0107***	**0.9403 ± 0.0090**
0.25	0.9381 ± 0.0070	0.9245 ± 0.0025	0.9372 ± 0.0087	0.9490 ± 0.0045	**0.9593 ± 0.0074***	0.9369 ± 0.0066
0.30	0.9392 ± 0.0066	0.9218 ± 0.0017	0.9420 ± 0.0048	0.9490 ± 0.0045	**0.9545 ± 0.0049***	0.9390 ± 0.0083
0.35	0.9383 ± 0.0049	0.9265 ± 0.0055	0.9411 ± 0.0048	0.9490 ± 0.0045	0.9500 ± 0.0032	0.9396 ± 0.0093
0.40	0.9376 ± 0.0048	0.9341 ± 0.0033	0.9435 ± 0.0042	0.9490 ± 0.0045	0.9500 ± 0.0041	**0.9421 ± 0.0086**
0.45	0.9401 ± 0.0031	**0.9389 ± 0.0082**	**0.9449 ± 0.0072**	0.9490 ± 0.0045	0.9504 ± 0.0027	0.9396 ± 0.0044
0.50	0.9403 ± 0.0029	0.9379 ± 0.0048	0.9444 ± 0.0038	0.9490 ± 0.0045	0.9504 ± 0.0027	0.9402 ± 0.0096

OutBoost.B1Man: Breast Cancer Recall

τ	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
0.00	0.9337 ± 0.0107	*0.9744 ± 0.0019*	*0.9834 ± 0.0042*	0.9569 ± 0.0022	0.9519 ± 0.0046	**0.9385 ± 0.0140**
0.05	**0.9468 ± 0.0116**	0.9197 ± 0.0198	0.8705 ± 0.0122	0.9569 ± 0.0022	0.8456 ± 0.0090	0.9253 ± 0.0169
0.10	**0.9518 ± 0.0082**	*0.9453 ± 0.0147*	0.9311 ± 0.0095	0.9569 ± 0.0022	0.7852 ± 0.0317	0.9345 ± 0.0133
0.15	**0.9477 ± 0.0085**	*0.9471 ± 0.0078*	0.9245 ± 0.0072	0.9569 ± 0.0022	0.8350 ± 0.0237	0.9344 ± 0.0122
0.20	0.9453 ± 0.0068	*0.9528 ± 0.0131*	*0.9395 ± 0.0135*	0.9569 ± 0.0022	0.8249 ± 0.0219	0.9336 ± 0.0110
0.25	0.9436 ± 0.0127	*0.9627 ± 0.0064*	*0.9420 ± 0.0064*	0.9569 ± 0.0022	0.8879 ± 0.0206	0.9336 ± 0.0129
0.30	0.9403 ± 0.0054	*0.9661 ± 0.0091*	*0.9411 ± 0.0106*	0.9569 ± 0.0022	0.9296 ± 0.0183	**0.9370 ± 0.0154**
0.35	0.9427 ± 0.0089	*0.9562 ± 0.0097*	*0.9386 ± 0.0170*	0.9569 ± 0.0022	0.9486 ± 0.0048	0.9345 ± 0.0131
0.40	**0.9461 ± 0.0076**	*0.9471 ± 0.0128*	*0.9386 ± 0.0091*	0.9569 ± 0.0022	0.9478 ± 0.0099	**0.9394 ± 0.0125**
0.45	**0.9461 ± 0.0077**	*0.9454 ± 0.0042*	*0.9460 ± 0.0111*	0.9569 ± 0.0022	0.9511 ± 0.0034	**0.9361 ± 0.0134**
0.50	0.9453 ± 0.0099	0.9337 ± 0.0074	0.9378 ± 0.0089	0.9569 ± 0.0022	0.9519 ± 0.0046	0.9353 ± 0.0120

set are displayed in Table 4.7. An interesting pattern was discovered when we used an identical class-specific threshold of 0.00 ($\tau_0 = \tau_1 = 0.00$) and 0.50 ($\tau_0 = \tau_1 = 0.50$), where it was observed that identical class-specific thresholds of 0.00 produced the same accuracy, precision and recall as the base learner, except when combining OutBoost.B1Man with a SVM or ANN. Additionally, identical class-specific thresholds of 0.50 produced the same results as AdaBoost.M1 when we applied AdaBoost.M1 to the same data set with the same base learner.

When an identical class-specific threshold of 0.00 caused OutBoost.B1Man to produce the same performance as the base learner, it may have been the product of an absence of training examples misclassified with an estimated probability of 1.00. If this were to occur, OutBoost.B1Man would not adjust the weight of any observations, which would result in the same behavior as the base learner, given that the base learner would be provided the same training examples for each iterations, where each training example would have the same initial weight. These results may also suggest why numerous examples of OutBoost.B1 producing the same performance results as the base learner were observed, as a similar scenario may have occurred.

We expected the identical class-specific threshold of 0.50 to produce similar results as AdaBoost.M1 as OutBoost.B1Man increases the weight of training

examples that are misclassified when the estimated probability of the true class for the training example is smaller or equal to the class-specific threshold of the true class ($Pr(y = y_i|\mathbf{x}_i, D_t) \leq 0.50$). This implies an identical class-specific threshold of 0.50 would increase the weight of any misclassified observation, as for an observation to be misclassified it must have an estimated probability of the true class that is smaller than or equal to 0.5, otherwise it would not be misclassified. In other words, OutBoost.B1Man would increase the weight of the same training examples that would be re-weighted by AdaBoost.M1, producing the same results.

Comparing the precision of OutBoost.B1Man to AdaBoost.M1 revealed 14 instances of OutBoost.B1Man exhibiting higher precision than AdaBoost.M1 and its associated base learner. However, we only observed a greater than 0.01 difference between OutBoost.B1 and AdaBoost.M1 for three of the fourteen instances, indicating that most of the examples are not considerably different. We also observed six instances of OutBoost.B1Man exhibiting higher precision than AdaBoost.M1 when AdaBoost.M1 failed to produce higher precision then the base learner when the base learner was a SVM+LR. However, we observed that only three of the six instances produced precision that was greater than a .01 difference then the precision of the base learner, once more indicating that the difference is not very substantial. Regardless, given that some of

the improvements noted for the precision are substantial, we could infer that OutBoost.B1Man may have potential for improving the precision produced by a base learner much like OutBoost.B1, but allowing a researcher to preselect an optimum threshold.

Surprisingly, we observed that OutBoost.B1Man utilizing a SVM+LR base learner on the Cancer data set produced the highest precision observed for this data set in our entire evaluation. However, this may be an optimistic performance estimate given that essentially, the test set was used to select the best threshold [21]. Additionally, it was noted that choosing different identical class-specific thresholds did not affect the accuracy, precision, or recall when combining OutBoost.B1Man and a SVM. We suggest that this is likely due to SVMs not producing useful probability estimates without the addition of a LR to the output [22], as such we would not recommended using a SVM with our techniques without the addition of a LR.

In order to investigate the decline in accuracy produced by a BBN base learner in our evaluation of OutBoost.B1, OutBoost.B1Man was combined with a BBN base learner and applied at 5, 10, 15 and 20 iterations to the Cancer data set at varying identical class-specific thresholds ranging from 0.00 to 0.50 at intervals of 0.05. We computed the mean and sample standard deviation of the accuracy, presented in Table 4.8, and discovered that as the

Table 4.8: Mean and sample standard deviation of the accuracy (percentage) of OutBoost.B1Man with a BBN base learner on the Cancer data set, at different identical thresholds and boosting iterations (T).

τ	$T = 5$	$T = 10$	$T = 15$	$T = 20$
0.00	*97.20 ± 0.16*	*97.20 ± 0.16*	*97.20 ± 0.16*	*97.20 ± 0.16*
0.05	95.22 ± 0.46	92.08 ± 0.39	90.93 ± 1.09	90.59 ± 1.15
0.10	95.65 ± 0.44	93.65 ± 0.96	92.05 ± 0.89	90.99 ± 0.86
0.15	95.48 ± 0.30	94.25 ± 0.71	92.73 ± 0.74	91.33 ± 0.88
0.20	*95.77 ± 0.28*	95.34 ± 0.48	94.51 ± 0.16	93.22 ± 0.68
0.25	*96.05 ± 0.30*	95.71 ± 0.47	94.74 ± 0.25	93.85 ± 0.51
0.30	*95.97 ± 0.50*	*95.88 ± 0.43*	95.28 ± 0.56	94.68 ± 1.06
0.35	*95.77 ± 0.58*	95.74 ± 0.49	95.51 ± 0.88	95.05 ± 0.97
0.40	95.74 ± 0.25	95.85 ± 0.34	95.79 ± 0.40	95.62 ± 0.36
0.45	*95.82 ± 0.19*	*96.14 ± 0.23*	*96.11 ± 0.19*	95.91 ± 0.26
0.50	95.74 ± 0.43	95.85 ± 0.29	95.94 ± 0.28	95.94 ± 0.28

number of iterations increased the number of examples of declining accuracy increased.

The accuracy decreased when compared to the performance of AdaBoost.M-1, at five iterations for identical class-specific thresholds ranging from 0.05 to 0.15 and at 10 iterations for identical class-specific thresholds ranging from 0.05 to 0.25. Additionally, the accuracy declined at 15 iterations for identical class-specific thresholds ranging 0.05 to 0.40 and at 20 iterations for identical class-specific threshold from ranging from 0.05 to 0.45. These results indicate that the accuracy of OutBoost.B1Man appears to rely on the class-specific thresholds selected and the number of boosting iterations performed. We also recognized that the accuracy decreased further as the number of iterations

increased.

Given these results, OutBoost.B1Man with a BBN base learner may be susceptible to over-fitting depending upon the number of iterations and the class-specific thresholds selected, or it may be sensitive to the choice of class specific thresholds and the number of iterations executed. These results may help explain why OutBoost.B1 experienced a declining accuracy when combined with a BBN base learner.

4.3.3 OutBoost.B2

The mean and sample standard deviation of the accuracy was computed for each of the OutBoost.B2 and base learner combinations applied to each of the selected data sets and are presented in Table 4.9. Additionally provided in Table 4.9 are the mean and sample standard deviation of the precision and recall for minority class of the selected imbalanced data sets, where the results follow the same formating convention introduced in Section 4.3.1.

Referring to Table 4.9, comparisons were made between the recall of OutBoost.B2 to AdaBoost.M1, where seven instances of OutBoost.B2 exhibiting higher recall than AdaBoost.M1 and its associated base learner were revealed. We observed a greater than 0.01 difference between OutBoost.B2 and AdaBoost.M1 for six of the seven instances, suggesting a considerable difference

Table 4.9: Mean and sample standard deviation of the selected performance metrics for OutBoost.B2.

OutBoost.B2: Accuracy (Percent)

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Disease	0.00	66.46 ± 2.92	*82.05 ± 0.70*	79.55 ± 1.16	83.63 ± 0.63	83.30 ± 0.37	79.48 ± 1.47
	0.50	69.96 ± 3.42	*83.50 ± 0.44*	77.25 ± 2.26	83.63 ± 0.63	83.30 ± 0.37	$\mathbf{80.81 \pm 0.85}$
Madelon	0.00	56.98 ± 4.23	54.60 ± 0.87	60.33 ± 1.14	*54.75 ± 0.60*	*53.97 ± 0.88*	59.19 ± 0.31
	0.50	60.86 ± 0.82	*58.66 ± 0.34*	54.79 ± 1.57	*54.75 ± 0.60*	*53.92 ± 0.71*	$\mathbf{59.42 \pm 0.68}$*
cancer	0.00	95.25 ± 0.99	95.45 ± 0.18	*96.34 ± 0.50*	*96.71 ± 0.20*	96.57 ± 0.14	95.45 ± 0.36
	0.50	84.66 ± 3.16	95.31 ± 0.33	*96.22 ± 0.75*	*96.71 ± 0.20*	96.54 ± 0.16	95.48 ± 0.96
Adult	0.00	82.09 ± 0.23	83.48 ± 0.02	$\mathbf{85.36 \pm 0.14}$	85.05 ± 0.02	85.05 ± 0.03	82.46 ± 0.58
	0.50	68.77 ± 5.63	83.48 ± 0.02	84.26 ± 0.08	85.05 ± 0.02	*85.06 ± 0.04*	$\mathbf{83.98 \pm 0.14}$
Corot	0.00	95.56 ± 0.06	98.53 ± 0.02	82.61 ± 3.70	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00
	0.50	98.55 ± 0.00	98.53 ± 0.02	*98.55 ± 0.00*	98.55 ± 0.00	98.55 ± 0.00	98.55 ± 0.00

OutBoost.B2: Precision

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	0.00	0.9185 ± 0.0130	0.8932 ± 0.0055	0.9257 ± 0.0075	*0.9492 ± 0.0047*	0.9510 ± 0.0038	0.9378 ± 0.0076
	0.50	0.8215 ± 0.0502	0.9127 ± 0.0038	0.9253 ± 0.0149	*0.9492 ± 0.0047*	0.9504 ± 0.0027	0.9367 ± 0.0090
Adult	0.00	0.6208 ± 0.0054	0.7153 ± 0.0007	$\mathbf{0.6807 \pm 0.0069}$	0.7440 ± 0.0004	0.7375 ± 0.0007	$\mathbf{0.7146 \pm 0.0309}$
	0.50	0.4869 ± 0.0301	0.7153 ± 0.0007	0.6441 ± 0.0026	0.7440 ± 0.0004	*0.7380 ± 0.0012*	0.6951 ± 0.0119
Corot	0.00	0.0282 ± 0.0040	0.0000 ± 0.0000	0.0327 ± 0.0086	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

OutBoost.B2: Recall

Data	α	DT $T = 1000$	NBC $T = 1000$	BBN $T = 1000$	SVM $T = 5$	SVM+LR $T = 5$	ANN $T = 5$
Cancer	0.00	0.9493 ± 0.0182	$\mathbf{0.9901 \pm 0.0021}$*	*0.9751 ± 0.0106*	*0.9586 ± 0.0031*	0.9519 ± 0.0046	0.9328 ± 0.0046
	0.50	0.8565 ± 0.0319	*0.9577 ± 0.0131*	*0.9751 ± 0.0028*	*0.9586 ± 0.0031*	0.9519 ± 0.0046	$\mathbf{0.9353 \pm 0.0229}$
Adult	0.00	$\mathbf{0.6599 \pm 0.0108}$	0.5217 ± 0.0007	0.7443 ± 0.0118	*0.5783 ± 0.0012*	0.5891 ± 0.0013	0.5229 ± 0.0690
	0.50	$\mathbf{0.7652 \pm 0.0499}$	0.5217 ± 0.0007	*0.7776 ± 0.0036*	*0.5783 ± 0.0012*	0.5888 ± 0.0008	$\mathbf{0.6140 \pm 0.0213}$
Corot	0.00	$\mathbf{0.0623 \pm 0.0088}$	0.0000 ± 0.0000	$\mathbf{0.1982 \pm 0.0510}$	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
	0.50	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

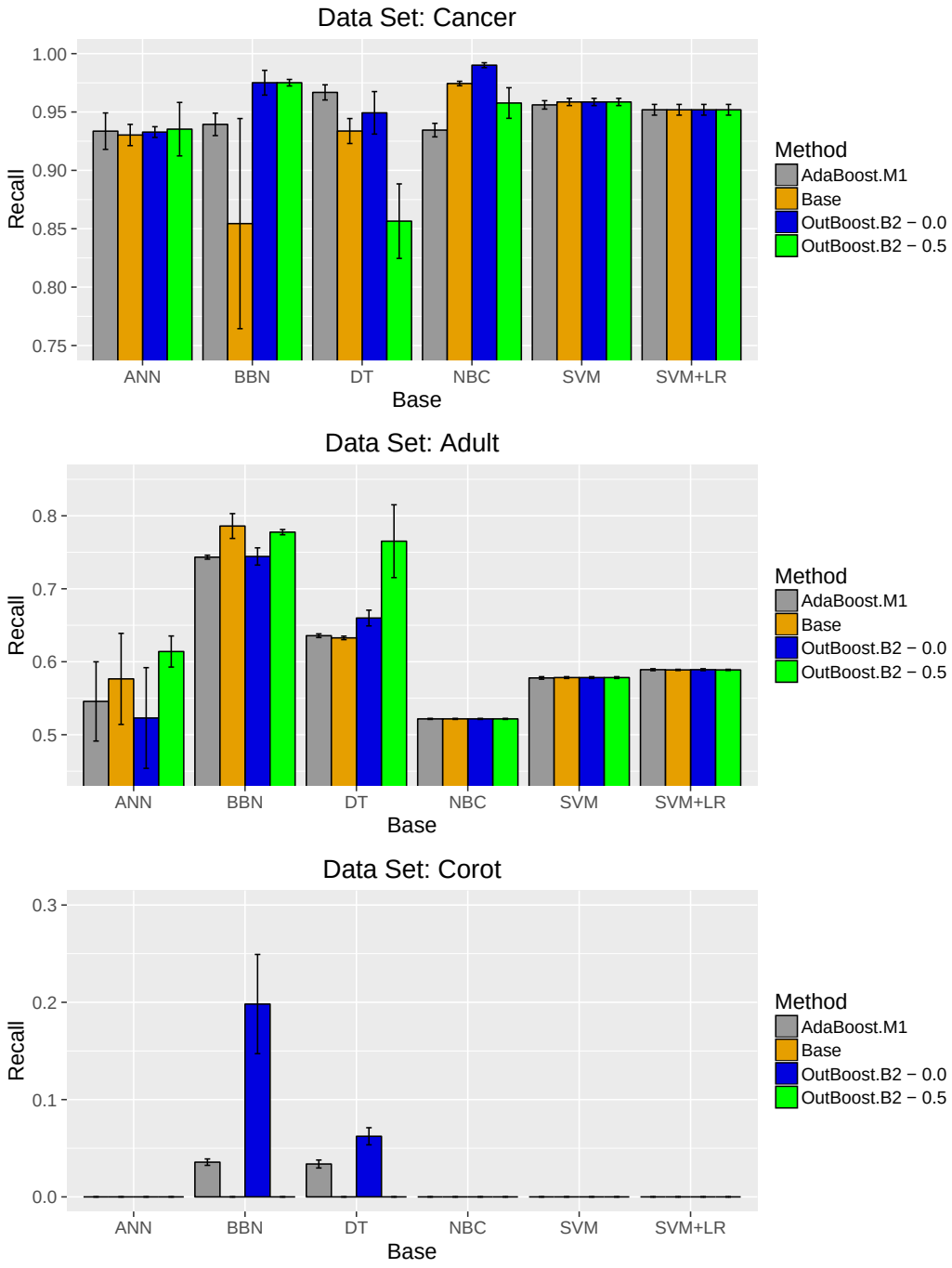


Figure 4.8: Comparison of the mean recall of OutBoost.B2, AdaBoost.M1, and its associated base learner for each data set, where we observed examples of OutBoost.B2 producing higher recall in the minority class than the base learner or AdaBoost.M1.

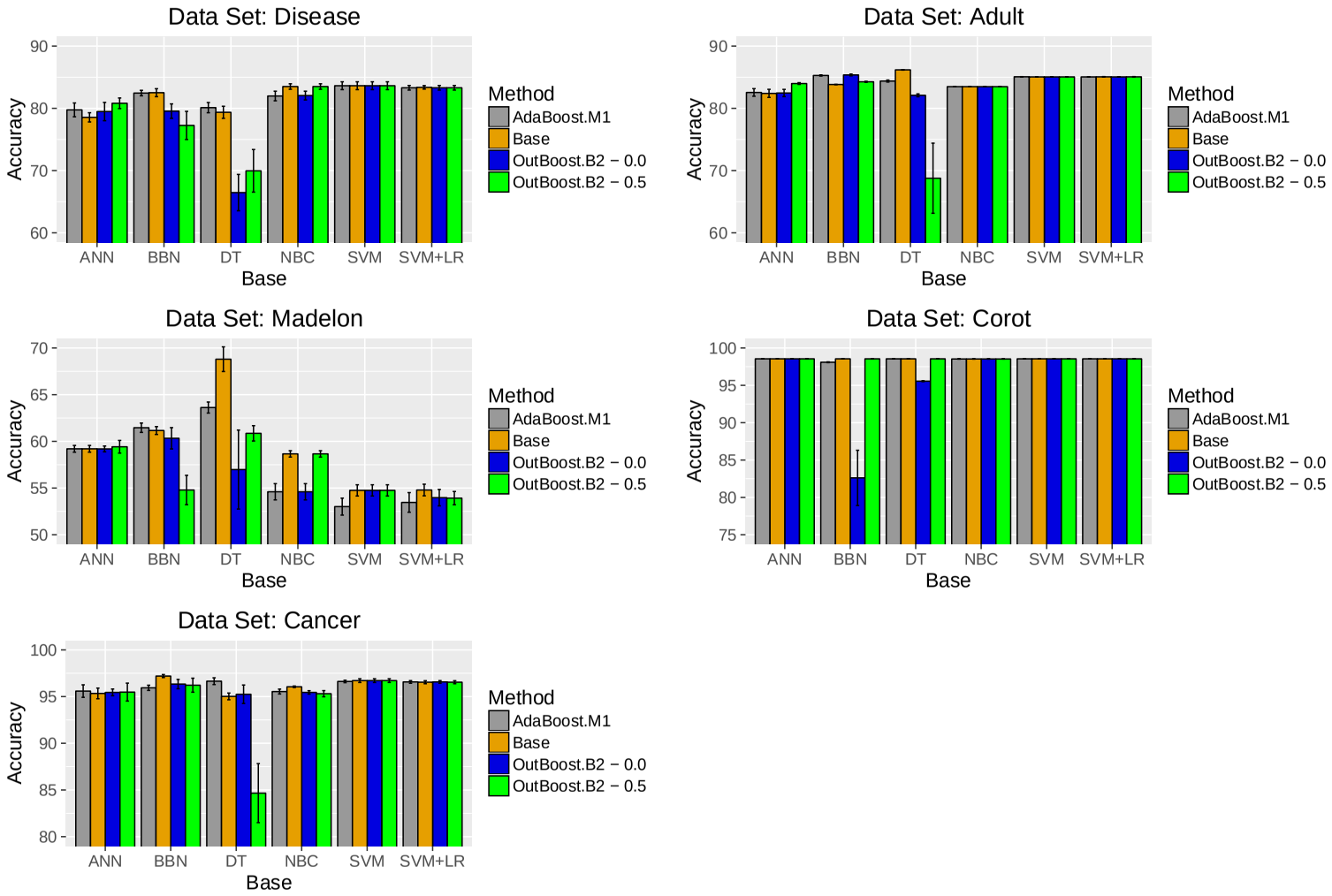


Figure 4.9: Comparison of the mean accuracy of OutBoost.B2, AdaBoost.M1, and its associated base learner for each data set. The few observed examples of OutBoost.B2 exhibiting higher accuracy than the base learner and AdaBoost.M1, indicate that OutBoost.B2 may have potential in terms of increasing the accuracy of the base learner.

in recall for most of the observed improvements. Figure 4.8 supports these implications by comparing the recall of OutBoost.B1, AdaBoost.M1, and its associated base learners, revealing numerous examples of substantial improvements.

We observed examples where OutBoost.B2 was applied to the Cancer and Corot data set and produced recall of the minority class that was higher then any of the evaluated base learners or AdaBoost.M1. Not only do these results imply that OutBoost.B2 can produce higher recall than AdaBoost.M1 when AdaBoost.M1 produces higher recall than its base learner, it can also produce higher recall then all of the base learners and AdaBoost.M1 and base learner combinations.

The results displayed in Figure 4.8 also suggest that OutBoost.B1 may be more robust in certain situations where AdaBoost.M1 suffers in terms of recall, where nine examples were noted where OutBoost.B2 produced higher recall than AdaBoost.M1 when AdaBoost.M1 failed to produce higher recall than its associated base learner. Additionally, a greater than 0.01% difference was witnessed for five of the eight instances, suggesting a substantial difference once again.

Overall, our results suggest that OutBoost.B2 may be useful for an imbalanced classification scenario where we emphasize increasing the recall over

increasing the precision. Particularly, OutBoost.B2 may be useful in machine-learning scenarios where TP are heavily rewarded and FP are not as damaging. Consequently, OutBoost.B2 may be useful when faced with an outlier detection scenario where finding the highest number of outliers possible, within reason, is more important than making certain all of the discovered outliers are true outliers.

Comparing the accuracy of OutBoost.B2 to AdaBoost.M1 revealed four instances of OutBoost.B2 exhibiting higher accuracy than AdaBoost.M1 and its associated base learner. However, we only observed a greater than 1.00% difference between OutBoost.B2 and AdaBoost.M1 for two of the four instances, suggesting that two of these observations may not have produced a meaningful difference. Regardless, these results suggest that OutBoost.B2 can produce accuracy that is higher than AdaBoost.M1 and its associated base learner, although with less frequency than OutBoost.B1.

Figure 4.9 compares the accuracy of OutBoost.B1, AdaBoost.M1 and the associated base learner, where both cutoffs of 0.0 and 0.5 are included. Like OutBoost.B1, using a DT as a base learner produced a number of examples of lower accuracy when compared to AdaBoost.M1 and its associated base learner. Additionally, we observed that using a BBN as a base learner with OutBoost.B2 produced a similar decline in accuracy for a number of examples,

once again comparable to OutBoost.B1. We suggest that this is likely due to the same underlying mechanism that produced this behavior in OutBoost.B1, which was explored further in Section 4.3.2.

We observed 14 examples of OutBoost.B2 producing higher accuracy than AdaBoost.M1 when AdaBoost.M1 failed to produce higher accuracy than the associated base learner. However, we only observed a greater than 1.00% difference between OutBoost.B2 and AdaBoost.M1 for four of the 14 instances. Our results imply that OutBoost.B2 may be more robust in certain situations where AdaBoost.M1 suffers in terms of accuracy.

Referring to Figure 4.9, we observe that choosing a cutoff of 0.5 ($\alpha = 0.50$) caused the above scenario to occur more frequently than choosing a cutoff of 0.0 ($\alpha = 0.00$). Additionally, there were many cases where a cutoff of 0.5 produced identical accuracy as the base learner. It is likely that these results are related to the phenomenon that was observed with OutBoost.B1 and was explored in Section 4.3.2. However, selecting a cutoff of 0.5 rather than 0.0 produced identical results to the base learner, due to the differences in OutBoost.B2 and which training examples would have their weights increased.

We also noted one example of OutBoost.B2 producing higher accuracy than AdaBoost.M1 and the base learner when AdaBoost.M1 failed to produce higher accuracy than its base learner. However, this instance did not produce

a greater than 1.00% difference between OutBoost.B2 and AdaBoost.M1 or the associated base leaner, raising suspicions regarding how consequential this result may be.

Chapter 5

Conclusions

Machine learning may be described as a group of methods used to automatically detect patterns in data in order to make decisions when dealing with uncertainty [22]. Supervised machine learning focuses on taking a set of inputs and outputs and given the input-output pairs makes predictions regarding previously unseen inputs [22]. In other words, if we can accurately predict an output given a set of inputs we can, in essence, predict the future. Consequently, this enables decision making that can be an asset to scientific, commercial, industrial, and medical fields.

Developing new and innovative ways of making predictions is key to the continued advancement of supervised machine learning, which is why we choose to investigate how the performance of machine-learning methods could be im-

pacted by combining those methods with outlier detection. Outlier analysis can be described as the study of detecting, exploring, and reacting to outliers in an effort to gain insight regarding the data that contains the outliers [2–4]. Given that machine learning and outlier analysis both share the same goal of gaining insight from data we felt it was only natural to combine the two fields in order to better enable machine-learning techniques to make predictions.

We set out in this thesis to discover if we could combine univariate outlier detection and machine-learning techniques in ways that could improve the performance of the machine-learning techniques involved. A framework was constructed where the class probability estimates could be examined for outliers, where discovered outliers could be used to construct class-specific thresholds. We then proposed two techniques, with a number of different variations, to employee this framework. Our evaluation was tailored to include the primary objective of increasing the accuracy of the selected machine learning methods, while our secondary objective was to increase the precision or recall of the minority class in an imbalanced data set.

We obtained mixed results for our proposed techniques, where the various techniques demonstrated gains and losses depending on the technique, base learner, data set, and metric observed. This could interpreted as discouraging, however a similar behavior is noted in the performance of AdaBoost.M1,

which is a well used and adopted technique. Although we did not observe overly compelling results for each technique with each base learner and data set combination, we should keep in mind the *No Free Lunch Theorem*: all models are wrong, however, some are useful [22]. In other words, no model will work well for all problems [22].

5.1 General Conclusions

Given the evaluation of our proposed techniques for utilizing class-specific thresholds presented in Chapter 4, we are prepared to make the following conclusions based on our results. Our results suggest that:

1. GenThresh.B1 and GenThresh.B2 behave similarly for accuracy, precision, and recall.

2. GenThresh.B1 and GenThresh.B2 can produce higher recall than their associated base learner.

3. OutBoost.B1 can produce higher accuracy than its associated base learner and AdaBoost.M1 with the same base learner.

4. OutBoost.B1 can produce higher precision than its associated base learner and AdaBoost.M1 with the same base learner.

5. OutBoost.B2 can produce higher recall than its associated base learner and AdaBoost.M1 with the same base learner.

6. OutBoost.B1 and OutBoost.B2 can produce higher accuracy, precision, and recall than AdaBoost.M1 (with the same base learner) when AdaBoost.M1 fails to produce higher accuracy, precision, and recall than its base learner.

7. The choice of the class-specific threshold and the number of boosting iterations can negatively impact accuracy for OutBoost.B1Man, which suggests that OutBoost.B1 and OutBoost.B2 may be similarly affected.

In more abstract terms, our results also suggest that class-specific thresholds determined via outlier detection can be utilized to improve certain metrics for machine-learning techniques.

5.2 Future Work

Based on our work, we recommend the following research be explored in order to better understand the algorithms proposed in this thesis.

5.2.1 Number of Iterations

Boosting techniques are sensitive to the number of iterations performed and OutBoost may overfit the training data when too many iterations are performed [26]. We observed this sensitivity when we applied OutBoost.B1Man to the Cancer data set when the accuracy decreased as the number of iterations increased. An investigation into the effect of varying the iterations for a few chosen base learners and data sets would better help us understand how the choice of iterations effects the proposed boosting techniques.

5.2.2 Error Break Point

AdaBoost.M1 operates by using an error break point ($\epsilon \geq 0.5$) that halts the operation of AdaBoost.M1 when the base learner exceeds the maximum error [26]. It may be possible that the maximum error that prevents AdaBoost.M1 from increasing the performance of the base learner could be different than the maximum error preventing OutBoost.B1 from increasing the performance of the base learner. We recommend examining the effect of changing the value of the epsilon break point for OutBoost with the hope of better understanding how the break point impacts performance OutBoost.

5.2.3 Other Outlier Detection Techniques

Walsh's Outlier Test is one of a number of available non-parametric univariate outlier detection techniques. It would be interesting to exchange Walsh's Outlier Test with other non-parametric outlier detection methods and evaluating the results. Examples of techniques that could be evaluated are histogram and kernel function based methods [4].

5.2.4 Modification of Probability Values Used

We also support evaluating another approach to finding outliers in the probability estimates. Instead of applying univariate outlier detection to all of the estimated probabilities for each class we could instead apply univariate outlier detection to those observations for each class where the observations truly belong to that class.

5.2.5 Simplification and Multiclass Extension

Another possible research area would be to implement and evaluate a multiclass extension of OutBoost.B1 (referred to as OutBoost.M1 (Algorithm 12). OutBoost.M1 is also a simplification of OutBoost.B1 as it does not check for normal or outlying observations between the class-specific thresholds. OutBoost.M1 will perform identically to AdaBoost.M1 if outlying observations

Algorithm 12 OutBoost.M1 is a modification of AdaBoost.M1 [12] and OutBoost.B1. OutBoost.M1 can work with multiple classes and operates by training a base learner with the goal of minimalizing the weighted error at each iteration. OutBoost.M1 determines class-specific thresholds by applying a univariate outlier detection technique to the probability estimates generated by the base learner. The weight of an observation that is misclassified and whose probability estimate falls below the class-specific threshold of its true class, is increased by the weighted error. The weights of all the observations are normalized to sum to one and used as a sampling distribution for the next iteration of the base learner. OutBoost.M1 combines the results of each iteration using a weighted voting scheme in order to build a final classifier.

function OUTBOOST.M1
  for $i = 1$ **to** m **do**
    $D_1(i) = \frac{1}{m}$
  end for
  for $t = 1$ **to** T **do**
    TRAINWEAKLEARNER(D_t)
    Get weak hypothesis $h_t \colon \mathcal{X} \to \mathcal{Y}$
    $\epsilon_t = \sum_{i \colon h_t(\mathbf{x}_i) \neq y_i} D_t(i)$
    if $\epsilon_t \geq \frac{1}{2}$ **then**
      $T = t - 1$
      Exit Loop
    end if
    $\beta_t = \frac{1-\epsilon_t}{\epsilon_t}$
    for $i = 1$ **to** m **do**
      $P_i = Pr(y|\mathbf{x}_i, D_t)$
    end for
    $\tau = \text{FINDTHRESHOLDS}(P)$
    for $i = 1$ **to** m **do**

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } (h_t(\mathbf{x}_i) \neq y_i) \text{ and} \\ & (Pr(y = y_i|\mathbf{x}_i, D_t) \leq \tau_{y_i}) \\ 1 & \text{otherwise} \end{cases}$$

    end for
  end for
  $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t \colon h_t(\mathbf{x}) = y} log\beta_t$
end function

fall between the class-specific thresholds.

5.2.6 Other Boosting Algorithms

Finally, another interesting avenue of research would be an investigation of other contemporary boosting algorithms in order to determine if those boosting algorithms may be modified in a similar fashion as AdaBoost.M1.

Appendices

Table 1: Mean and sample standard deviation of the selected performance metrics for AdaBoost.M1.

AdaBoost.M1: Accuracy

Dataset	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
Disease	**79.37 ± 1.08**	82.05 ± 0.57	82.45 ± 0.44	83.57 ± 0.62	83.04 ± 0.18	**79.36 ± 1.51**
Cancer	**95.94 ± 0.26**	95.51 ± 0.30	95.85 ± 0.29	96.65 ± 0.13	96.54 ± 0.16	**95.62 ± 0.68**

AdaBoost.M1: Precision

Dataset	DT $T = 10$	NBC $T = 10$	BBN $T = 1000$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
Cancer	**0.9403 ± 0.0029**	**0.9379 ± 0.0048**	**0.9444 ± 0.0038**	0.9490 ± 0.0045	0.9504 ± 0.0027	**0.9402 ± 0.0096**

AdaBoost.M1: Recall

Dataset	DT $T = 10$	NBC $T = 10$	BBN $T = 10$	SVM $T = 10$	SVM+LR $T = 10$	ANN $T = 10$
Cancer	**0.9453 ± 0.0099**	0.9337 ± 0.0074	0.9378 ± 0.0089	0.9569 ± 0.0022	0.9519 ± 0.0046	**0.9353 ± 0.0120**

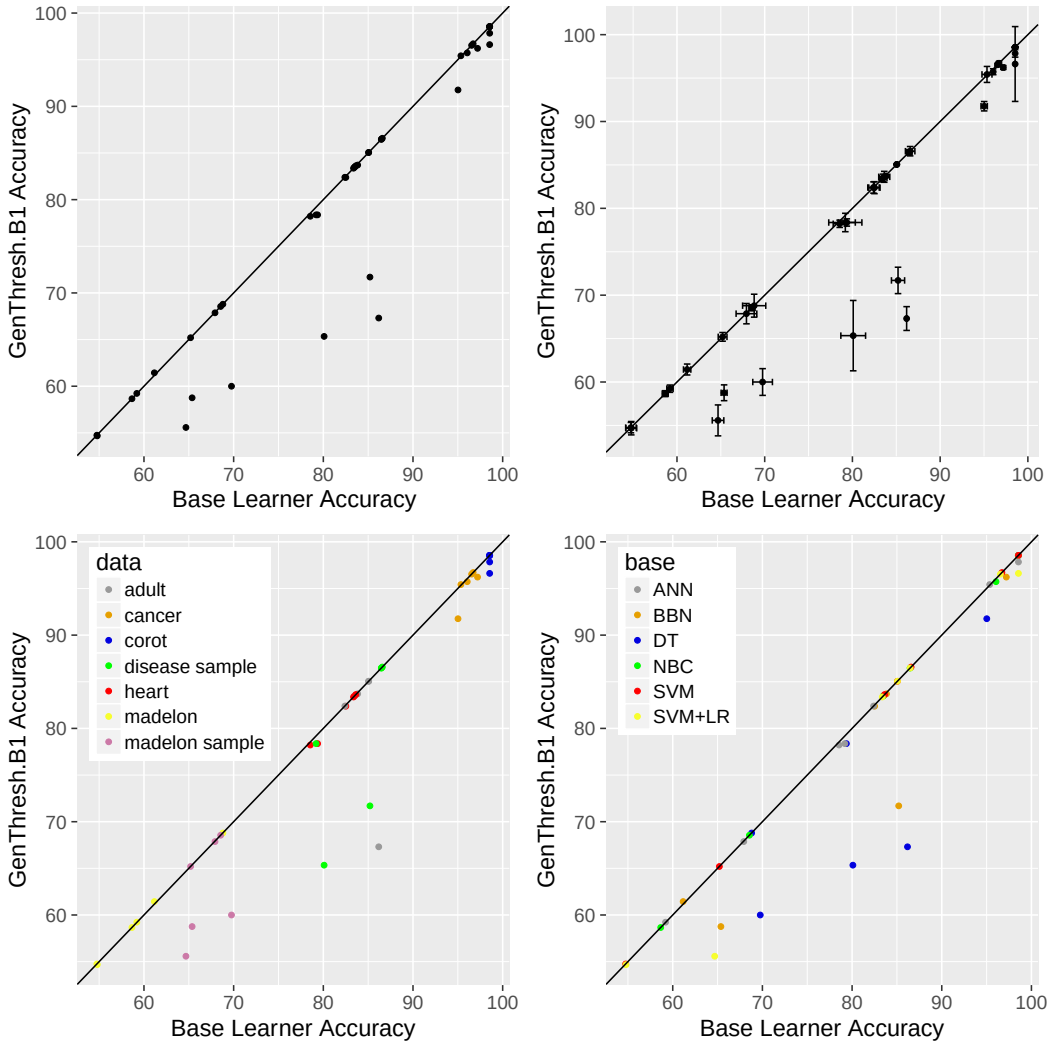


Figure 1: Comparison of the mean accuracy of GenThresh.B1 and its associated base learner, where $\alpha = 0.0$.

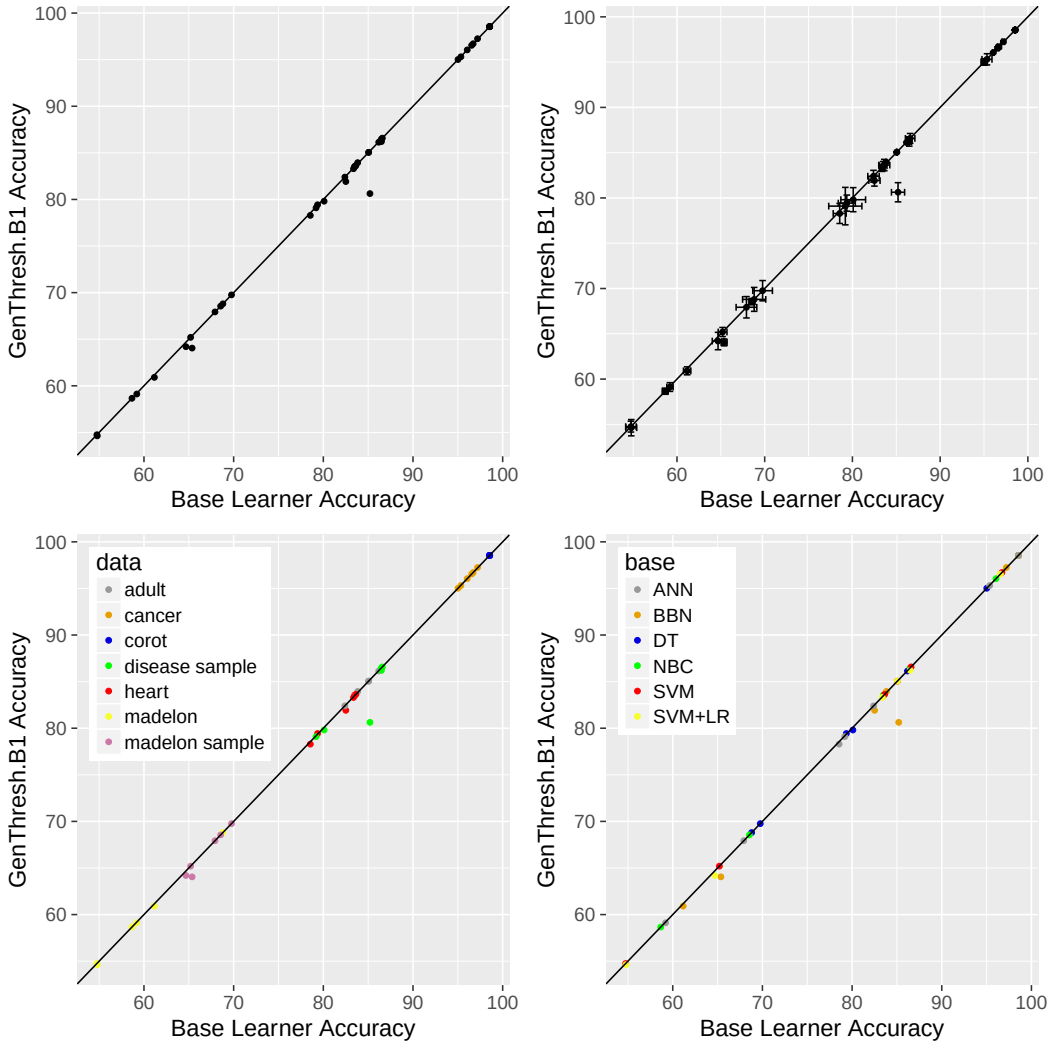


Figure 2: Comparison of the mean accuracy of GenThresh.B1 and its associated base learner, where $\alpha = 0.5$.

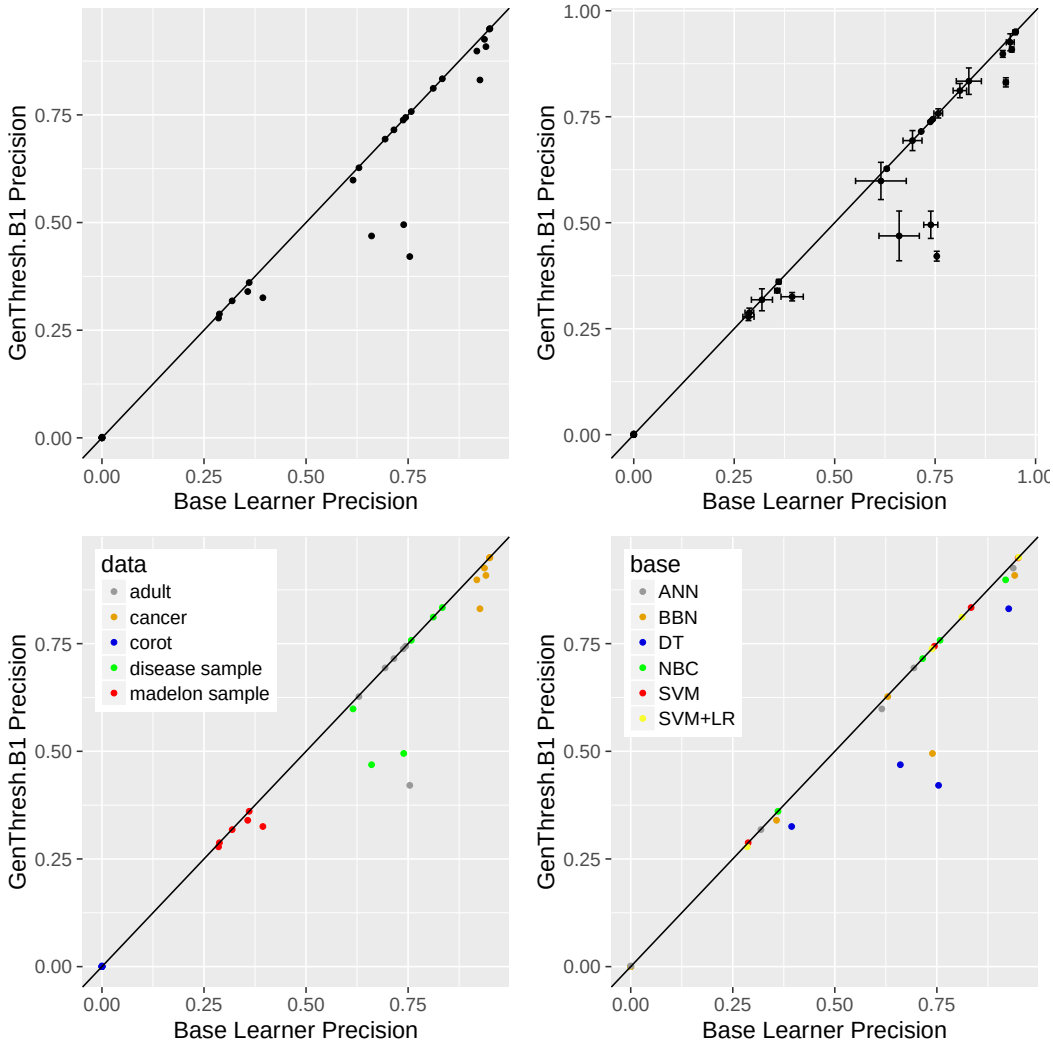


Figure 3: Comparison of the mean precision of GenThresh.B1 and its associated base learner, where $\alpha = 0.0$.

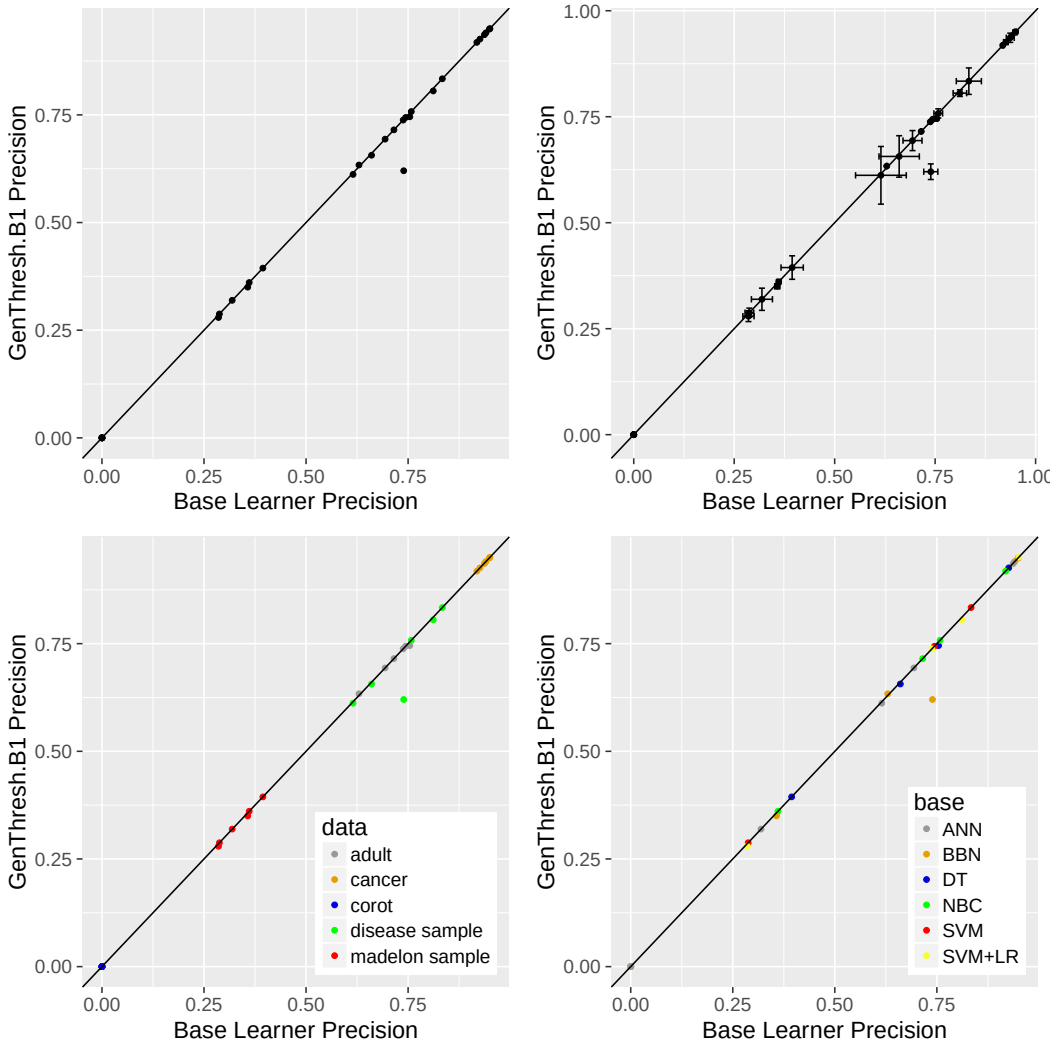


Figure 4: Comparison of the mean precision of GenThresh.B1 and its associated base learner, where $\alpha = 0.5$.

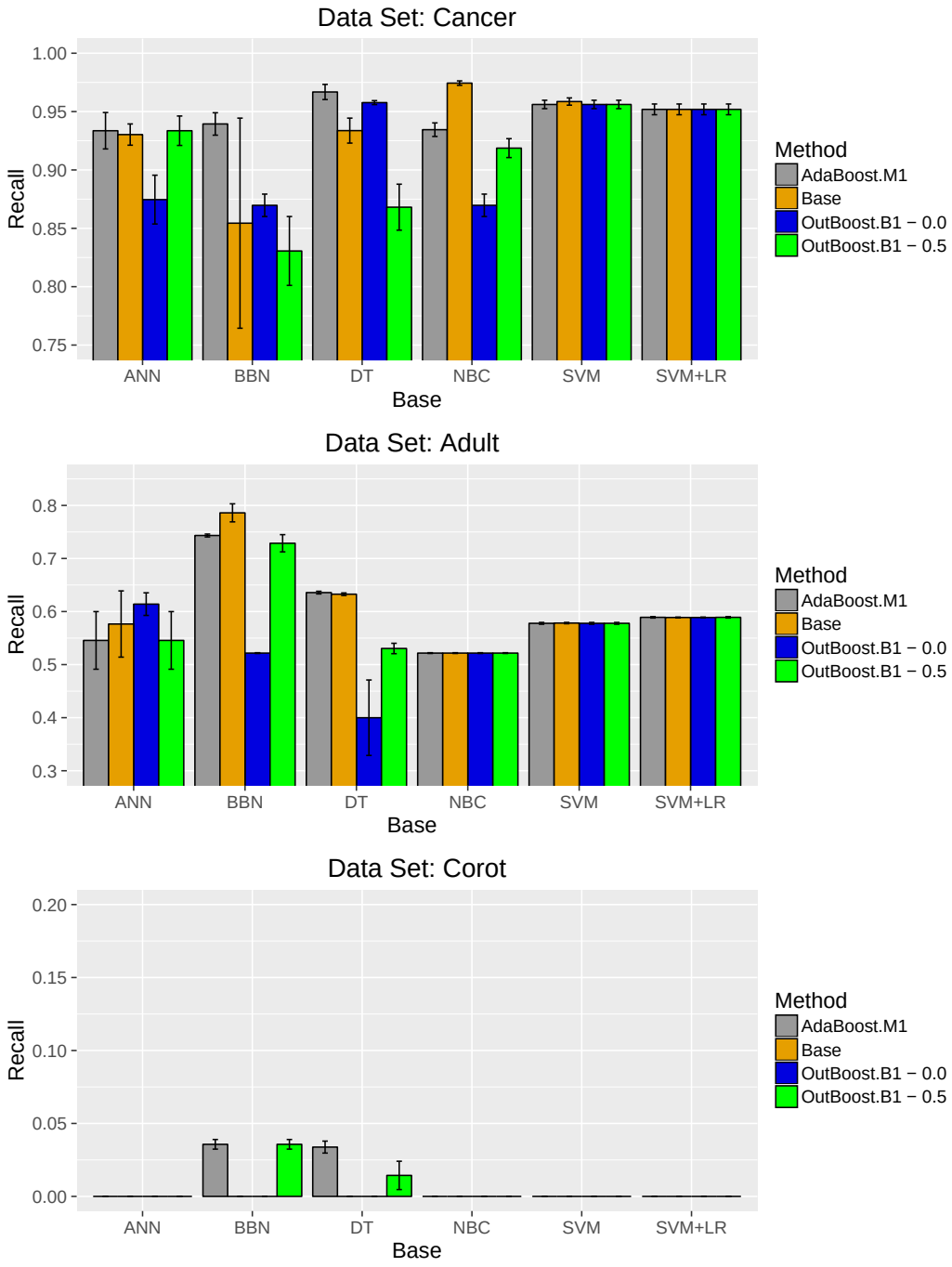


Figure 5: Comparison of the mean recall of OutBoost.B1, AdaBoost.M1, and its associated base learner.

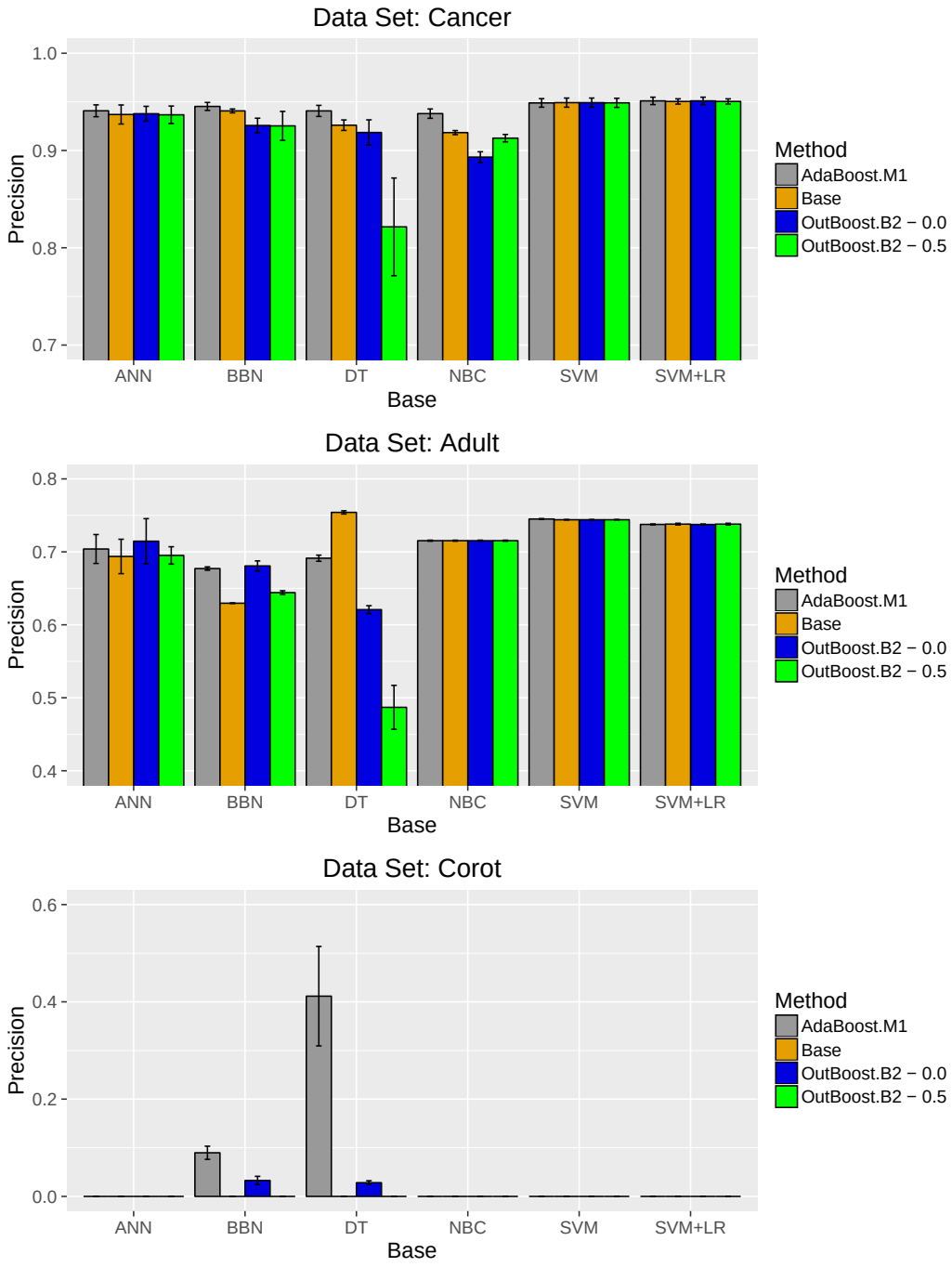


Figure 6: Comparison of the mean precision of OutBoost.B2, AdaBoost.M1, and its associated base learner.

Bibliography

[1] United States Environmental Protection Agency. *Data Quality Assessment: Statistical Methods for Practitioners*. Office of Environmental Information, 2006.

[2] Charu C Aggarwal. *Outlier analysis*. Springer Science & Business Media, 2013.

[3] Vic Barnett and Toby Lewis. *Outliers in statistical data*. Third edition, 1994.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[6] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[7] FY Edgeworth. Xli. on discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 23(143):364–375, 1887.

[8] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[9] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[10] David Freedman, Robert Pisani, and Roger Purves. *Statistics*. Fourth edition, 2007.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[12] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[13] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems*, pages 545–552, 2004.

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

[15] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[16] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186. Nashville, USA, 1997.

[17] M. Lichman. UCI machine learning repository, 2013.

[18] Clifford Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008.

[19] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012.

[20] Jeffrey Mervis. Agencies rally to tackle big data. *Science*, 336(6077):22–22, 2012.

[21] Tom M Mitchell. *Machine Learning*. McGraw Hill, 1997.

[22] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[23] David Poole. *Linear algebra: A modern introduction*. Cengage Learning, 2014.

[24] Foster J Provost, Tom Fawcett, et al. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *KDD*, volume 97, pages 43–48, 1997.

[25] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[26] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[27] Barbara G Tabachnick and Linda S Fidell. *Using multivariate statistics*. Pearson, sixth edition, 2013.

[28] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, et al. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006.

[29] John E Walsh. Large sample nonparametric rejection of outlying observations. *Annals of the Institute of Statistical Mathematics*, 10(3):223–232, 1959.

[30] John E Walsh et al. Some nonparametric tests of whether the largest observations of a set are too large or too small. *The Annals of Mathematical Statistics*, 21(4):583–592, 1950.

[31] John E Walsh et al. Correction to "some nonparametric tests of whether the largest observations of a set are too large or too small". *The Annals of Mathematical Statistics*, 24(1):134–135, 1953.

[32] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.