On Boosting and Noisy Labels

by

Jeffrey Chan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author .....
Department of Electrical Engineering and Computer Science
May 29, 2015

Certified by .....
Patrick H. Winston
Professor
Thesis Supervisor

Accepted by .....
Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

On Boosting and Noisy Labels

by

Jeffrey Chan

Submitted to the Department of Electrical Engineering and Computer Science
on May 29, 2015, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science

Abstract

Boosting is a machine learning technique widely used across many disciplines. Boosting enables one to learn from labeled data in order to predict the labels of unlabeled data. A central property of boosting instrumental to its popularity is its resistance to overfitting. Previous experiments provide a margin-based explanation for this resistance to overfitting.

In this thesis, the main finding is that boosting's resistance to overfitting can be understood in terms of how it handles noisy (mislabeled) points. Confirming experimental evidence emerged from experiments using the Wisconsin Diagnostic Breast Cancer(WDBC) dataset commonly used in machine learning experiments. A majority vote ensemble filter identified on average that 2.5% of the points in the dataset as noisy. The experiments chiefly investigated boosting's treatment of noisy points from a volume-based perspective. While the cell volume surrounding noisy points did not show a significant difference from other points, the decision volume surrounding noisy points was two to three times less than that of non-noisy points. Additional findings showed that decision volume not only provides insight into boosting's resistance to overfitting in the context of noisy points, but also serves as a suitable metric for identifying which points in a dataset are likely to be mislabeled.

Thesis Supervisor: Patrick H. Winston
Title: Professor

Acknowledgments

I thank Patrick Winston, my thesis supervisor. He provided me with insight, guidance, and support, without which this thesis would not be possible.

I thank Luis Ortiz for his helpful discussions concerning this work. His thoughts and expertise provided me with the resources to undertake this project.

Finally, I thank my parents, Wenyaw and Alice, and my sister, Stephanie. Their continuous and unwavering support throughout my academic career has provided me with the environment to grow, learn, and succeed, not only as an academic, but also as a person. It is to them that I owe all of my accomplishments.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, the main finding is that boosting's resistance to overfitting can be understood in terms of how it handles noisy (mislabeled) points. The experiments were applied to the Wisconsin Diagnostic Breast Cancer(WDBC) dataset commonly used in machine learning experiments. A majority vote ensemble filter identified on average that 2.5% of the points in the dataset as noisy. The experiments chiefly investigated boosting's treatment of noisy points from a volume-based perspective. Two chief volume-based metrics are used: *cell volume* defined as the volume of the region bounded by the closest base learners (or rules of thumb) to a point and *decision volume* defined as the volume of the region bounded by the closest decision boundary to a point. While the cell volume surrounding noisy points did not show a significant difference from other points, the decision volume surrounding noisy points was two to three times less than that of non-noisy points. Additional findings showed that decision volume not only provides insight into boosting's resistance to overfitting in the context of noisy points, but also serves as a suitable metric for identifying which points in a dataset are likely to be mislabeled.

The field of machine learning uses algorithmic techniques to learn and adapt from past observations or experience to make predictions accurately. For example, imagine that we want to give the computer a task of being able to 'read' scanned images of handwritten digits. A common machine learning framework for such a problem would involve gathering as many examples of scanned handwritten numbers as possible of

every type of digit. Next, feed these examples along with the answer for the machine learning algorithm to learn and adaptively 'train' its model. After going through this training phase, the machine learning algorithm would have formulated a classification rule that it can follow on new unlabeled examples. The goal of this machine learning framework is to be able to develop the most accurate rule to classify unseen images or examples.

Formulating an accurate classification rule is a very difficult problem since computers do not possess the pattern recognition abilities that humans do. However, one can come up with some basic rules that could improve classification. For example, if the digit is round and mostly symmetric, then it is likely to be a 0. While this rule alone does not suffice as a completely accurate classification rule, it slightly improves our classification rate compared to guessing at random.

The premise behind boosting is that developing the appropriate single prediction rule in one shot is often harder than finding many simple rules of thumb that classify a bit better than random. The boosting approach combines many simple rules of thumb generated from different weightings of the original set of training examples. Each iteration that the algorithm is called it creates a new simple rule based on the re-weighted distribution of training examples. Then, the boosting algorithm combines all of the simple rules (called base learners) and combines them in an overall prediction rule that should be much more accurate that any individual rule.

In order for this approach to work, there are some technical details that need to be resolved.

- How should the distribution over each dataset be chosen and updated after each round of boosting?
- How exactly do we combine the base learners into a single unified rule?

Commonly techniques in boosting choose the distribution over the training set to focus the most on the points that are difficult to develop a rule for. That is, weight the difficult to classify examples highly. For the second question, a natural manner in which to combine base learners is to simply hold a weighted majority vote for the

correct label.

Boosting is a popular machine learning technique used in many fields. Boosting owes its popularity to its resistance to overfitting. Previous experiments provide a margin-based explanation for this resistance to overfitting.

1.1 Background of Boosting

Boosting can trace its roots back to the Probably Approximately Correct (PAC) model developed by Valiant [36]. The idea behind the PAC model was that given a random set of training examples from a specified distribution, a *weak PAC learning* algorithm is defined as for any distribution the algorithm can with high probability find a classifier with a generalization error slightly better than random guessing given a polynomial number of examples and time. A *strong PAC learning* algorithm could find a classifier with an arbitarily small generalization error in a polynomial number of examples and time. This lead to Kearns and Valiant [19] in 1989 positing whether weak learnability implied strong learnability and came up with the idea of 'boosting' a weak learning algorithm to become a strong one.

This question led to Schapire [28] a year later proving the equivalence of weak and strong learnability and developing the first provable boosting algorithm that transforms a weak learner into a strong learner by using a recursively defined filtering strategy that filters the training set so that on each iteration the weak learner only received a subset of the training data. A year later, Freund [8] developed an optimal boosting algorithm based on the known upper bounds developed by the PAC framework using a simpler and more efficient algorithm that removes the filtering strategy and simply takes the majority of the outputs from each weak learner. A few years later Drucker et al. [6] ran the first experiments on boosting algorithms in application to optical character recognition data and developed error metrics to understand the performance of boosting, but found that the boosting algorithm was limited by practical drawbacks. In 1995, Freund and Schapire [10] developed their landmark algorithm, Adaboost, which solved many of the practical issues discovered

by Drucker et al. in earlier boosting algorithms.

1.2 Applications of Boosting

Boosting is one of the most popular machine learning techniques and has been widely applied throughout many fields. In medicine, Tan et al. [32] utilized boosting methods for cancer classification on microarray gene expression data, which had the additional benefit of identifying which genes were important to cancer diagnosis. Biological applications are not limited to cancer prediction but also extends to the difficult problem of protein structure prediction in which Niu et al. [24] employed an Adaboost learner. Guan et al. [15] used boosting in order to discover microRNAs among the entire genome while Xie et al. [40] employed it in a similar setting to discover DNA promotors. Furthermore, Middendorf et al.[22] used Adaboost to predict genetic regulatory response.

In the field of natural language processing, many techniques have been developed to employ or modify Adaboost in application to classifying text documents as well as classifying and understanding spoken language [1], [35]. Additional work has been done in natural language processing to disambiguate word sense using a modified boosting approach [7].

In robotics, Viola and Jones [38] use boosting in order to rapidly detect objects. Further work in robotics employed the boosting framework to enable a mobile robot to detect various places within an environment [23]. In computer vision, boosting has been used in the methodology of image retrieval [33]

1.3 Understanding of Boosting

With such a diverse set of applications for boosting, it is important to understand what makes boosting such a popular technique and understand its strengths and limitations.

Many previous studies have delved into understanding various theoretical aspects

of boosting. Lugosi et al. [21] showed that certain regularized bosting classifiers are Bayes- risk consistent. Jiang [17] showed that when boosting is performing suboptimally then a near-optimal way of minimizing prediction error and resisting overfitting is terminating the process early. Rudin et al. [27] analyzed the cylical properties of AdaBoost and the convergence properties in relation to the boosting margins. Kivinen and Warmuth [20] examines the choosing of new weights for boosting as an entropy projection of the distribution of weights on to the distribution of mistakes.

Many studies have also modified boosting to adapt to new settings or contexts. Bennett et al. [3] developed methodology to extend boosting methodology to the semi-supervised setting. Friedman [13] developed a method that adds randomization to gradient boosting which both improves speed and accuracy of the method. Ridgeway et al. [26] reformulated boosting to the context of regression problems. Freund and Mason [9] employed alternating decision trees as the base learner for boosting which achieved competitive performance and gave a natural notion of classification confidence.

The goal of this thesis is to further the understanding of boosting via a volume-based approach by developing an understanding for how the weak learners are chosen to affect noisy points as opposed to non-noisy points. The findings further tie this approach to our understanding of boosting in a margin sense and how these margins change with each successive iteration on noisy training examples. An important component of the analysis centers around the ability to filter noisy data, so the efficacy of the noise filter techniques in the literature is compared with a noise identification scheme based on examining the decision volume around each training example and apply this methodology to real-world datasets for validation.

This paper is organized as follows:

- Chapter 2 contains all related work regarding boosting and noise filtering
- Chapter 3 describes the methodology and dataset which we will employ
- Chapter 4 summarizes the results of the experiments

- Chapter 5 concludes the experiments and delves into the future directions of this study

Chapter 2

Related Work

2.1 Boosting

Boosting is a powerful machine learning algorithm founded on the idea that combining the labels of many 'weak' classifiers or learners translates to a strong robust one. Boosting is a greedy algorithm that fits adaptive models by sequentially adding these base learners to weighted data where difficult to classify points are weighted more heavily. Experts claim that boosting is the best off-the-shelf classifier developed so far.

The goal of boosting is to minimize

$$\min_{h} \sum_{i=1}^{n} \text{Loss}(y_i, h(x_i))$$

in a stagewise manner where many different loss functions can be used. At each iteration, the goal of minimizing the above problem is approached in a stagewise manner where a new classifier is added each time. Since the previous parameters cannot be changed, we call this approach *forward stagewise additive modeling*. The primary tuning parameter in forward stagewise additive modeling is the number of iterations. This parameter can be tuned via a validation set where the parameter can be chosen to be the point where the performance begins to decrease called *early stopping*. Alternative parameters such as AIC or BIC can also be used. Another

technique for achieving better generalization performance is to enforce a learning rate on each update making the first few iterations more 'important' than the last few. This technique is typically referred to as *shrinkage*.

In binary classification problems, it is natural to use 0-1 loss; however, since 0-1 loss is not differentiable different boosting techniques may use logloss or exponential loss as a convex upper bound for 0-1 loss.

2.1.1 Discrete Adaboost

The most popular boosting algorithm, Adaboost, was developed by Freund and Schapire (1997) [10] solving many of the practical drawbacks of earlier boosting methods utilizing an exponential loss function. The Adaboost algorithm takes a training dataset $S_n = \{(x_1, y_1), (x_2, y_2) \cdots (x_n, y_n)\}$ as input with binary labels $Y = \{-1, +1\}$ (although it can be extended to the multiclass case which we will cover in a later section). Adaboost iteratively calls a series of base learners in a series of rounds $t = 1 \cdots T$. Each new base learner $h(x; \theta_t)$ improves upon the overall classification of the 'committee' of base learners by weighting the ith training examples for round t denoted by $W_t(i)$. The weights are determined based on the classification performance of the ensemble of classifiers in previous iterations. If the classifier misclassifies a training example, then the weight of the the training example increases. Thus, the subsequent base learners focus on the examples that are hard to classify. After a base learner is chosen, Adaboost chooses an importance weight α_t for the classifier based on the error of the chosen classifier on the weighted training set. Thus, as the error of iteration t, ϵ_t increases, then the importance weight α_t decreases. Note that α_t cannot be updated in subsequent rounds but can only be changed by choosing the same learner in a later boosting round. Thus, we do not require that they sum to 1 and can simply normalize them later on. The final hypothesis resulting from the Adaboost algorithm is a weighted majority vote by the committee of base learners.

The final ensemble $H_T(x)$ can be written as

$$H_T(x) = \sum_{t=1}^{T} \alpha_t h(x; \theta_t)$$

where $h(x; \theta_t)$ is the base learner in boosting round t with parameters θ_t.

Adaboost in particular trains its ensemble classifier based on an exponential loss function

$$\text{Loss}(y, h(x)) = \exp(-yh(x))$$

where h is the classifier. The primary advantage of the exponential loss function is computational in its simplicity. We choose the values of α_t, θ_t by minimizing the loss function.

$$\begin{aligned} J(\alpha_t, \theta_t) &= \sum_{i=1}^{n} \text{Loss}(y_i, H_t(x_i)) \\ &= \sum_{i=1}^{n} \exp(-y_i H_{m-1}(x_i) - y_i \alpha_t h(x_i; \theta)) \\ &= \sum_{i=1}^{n} \exp(-y_i H_{m-1}(x_i)) \exp(-y_i \alpha_t h(x_i; \theta)) \\ &= \sum_{i=1}^{n} W_{t-1}(i) \exp(-y_i \alpha_t h(x_i; \theta)) \end{aligned}$$

Splitting the loss function into classified and misclassified points yields

$$\begin{aligned} J(\alpha_t, \theta_t) &= e^{-\alpha_t} \sum_{y_i = h(x_i;\theta)} W_{t-1}(i) + e^{\alpha_t} \sum_{y_i \neq h(x_i;\theta)} W_{t-1}(i) \\ &= (e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^{n} W_{t-1}(i) \mathbb{I}(y_i \neq h(x_i; \theta)) + e^{-\alpha_t} \sum_{i=1}^{n} W_{t-1}(i) \end{aligned}$$

Thus, the optimal base learner $h(x_i; \theta_t)$ is the base learner that minimizes the weighted

training error written as

$$h(x_i;\theta_t) = \arg\min_{h(x_i;\theta)} W_{t-1}(i)\mathbb{I}(y_i \neq h(x_i;\theta))$$

The appropriate importance weight can be found by differentiating the loss function to get

$$\frac{\partial}{\partial \alpha_t} J(\alpha_t, \theta_t) = -\sum_{i=1}^{n} W_{t-1}(i) y_i h(x_i;\theta_t)$$

Note that since this derivative is negative, we can expect the training loss to decrease by adding our new base learner. Now that we have all of the necessary components for the Adaboost algorithm, we can now explicitly define the steps.

We will only examine Discrete Adaboost which uses decision stumps $h(x;\theta) = sign(\theta_t x + \theta_t)$ as base learner (though there are other versions that use real-valued classifiers such as in Real Adaboost). Often it has been found that classification trees make good base learners. Note that decision stumps are special cases of a classification tree with depth 1.

2.1.2 Understanding Adaboost Error

We describe the weighted training error ϵ_t as the weighted error of the tth base learner with respect to the weights $W_{t-1}(i)$ on the training set. We can also measure the performance of the classifier on the next iteration by taking its error with respect to $W_t(i)$.

Weighted Training Error

The weighted training error ϵ_t with respect to $W_{t-1}(i)$ increases as more boosting iterations occur although it does not do so in a monotonic fashion as shown in Figure 2-1. This matches our intuition since the weights increase for difficult to classify points making the weighted training error more difficult to minimize for each subsequent base learner.

Algorithm 1 Discrete Adaboost (or Adaboost.M1)

1. Set $W_0(i) = \frac{1}{n}$ for $i = 1, 2, \cdots n$

2. At stage t, find a base learner (a decision stump) $h(x; \theta_t) = sign(\theta_{1,t}x + \theta_{0,t})$ that minimizes
$$-\sum_{i=1}^{n} W_{t-1}(i) y_i h(x_i; \theta_t) = 2\epsilon_i - 1$$
where ϵ_i is the weighted zero-one loss error on the training examples. The weights are the normalized by the weights from the previous boosting stage $W_{t-1}(i)$.

3. Given a stump, set the importance weight to
$$\alpha_t = 0.5 \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$
where ϵ_t is the weighted error computed above. α_t is chosen such that it minimizes the weighted exponential training loss
$$J(\alpha_t, \theta_t) = \sum_{i=1}^{n} W_{t-1}(i) \exp(-y_i \alpha_t h(x_i; \theta_t))$$

4. Update the weights on the training example with the new weights.
$$W_t(i) = \frac{1}{Z} W_{t-1}(i) \exp(-y_i \alpha_t h(x_i; \theta_t))$$
where Z is the normalization constant to ensure that the weights sum to 1. The weights $W_t(i)$ can be seen as the normalized loss of the ensemble $H_t(x_i)$ on training example i.

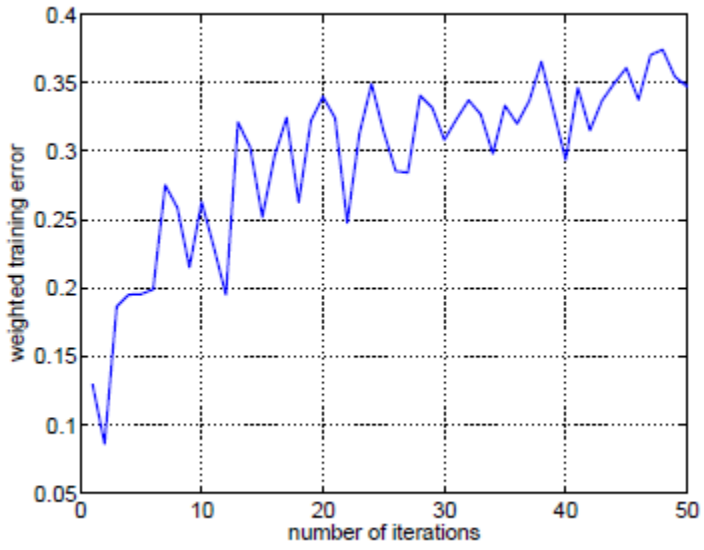

Figure 2-1: Weighted training error ϵ_t for the boosting algorithm on an artificial dataset.

Weighted Training Error on Updated Weights

The weighted training error with updated weights $W_t(i)$ is exactly 0.5 (equivalent to random guessing). We will show this by optimizing for α_t given θ_t

$$J(\alpha_t, \hat{\theta}_t) = \sum_{i=1}^{n} W_{t-1}(i) \exp(y_i \alpha_t h(x_i; \hat{\theta}_t))$$

$$\frac{d}{d\alpha_t} J(\alpha_t, \hat{\theta}_t)\Big|_{\alpha_t = \hat{\alpha}_t} = -\sum_{i=1}^{n} W_{t-1}(i) \exp(y_i \alpha_t h(x_i; \hat{\theta}_t))(y_i h(x_i; \hat{\theta}_t))$$

Thus, we have

$$\sum_{i=1}^{n} W_{t-1}(i) y_i h(x_i; \hat{\theta}_t) = 0$$

implying that the weighted agreement of the predicted and true label on the updated weights is 0 (this is equivalent to random guessing). This property has strong implications as it implies that the base learner from the tth boosting iteration will be useless for the next iteration. This prevents the same base learner from being chosen two iterations in a row. This makes sense as if you had the same base learner twice

in a row, this is equivalent to choosing it once while summing their α weights. This property ensures that weights are assigned efficiently. However, the same learner can appear in the future with respect to different weights since we nver have a chance to go back and update previous α_t's, so in order to tune them, we have to include the learner in a future iteration.

Ensemble Training Error

Now that we have examined how the error varies from iteration to iteration, it is important to understand how the error of the entire ensemble behaves. The ensemble training error does not decrease monotonically with each boosting iteration. However, the exponential loss function which Adaboost chooses weights and base learners to sequentially optimize for does decrease monotonically with each boosting iteration. We can compute exactly how much the exponential loss decreases on each successive iteration.

$$\begin{aligned}
J(\hat{\alpha}_t, \hat{\theta}_t) &= \sum_{i=1}^{n} W_{t-1}(i) \exp(y_i \hat{\alpha}_t h(x_i; \hat{\theta}_t)) \\
&= \sum_{i:y=h(x_i;\theta_t)} W_{t-1}(i) \exp(-\hat{\alpha}_t) + \sum_{i:y\neq h(x_i;\theta_t)} W_{t-1}(i) \exp(\hat{\alpha}_t) \\
&= (1-\hat{\epsilon}_t) \exp(-\hat{\alpha}_t) + \hat{\epsilon}_t \exp(\hat{\alpha}_t) \\
&= (1-\hat{\epsilon}_t) \sqrt{\frac{\hat{\epsilon}_t}{1-\hat{\epsilon}_t}} + \hat{\epsilon}_t \sqrt{\frac{1-\hat{\epsilon}_t}{\hat{\epsilon}_t}} \\
&= 2\sqrt{\hat{\epsilon}_t(1-\hat{\epsilon}_t)}
\end{aligned}$$

Notice that the above expression $J(\hat{\alpha}_t, \hat{\theta}_t)$ is equivalent to the amount that we renormalize the weights by and also the amount that the exponential loss decreases on each iteration. When $\epsilon_t < 1/2$, then this value is < 1, so the exponential loss is ensured to decrease by a factor of $2\sqrt{\hat{\epsilon}_t(1-\hat{\epsilon}_t)}$. Thus, the overall ensemble training error for exponential loss $Err_n(H_t)$ after t iterations is simply a product of these normalization constants

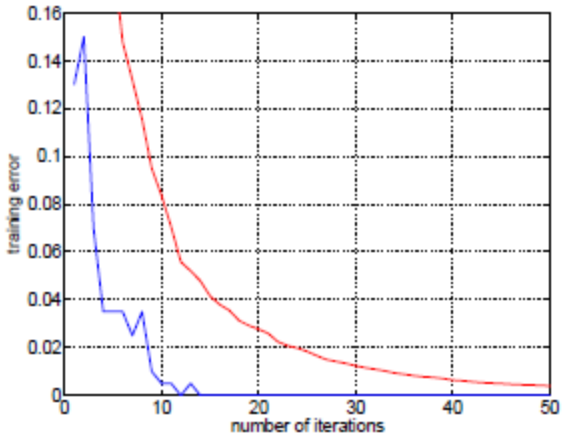

Figure 2-2: Ensemble training error with exponential upper bound for the boosting algorithm on an artificial dataset

$$Err_n(H_t) = \frac{1}{n}\sum_{i=1}^{n}\exp(y_i H_t(x_i)) = \prod_{k=1}^{t} 2\sqrt{\hat{\epsilon}_k(1-\hat{\epsilon}_k)}$$

Since the zero-one loss in computing ensemble training error is upper bounded by our exponential loss definition, then the ensemble training error is guaranteed to decrease the more iterations that occur. A plot of the ensemble training error (blue) and the exponential loss upper bound (red) can be seen in Figure 2-2

Ensemble Testing Error

A popular method for understanding ensemble testing error is to examine it in the sense of margins similar to what is done for Suppert Vector Machines (SVMs). This treatment of ensemble testing error is very important and we provide a more detailed treatment of the topic in the next section. The two notable properties that make boosting very powerful is that the testing error continues to decrease once the training error is zero and the testing error does not increase after a large number of iterations (ie it is resistant to overfitting). This is shown in Figure 2-3. The first property can be explained via a margin argument. The second property regarding resistance to overfitting is a result of the fact that the complexity of boosting does not increase very quickly with the addition of more iterations. In addition, another reason why

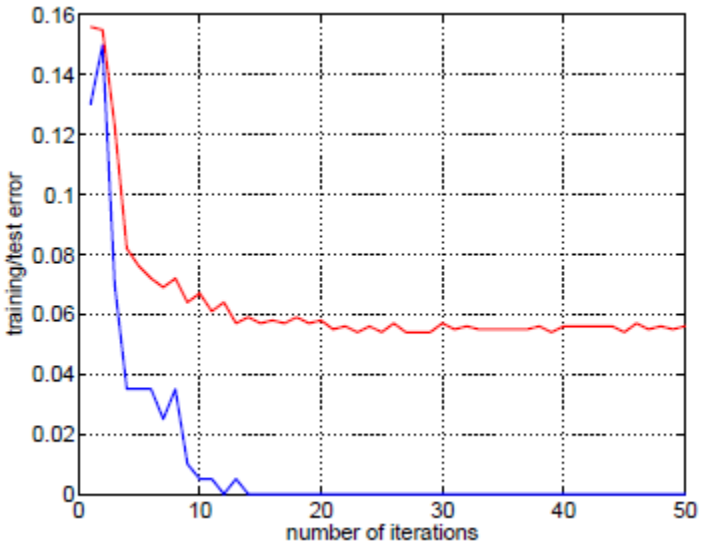

Figure 2-3: Training error(blue) vs Testing error(red) of the boosting algorithm on an artificial dataset

boosting is resistant to overfitting is that since it sequentially optimizes for exponential loss rather than jointly optimizing and updating α's, it is not very effective, so it is much harder to overfit.

2.1.3 Multiclass Classification

Boosting has commonly been used in the two-class case. Common approaches for extending boosting to the multi-class classification problem is to reduce it to a series of two-class classification problems. In the case of Adaboost, a natural extension to the multiclass problem developed by Schapire and Freund [10] based on pseudo-loss.

However, this extension of the binary problem has some drawbacks. The importance weight α requires each error ϵ to be less than half with respect to the distribution it was trained on in order for the classifier to be properly boosted. Otherwise, the importance weight α_t will be negative. In the case of binary classification, this stipulation just requires the classifier to perform better than random guessing which is usually satisfied. However, when extending to the multiclass case, the random guessing rate is $1/K$, but the classifier still must perform better than $1/2$ to avoid negative weights, which is much harder for a base learner to satisfy.

Algorithm 2 Multiclass Adaboost

1. Initialize the weights to be uniform across all $k-1$ incorrect labels

$$W_0(i,l) = \mathbb{I}(l \neq y_i)/n(k-1)$$

2. For each boosting stage t, find a base learner (a decision stump) $h(x;\theta_t)$ that minimizes the pseudoloss using our weights W

3. Let

$$\epsilon_t = \frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{k} D_t(i,l)(\mathbb{I}(y_i \neq h_t(x_i)) + \mathbb{I}(l = h_t(x_i)))$$

4. Assign the weight

$$\alpha = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$$

5. Update the weights

$$W_{t+1}(i,l) = W_t(i,l)\exp(\alpha_t[\mathbb{I}(y_i \neq h_t(x_i)) + \mathbb{I}(l = h_t(x_i))])$$

6. Normalize the new weights

7. The final classifier is

$$H(x) = \arg\max_{l \in Y} \sum_{t=1}^{T} \alpha_t \mathbb{I}(l \in h_t(x))$$

Zhu et al [41] modified the importance update step in an algorithm called Stage-wise Additive Modeling using a Multi-Class Exponential loss (SAMME) and provided theoretical guarantees on the new algorithm. The modification simply tacks on a $\log(K-1)$ term. Note in their algorithm, once again the error simply only needs to be better than the random guessing rate $\frac{1}{K}$.

Algorithm 3 SAMME

1. Initialize the weights to be uniform across all $k-1$ incorrect labels
$$W_0(i,l) = \mathbb{I}(l \neq y_i)/n(k-1)$$
2. For each boosting stage t, find a base learner (a decision stump) $h(x;\theta_t)$ that minimizes the pseudoloss using our weights W
3. Let
$$\epsilon_t = \frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{k} D_t(i,l)(\mathbb{I}(y_i \neq h_t(x_i)) + \mathbb{I}(l = h_t(x_i)))$$
4. Assign the weight
$$\alpha = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t}) + \log(K-1)$$
5. Update the weights
$$W_{t+1}(i,l) = W_t(i,l)\exp(\alpha_t[\mathbb{I}(y_i \neq h_t(x_i)) + \mathbb{I}(l = h_t(x_i))])$$
6. Normalize the new weights
7. The final classifier is
$$H(x) = \arg\max_{l \in Y} \sum_{t=1}^{T} \alpha_t \mathbb{I}(l \in h_t(x))$$

2.1.4 Other Boosting Algorithms

L2Boosting

L2Boosting [5] utilizes a squared error loss function which on the tth boosting iteration minimizes

$$\text{Loss}(y, H_t(x)) = (y_i - H_{t-1}(x_i) - h(x_i; \theta_t))^2$$

Buhlmann and Yu (2003) showed that when boosting with squared error loss can achieve identical results to Least Angle Regression (LARS) which is useful for variable selection.

LogitBoost

The problem with Adaboost and utilizing an exponential loss function is that it puts a significant amount of weight on the misclassified or 'difficult' examples. This causes the algorithm to be very sensitive to outliers (mislabeled training examples). In addition to this sensitivity, since the exponential loss function is not the logarithm of any density functions, it is difficult to extract corresponding probability estimates from the value returned by $H(x)$. A less harsh alternative is logloss which linearly concentrates on mistakes rather than exponentially. Such a boosting algorithm incorporating log loss seeks to minimize

$$L_t(h) = \sum_{i=1}^{n} \log(1 + \exp(-2y_i(H_t(x_i) + h(x_i))))$$

We can recover the probability estimates as

$$p(y = 1 \mid x) = \frac{1}{1 + e^{-2h(x)}}$$

The resulting binary classification boosting algorithm developed by Friedman et al. [11] is shown below.

Algorithm 4 LogitBoost

1. Initialize $W_i = \frac{1}{n}$ and $\epsilon_i = \frac{1}{2}$
2. At boosting stage t, compute the weights $W_i = \epsilon_i(1 - \epsilon_i)$
3. Find a base learner such that

$$h(x; \theta_t) = \arg\min_h = \sum_{i=1}^{N} w_i \left(\frac{y_i - \pi_i}{\pi_i(1 - \pi_i)} - h(x_i; \theta_t) \right)$$

4. Add to the ensemble our new base learner

$$H_t(x) = H_{t-1}(x) + \frac{1}{2} h(x; \theta_t)$$

5. Update the error to get

$$\pi_i = 1/(1 + \exp(-2H_t(x_i)))$$

Gradient Boosting

As we can see from the above versions of boosting, a unique boosting algorithm can be derived for each loss function and its performance can vary depending on which base learner. We can derive a generic version of boosting called gradient boosting [12]. In this approach, we seek to find the function that minimizes the loss function written as

$$\hat{h} = \arg\min_h L(h)$$

where h is our function. We can view this as a gradient descent in the function space. We perform a stagewise gradient descent in a stagewise fashion to get the gradient g_t

$$g_t(x_i) = \left[\frac{\partial L(y_i, h(x_i))}{\partial h(x_i)} \right]_{h=H_{t-1}}$$

We then update our ensemble to

$$H_t = H_{t-1} - \rho_t g_t$$

where ρ_t is the stepsize of the functional gradient descent. So far, this approach is not particularly useful because the function itself will most likely overfit the training data. We would like to find a base learner that can approximate the negative gradient with the following function

$$\theta_t = \arg\min_t \sum_{i=1}^{N} (-g_t(x_i) - h(x_i; \theta_t))^2$$

We can write the full algorithm seen below. Common versions of gradient boosting

Algorithm 5 Gradient Boosting

1. Initialize $h(x; \theta_0)$ as $\theta_0 = \arg\min_\theta \sum_{i=1}^{n} L(y_i, h(x_i; \theta))$
2. For boosting stage t, compute the gradient $g_t(x_i)$
3. Find the base learner $h(x; \theta_t)$ that minimizes $\sum_{i=1}^{n} (g_t(x_i) - h(x_i; \theta_t))^2$
4. Now add the learner to the new ensembler $H_t(x) = H_{t-1}(x) + \alpha_t h(x_i; \theta_t)$

utilizes decision trees as the base learner.

2.2 Max Margin

Early work by Freund and Schapire [10] sought to prove an upper bound on generalization error in order to develop an understanding for the performance of boosting. Initial upper bounds were computed as

$$\tilde{P}(H(x) \neq y) + O\left(\sqrt{\frac{Td}{n}}\right)$$

where $\tilde{P}$ is the empirical probability from the training set, T is the number of boosting iterations, d is the VC-dimension, and n is the size of the training set. The surprising finding during early experiments was that boosting did not tend to overfit the data despite the number of boosting iterations run on the training dataset with T acting as an upper bound. Typically, as hypotheses get more and more complex,

the generalization commonly degrades due to overfitting. Furthermore, not only did it not overfit the data, even after the training error reached zero, the generalization error continued to decrease with each subsequent iterations. There was significant interest behind understanding why such a complex hypothesis yielded extremely low error rates. Based on the work done by Bartlett [2] and Schapire et al. [29], in order to understand why boosting was resistant to overfitting, the authors did not simply examine how boosting effected the training error (the number of misclassified examples), but rather examined the confidence of the classification. In their work, they modeled the confidence of classification as the *margin* from the decision boundary. The margin of an example (x_i, y_i) can be written as

$$\text{margin}(x, y) = \frac{y_i \sum_{t=1}^{T} \alpha_t h(x_i; \theta_t)}{\sum_{t=1}^{T} \alpha_t}$$

Notice that the margin is in the range $[-1, +1]$ and is positive when correctly classified and negative when misclassified. The margin illustrates the level of certainty of confidence of the prediction. For instance, if all classifiers predict x_i as the same label, then the margin (or confidence) is 1. Note that this margin is different from the geometric margin utilized in SVMs which we will describe later. Their work proved an upper bound for the generalization error in terms of the margin and independent of T.

$$P(\text{margin}(x, y) \leq \rho) + O\left(\sqrt{\frac{d}{n\rho^2}}\right) \tag{2.1}$$

where $\rho > 0$ is the margin threshold. Since boosting is aggressive in concentrating on reducing the margin of the 'hard to classify' examples, it is able to continue to reduce the generalization error after the training error is 0.

Margin distribution graphs are helpful tools in understanding how the distribution of margins changes over each boosting iteration. Figure 2-4 is an example of the margin distribution graph at the beginning of boosting while Figure 2-5 represents the margin distribution graph after a few iterations. Notice that overall the distribution decreases and moves towards the right since boosting focuses on the examples with

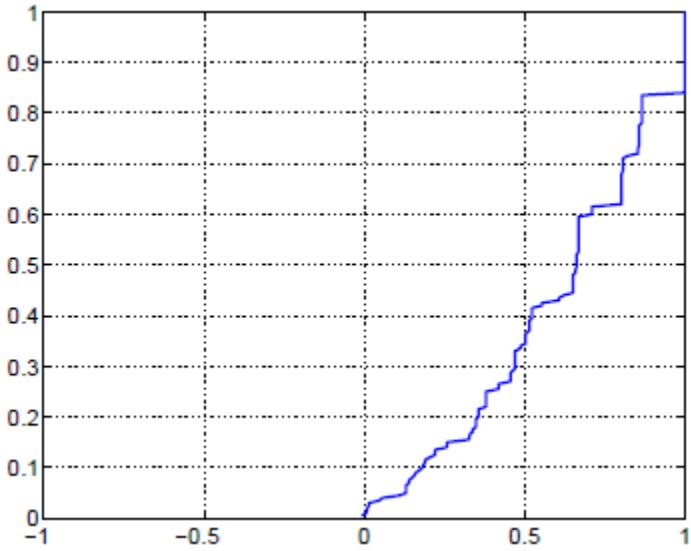

Figure 2-4: Initial margin distribution for the first iteration of the boosting algorithm on an artificial dataset

small margins.

2.2.1 Relation to Support Vector Machines

The margin theory developed by Vapnik [37] highlights a strong theoretical connection between boosting and support-vector machines. In analyzing Adaboost's generalization error in Equation 2.1, a way to approach the analysis is to minimize the bound. While Adaboost does not explicitly try to maximize the minimum margin, it does generally try to increase the margins to be as large as possible, so analyzing the maximization of the minimal margin is an approximation of Adaboost. Denote the importance and stumps vector as $\alpha = [\alpha_1, \cdots, \alpha_T]$ and $\mathbf{h} = [h(x;\theta_1), \cdots h(x;\theta_T)]$. The maximization of the minimal margin problem can be written as

$$\max_{\alpha} \min_{i} \frac{(\alpha \cdot \mathbf{h}(x_i))y_i}{\|\alpha\|\|\mathbf{h}(x_i)\|}$$

In the boosting case, the norms in the denominator are defined by the the L1 norm for $\|\alpha\|_1 = \sum_t |\alpha_t|$ and the L_∞ norm for $\|\mathbf{h}(x_i)\|_\infty = \max_t \mathbf{h}(x;\theta_t)$. In relation to SVMs whose explicit goal is to maximize the minimal geometric margin, both norms are the L2 norm $\|\alpha\|_2 = \sqrt{\sum_t \alpha_t^2}$ and $\|\mathbf{h}(x_i)\|_2 = \sqrt{\sum_t \mathbf{h}(x;\theta_t)^2}$.

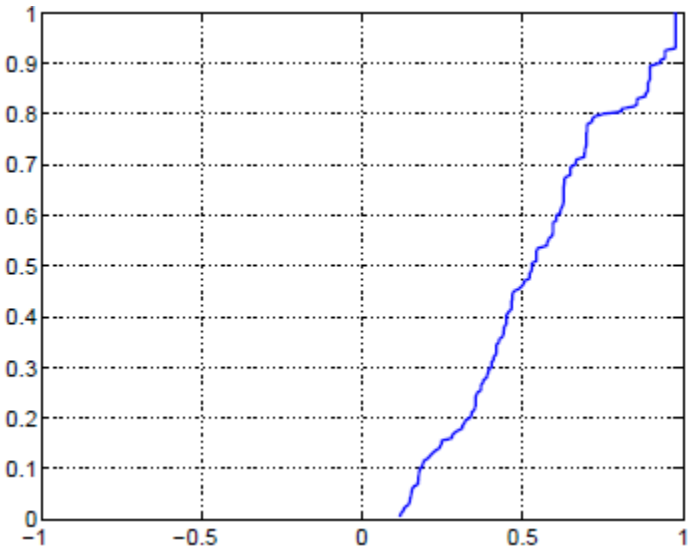

Figure 2-5: Final margin distribution for the 50th iteration of the boosting algorithm on an artificial dataset

The differences in norm have only a slight effect in low-dimensional space; however, in high-dimensional spaces, the margins can be drastically different. The difference can be exacerbated when α is very sparse which would cause Adaboost to have a much larger margin than SVM. A reason for boosting's resistance to overfitting is that the L1 norm ignores the irrelevant features.

2.3 Noise Identification

In order to gain an understanding the behavior of boosting methods on noisy data points, a method must be developed for identifying these noisy points on real-world data. Unfortunately, methodology for identifying noise still remain very rudimentary; otherwise, every machine learning problem would begin with filtering out all of the noisy data points from every dataset. Separating the signal from the noise in itself is a hard problem and has strong effects on machine learning algorithms' ability to generalize [4].

The source of labeling errors can come in a variety of forms from subjective evaluation, data entry error, or simply inadequate information. For example, subjectivity

can arise when a medical practitioner is attempting to classify disease severity. Another example in which there may be inadequate information is if a human records an image pixel based on color rather than the numeric input used by the algorithm to classify the data.

Wilson [39] first used the idea of noise filtering in which noisy points were identified to eliminate and improve classification performance. Wilson employed a k nearest neighbor classifier to filter the dataset and fed the correctly classified points into a 1-nearest neighbor classifier. Tomek [34] extended Wilson's algorithm for varying levels of k. Much further work was done on instance-based selection for the purpose of exemplar-based learning algorithms.

The danger of automatically flagging points that are difficult to correctly classify as noisy data is that they could be an exception to the rule rather than a noisy data point. An important question is to determine a method for differentiating between exceptions and noise. Guyon et al. [16] developed an information criterion to determine how typical a data point was, but since it was an on-line algorithm, it was sensitive to ordering. Srinivasan et al. [30] utilized an information theory-based method to separate exceptions and noise in the context of logical theory.

Oka et al. [25] developed methodology for learning generalizations and exceptions to the rule separately by separately noting which data points were correctly and incorrectly classified in the neighborhood around each training example. Their algorithm for differentiating the noise from the exceptions rested upon a user input which made sure the classification rate passed the threshold.

Gamberger et al. [14] developed a noise identification algorithm that focused first on inconsistent data points, that is points with both labelings present for the same feature values. After removing the inconsistent data, they transformed the features into binary values and removed features that most significantly reduced the number of literals needed to classify the data.

While many of the previous works have found noise detection methods for certain contexts in order to improve generalization and prediction, this does not fall in accordance with this study. Our goals are to identify noisy points to examine how

boosting handles these points in a volumetric sense. To this end, we seek to employ the general framework developed by Brodley et al. [4]. In their work, they use a cross-validation procedure in which each fold gets to vote on the left out fold and compares how difficult it is to classify.

2.3.1 Noise Filtering Algorithms

The general framework for the noise filtering algorithms we will use is as seen below.

Algorithm 6 Noise Filtering

1. Divide the dataset into n folds
2. For each fold, train the m classifiers on the remaining $n-1$ folds
3. The m classifiers are used to label each data point in the nth fold.
4. A point is called misclassified by a classifier, if a classifier assigns the point a label different from its given label.
5. A metric is used to determine how heavily misclassified a point to determine whether to consider it noise.

Single Algorithm Filters

There are two types of single algorithm filters. One uses the same algorithm for both filtering and classifying. An example of this idea is fitting a dataset in regression analysis, removing the datapoints, and re-fitting on the modified dataset. Another common example of this approach is employed by John [18] in which a decision tree is created and subsequently pruned. Once you remove the corresponding pruned training points, then you can re-fit the data to a decision tree. This differs slightly with the framework we will follow in that John's approach runs multiple iterations rather than utilizing cross-validation.

The other approach in single algorithm filters is to use one algorithm for filtering and another one for classifying. The reason one might choose this approach over using the same algorithm for both is that some algorithms act as better filters while others may be better at classification.

Ensemble Algorithm Filters

Ensemble classifier algorithms combines the labels of multiple base (or weak) learners to determine a label. Boosting is an example of a ensemble classifier. There are two types of ensemble filters. One approach is the majority vote ensemble filter. The majority vote ensemble filter trains the ensemble of base learners on the $n-1$ folds, then labels a point in the nth fold as mislabeled if a proportion x of the m base learners incorrectly classifies the training point. The other approach for ensemble filtering is consensus filtering in which a point is labeled as noise if none of the base learners can correctly classify the data point. Consensus filtering is the more conservative of the two approaches and preferrable if your approach is averse to getting false positives. On the other hand, consensus ensemble filtering retains many false negatives.

Filter Errors

When identifying noisy training examples, two types of noise errors will often result for any imperfect filter. The first type are false positives where correctly labeled points are identified as noisy points in the dataset (the yellow discarded portion in Figure 2-6). The second type is when a mislabeled (noisy) point is not correctly filtered out of the dataset and identified (the blue mislabeled portion in Figure 2-6). In order to appropriately choose what type of enesemble filtering algorithm to utilize, it is important to develop an understanding of the relative errors of the two approaches.

In the majority vote ensemble filter case, the event of incorrectly marking a non-noisy data point as noisy occurs when more than half of the m base learners fail to correctly classify the data point. Define $P(\text{Discarded }_i)$ as the probability for a given classifier i to incorrectly classify a point. Assume for simplicity that all classifiers have the same discard probability and assume that errors on data points are independent, we can express the overall probability that an majority vote ensemble filter will discard a good training example as

$$P(\text{Discarded}) = \sum_{j>m/2}^{m} P(\text{Discarded}_i)^j (1 - P(\text{Discarded}_i))^{m-j} \binom{m}{j}$$

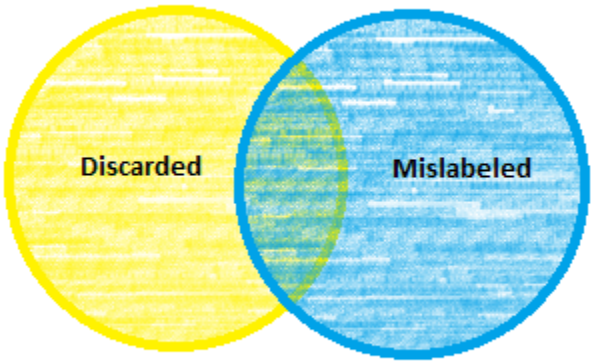

Figure 2-6: A schematic of discarded vs mislabeled points for a noise identification filter

where each term in the sum represents the probability of j errors occuring among the m base learners. If the probability of each classifier making such an error is < 0.5, then the majority vote ensemble filter is expected to perform better than a single-algorithm filter using the same base learner.

Now we examine the probability of mislabeling a noisy instance as a non-noisy one (the blue circle in the figure). Once again this occurs when more than half of the base learners classifies the noisy point as the incorrect class. Similar to before, we define $P(\text{Mislabeled}_i)$ as the probability for a given classifier i to mislabel a noisy point. Once again, we make the same assumptions as before regarding independent errors and equal error probability for each base learner. The probability under these assumptions for a given noisy data point is given as

$$P(\text{Mislabeled}) = \sum_{j>m/2}^{m} P(\text{Mislabeled}_i)^j (1 - P(\text{Mislabeled}_i))^{m-j} \binom{m}{j}$$

Similar to before, if the probability of each base learner making an error is < 0.5, then it is expected to make fewer errors than the single-algorithm with the same base learner.

For consensus ensemble filters, we use the same notation as we did for majority

vote ensemble filters. Then, $P(\text{Discarded})$ can be written as

$$P(\text{Discarded}) = P(\text{Discarded}_1)P(\text{Discarded}_2 \mid \text{Discarded}_1)\cdots$$

If we include the assumptions from before, then we can rewrite the probabilities as a product of this form

$$P(\text{Discarded}) = \prod_{i=1}^{m} P(\text{Discarded}_i)$$

When the independence assumption holds, the probability of error is less than that of a single-algorithm filter.

Once again we now wish to compute the probability of a consensus ensemble filter mislabeling a noisy example as non-noisy. This event occurs when one or more of the base learners correctly classifies the example. We can write this probability as

$$P(\text{Discarded}) = 1 - (1 - P(\text{Discarded}_1))(1 - P(\text{Discarded}_2 \mid \bar{\text{Discarded}}_1))\cdots$$

When the errors are independent, we can write them as

$$P(\text{Discarded}) = 1 - \prod_{i=1}^{m}(1 - P(\text{Discarded}_i))$$

In contrast to the other cases, a single algorithm filter would be expected to make fewer mistakes than an consensus ensemble one with the same base learner.

Chapter 3

Methodology and Experiments

3.1 Dataset

The experiments are conducted on the commonly used Wisconsin Diagnostic Breast Cancer(WDBC) machine learning dataset publicly available at the University of California Irvine repository (http://archive.ics.uci.edu/ml/). The breast cancer dataset was first collected and published in 1995 by Street et al. [31]. The dataset contains 699 total examples with 458 benign examples and 241 malignant examples. The dataset contains 10 features with values ranging from 1 - 10 containing information such as clump thickness, uniformity of cell size, etc. There are 16 missing values. Missing values were replaced with the mean value of the feature. The features are computed from a digitized image of a fine needle aspirate (FNA) of breast mass as can be seen in Figure 3-1. The features describe the properties of the cell nuclei. All features contain 4 significant digits.

3.2 Noise Identification Method

In order to properly evaluate how boosting affects noisy points on real data, it is necessary to develop a noise identification algorithm which is able to accurately determine which points are noisy. It is important that both false positives and false negatives are equally minimized. For that reason, a majority vote ensemble filter was

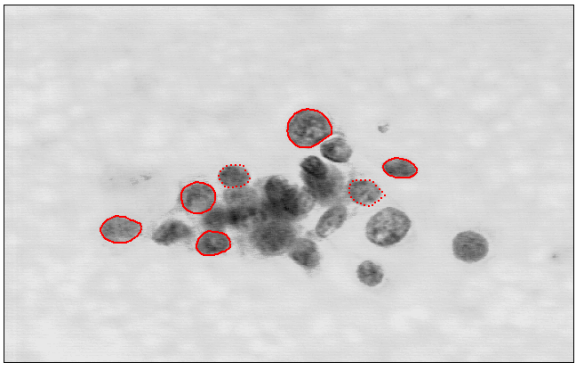

Figure 3-1: An example of a fine needle aspirate image used for the Wisconsin Diagnostic Breast Cancer dataset

chosen using Adaboost run on a small number of iterations as the filter.

Algorithm 7 Majority Vote Ensemble Filter (Adapted from Brodley et al)

1. Randomly shuffle and divide the dataset into k folds.
2. For each fold, run Adaboost on the remaining $k-1$ folds using m base learners (where m is relatively small).
3. For a given training point i, if more than a fraction p of the base learners misclassify it, label it as a noisy point.

3.3 Decision Volume

Define decision volume as the volume of the region around a given point p where all points in the region have the same label. The region must be continuous (ie you can connect any two points in the region without having to cross a point with a different label). This quantity is of interest because it may serve as a good metric for noise identification in addition to providing insight into the mechanics behind Adaboost. It is worth noting that it is not uncommon for the decision volume value to be infinite, so for practical computations the volume is restricted to be within the d- dimensional hyperrectangle that exactly circumscribes the dataset (ie no hyperrectangle with smaller volume could contain the entire dataset). Even when restricting the experiments to Discrete Adaboost, decision volume becomes computationally challenging to compute

once the number of points and dimensions scales, so an approximation of decision volume will be used.

Algorithm 8 Approximate Decision Volume Algorithm for Discrete Adaboost

1. Define $MIN[d]$ and $MAX[d]$ as the minimum and maximum value respectively of the dataset along the dth dimension
2. For each dimension d, select all decision stumps that partition the space along d.
3. Sort the selected stumps.
4. Search for the closest stumps before and after each point with a different label.
5. If there is no closest stump before or after the point, replace with $MIN[d] - \epsilon$ and $MAX[d] + \epsilon$ where $\epsilon = 0.01$.
6. Update the volume for each point given the length as the difference between the two stumps

Also, aother application of interest is the usage of decision volume as a proxy for noise identification.

Algorithm 9 Decision Volume Noise Filter

1. Compute the decision volume for each point
2. Take the fraction p with the smallest decision volumes as noisy.

3.4 Cell Volume

Another interesting volume-based metric is how Adaboost affects the cell volume (defined as the volume of the region bounded by the base learners surrounding a given point) surrounding noisy points in contrast with the non-noisy points. This insight into how Adaboost affects the cell volumes of noisy points with each successive boosting iteration could lead to a unique perspective of the underlying mechanism that allows Adaboost to be resistant to overfitting, an integral part of boosting's popularity and wide applications. Computing cell volume for general base learners is

difficult and computationally intensive so for these experiments, Adaboost is restricted to the class of decision stumps as a base learner.

Algorithm 10 Cell Volume Algorithm for Discrete Adaboost (with base learner of decision stumps)

1. Define $MIN[d]$ and $MAX[d]$ as the minimum and maximum value respectively of the dataset along the dth dimension
2. For each dimension d, select all decision stumps that partition the space along d.
3. Sort the selected stumps.
4. Search for the closest stumps before and after each point.
5. If there is no closest stump before or after the point, replace with $MIN[d] - \epsilon$ and $MAX[d] + \epsilon$ where $\epsilon = 0.01$.
6. Update the volume for each point given the length as the difference between the two stumps

3.5 Margin Distribution

In order to compare the volume-based approach to resistance to overfitting, decision and cell volume are compared the the results with the classical margin maximization explanation for boosting's resistance to overfitting.

Algorithm 11 Margin

1. For every point x_i, compute

$$\sum_t \alpha_t y_i h(x_i; \theta_t)$$

2. Normalize by $\sum_t \alpha_t$

Chapter 4

Results and Discussion

This thesis examined how the noise filter performs as a function of the number of iterations used in the Adaboost majority vote ensemble filter as well as the threshold of votes used to determine as noisy. I set the number of folds $K = 3$ and thresholds $= \{0.5, 0.6, 0.7, 0.8, 0.9\}$. The maximum number of estimators allowed was 50. The results are shown in Figure 4-1. From the figure, the highly noisy points disappear if the filter has 7 estimators. For a noise threshold of 0.5, the fraction of noisy points stabilizes around 0.45.

Next, experiments were run to understand the relationship between cell volume and Adaboost. The majority vote ensemble filter was used to identify noise. I set the number of folds $K = 3$, the threshold $\epsilon = 0.5$, and the number of estimators in the ensemble to 4. The experiment compared the cell volume and the training error for 300 iterations of the boosting algorithm. The noise filter classified 2.4% of the data points as noisy. The results are shown in Figure 4-2. Note that the boosting algorithm initially reduces the cell volume of the 'easy to classify' non- noisy points first in the first 240 iterations. Then, once the training error stabilizes, it focuses on reducing the cell volume of the noisy data points.

Next, experiments were run on decision volume, and how it differs from that of the cell volume. Once again the majority vote ensemble was used for noise identification. The noise filter parameters were set the same as before. The decision volume and the training error over 50 iterations of the boosting algorithm were investigated. The

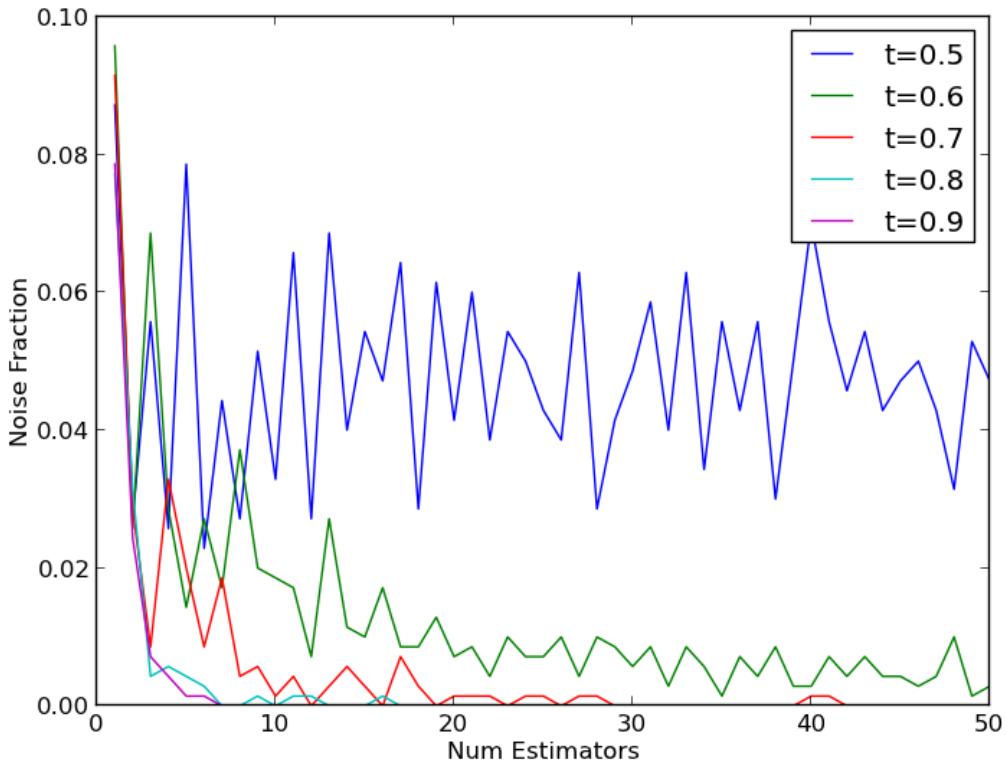

Figure 4-1: The fraction of noisy points in the dataset from the majority vote ensemble filter with respect to the number of iterations in boosting and noise threshold.

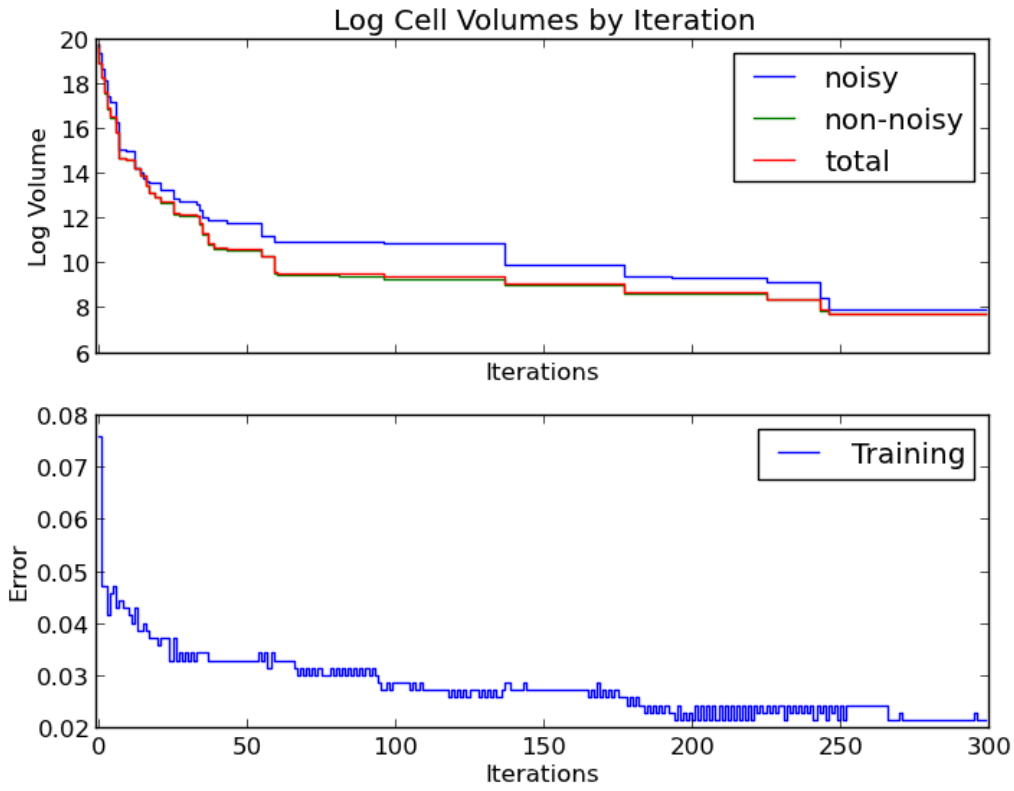

Figure 4-2: The log cell volume with each successive iteration of boosting for noisy and non-noisy points. Below is the training error for each iteration of boosting.

noise filter selected 2.28% of the points as noisy. The results of the experiment are shown in Figure 4-3. Note that the boosting algorithm immediately cuts the decision volume for the non-noisy points within the first few iterations. However, once this occurs its begins to focus on the noisy points until the mean decision volumes for noisy points are significantly less than that of non -noisy ones. The difference in volume size stabilizes such that the non-noisy point volumes are 2-3 times the size of noisy ones. This agrees with our understanding of boosting as it should be incentivized to minimize the decision volume for the noisy points because the point is the result of a labeling error. The point at which it begins to focus on the noisy points is around the 4th iteration, which coincides with the number of base learners in the noise filter used. This is because in prior iterations a large portion of the points were all candidates for being noisy points, so the decision volume of the ones Adaboost could correctly classify were significantly smaller until it reached the 4th iteration where it hones in on the criteria for mislabeled points, and seeks to classify them correctly.

The margin distributions of the points are computed to understand how the margins change over iterations in this context. Again, the same noise filter and parameter settings are used as before. As shown in Figure 4-4, in this context the margins for noisy points behave in a way as described in previous boosting margin experiments in the literature.

From the earlier experiments, it has been shown that noisy data points tend to have smaller decision volumes than their non-noisy counterparts. This leads to experimenting with the usage of decision volume as a proxy for noise identification. In order to entertain this possibility, the overlap of noisy points chosen by the ensemble filter and the decision volume based approach is computed. Note that it is difficult to identify the true noisy points. As before, the ensemble filter is set to have $K = 3$, and number of estimators is set to 4. In this experiment, the results are presented for varying thresholds $\{0.5, 0.6, 0.7, 0.8, 0.9\}$. Noisy points for the decision volume approach are chosen by computing the bottom fraction of decision volumes. This fraction is based on the fraction of noisy points chosen in the equivalent ensemble filter for a given threshold. The results are shown in Figure 4-5. Note that the behavior

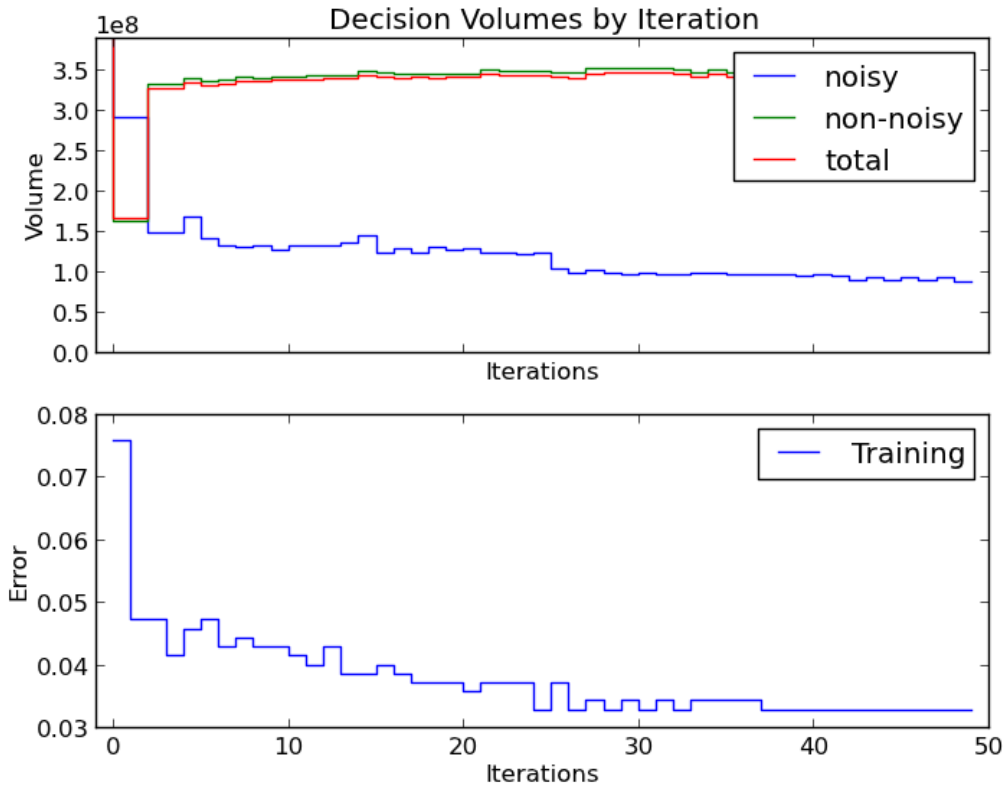

Figure 4-3: The mean decision volumes with each successive iteration of boosting for noisy and non-noisy points. On the bottom is the training error for each iteration of boosting.

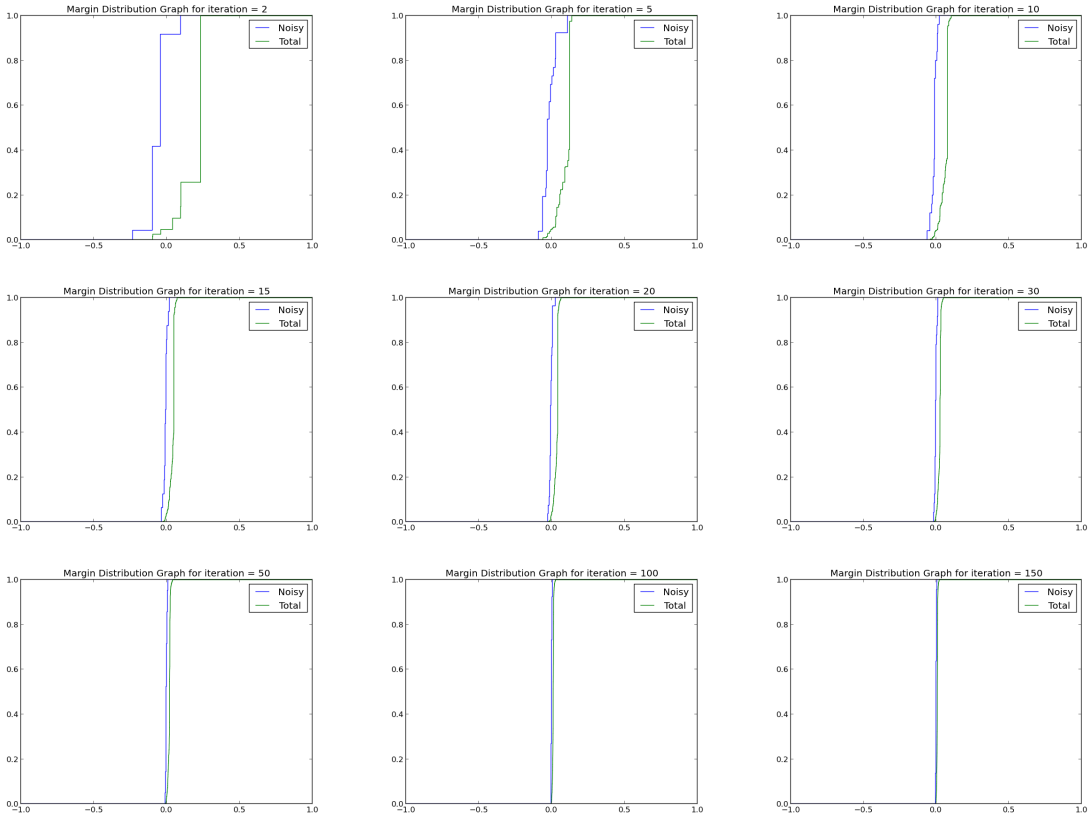

Figure 4-4: The margin distribution (expressed as a cumulative distribution) for noisy and non-noisy points different iterations of boosting.

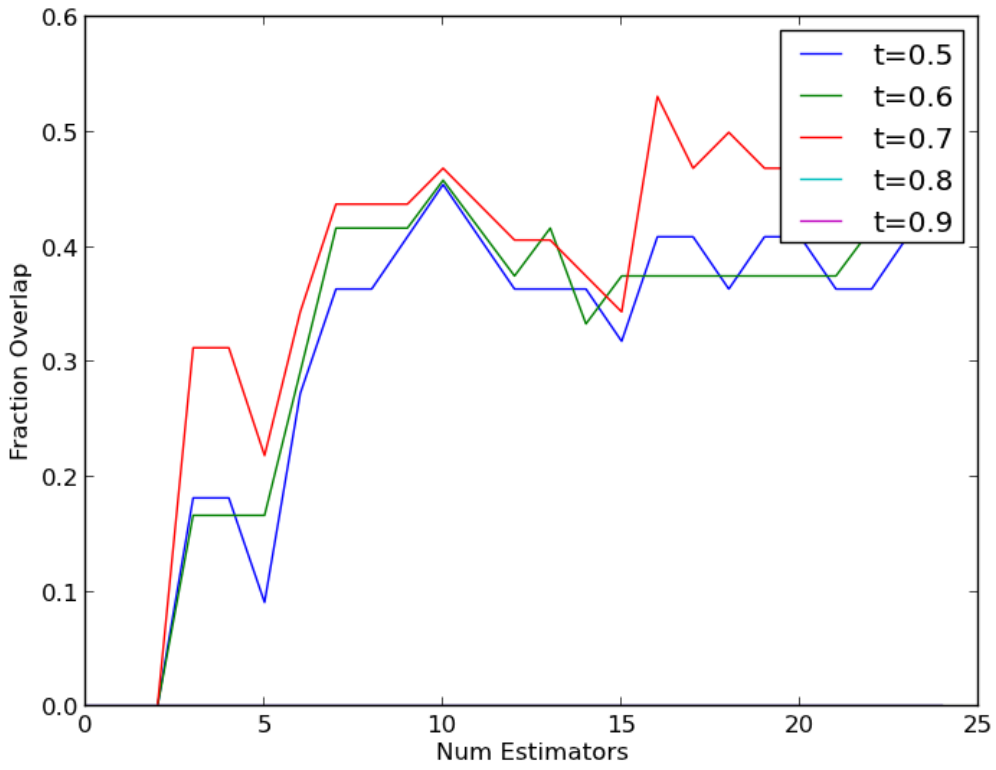

Figure 4-5: The fraction of overlap between the decision volume filter and ensemble filter for a varying number of iterations of the decision volume filter and varying thresholds for the ensemble filter

for the first few iterations is irregular since all decision volumes within the first few iterations are roughly uniform. In addition, for high thresholds, there is minimal overlap since very few points meet such a high threshold. After a few iterations the overlap of noisy points stabilizes around 0.5, which indicates that a decision volume filter is a viable alternative to identifying noisy points.

Chapter 5

Contributions

I have demonstrated that Adaboost seeks to minimize the decision volume of noisy points with each additional iteration in comparison to non-noisy points. This minimization of the decision volume of noisy points appears to be a consequence of boosting's ability to generalize well and not overfit the data. The reduction of the decision volume surrounding noisy points prevents the model from suffering from the effects of overfitting as the decision volume can be thought of as the points 'region of influence.'

I have also found that Adaboost does not seem to particularly affect the cell volumes of the noisy points but instead decreases the cell volume with each iteration in a uniform manner. This implies that Adaboost does not simply attempt to place stumps near noisy points to reduce the cell volume, but as can be seen from the derivation attempts to reduce the decision volume by minimizing the exponential loss.

This volume-based understanding of Adaboost provide an interesting perspective of Adaboost. The margin-based explanation of Adaboost's resistance to overfitting uses a notion of margin that incorporates the 'distance' of all of the weak learners. Cell volume only accounts for the weak learners immediately surrounding it without accounting for labeling or weights, which is a more local approach compared to the margin's more global view of Adaboost. Decision volume accounts for the closest decision boundary in a similarly local manner to cell volume, but the decision boundary

is a result of the weighting of all of the weak learners, so it combines both the local and global perspective of the two.

Finally, the finding that decision volume could be used as a noise identification procedure requires further research. The application of this procedure to different datasets especially those where the noisy labels are known or artificially created would provide for an interesting study. If decision volume is a viable noise identification metric, its performance can be used in comparison with other noise identification metrics and used as a filtering method to make a dataset friendlier to algorithms that are not as resistant to overfitting.

Bibliography

[1] Baharum Baharudin, Lam Hong Lee, and Khairullah Khan. A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, 1(1):4–20, 2010.

[2] Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *Information Theory, IEEE Transactions on*, 44(2):525–536, 1998.

[3] Kristin P Bennett, Ayhan Demiriz, and Richard Maclin. Exploiting unlabeled data in ensemble methods. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–296. ACM, 2002.

[4] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. 1999.

[5] Peter Bühlmann and Bin Yu. Boosting with the l 2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

[6] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):705–719, 1993.

[7] Gerard Escudero, Lluís Màrquez, and German Rigau. Using lazyboosting for word sense disambiguation. In *The Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems*, pages 71–74. Association for Computational Linguistics, 2001.

[8] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[9] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *icml*, volume 99, pages 124–133, 1999.

[10] 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.

[11] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[12] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[13] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[14] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise elimination in inductive concept learning: A case study in medical diagnosis. In *Algorithmic Learning Theory*, pages 199–212. Springer, 1996.

[15] Dao-Gang Guan, Jian-You Liao, Zhen-Hua Qu, Ying Zhang, and Liang-Hu Qu. mirexplorer: detecting microrna s from genome and next generation sequencing data using the adaboost method with transition probability matrix and combined features. *RNA biology*, 8(5):922–934, 2011.

[16] Isabelle Guyon, Nada Matic, Vladimir Vapnik, et al. Discovering informative patterns and data cleaning., 1996.

[17] Wenxin Jiang. Process consistency for adaboost. *Annals of Statistics*, pages 13–29, 2004.

[18] George H John. Robust decision trees: Removing outliers from databases.

[19] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[20] Jyrki Kivinen and Manfred K Warmuth. Boosting as entropy projection. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 134–144. ACM, 1999.

[21] Gábor Lugosi and Nicolas Vayatis. On the bayes-risk consistency of regularized boosting methods. *Annals of Statistics*, pages 30–55, 2004.

[22] Manuel Middendorf, Anshul Kundaje, Chris Wiggins, Yoav Freund, and Christina Leslie. Predicting genetic regulatory response using classification. *Bioinformatics*, 20(suppl 1):i232–i240, 2004.

[23] O Martınez Mozos, Cyrill Stachniss, and Wolfram Burgard. Supervised learning of places from range data using adaboost. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1730–1735. IEEE, 2005.

[24] Bing Niu, Yu-Dong Cai, Wen-Cong Lu, Guo-Zheng Li, and Kuo-Chen Chou. Predicting protein structural class with adaboost learner. *Protein and peptide letters*, 13(5):489–492, 2006.

[25] Natsuki Oka and Kunio Yoshida. A noise-tolerant hybrid model of a global and a local learning module. *Behaviormetrika*, 26(1):129–143, 1999.

[26] Greg Ridgeway, David Madigan, and Thomas Richardson. Boosting methodology for regression problems. In *Proceedings of the International Workshop on AI and Statistics*, pages 152–161, 1999.

[27] Cynthia Rudin, Ingrid Daubechies, and Robert E Schapire. The dynamics of adaboost: Cyclic behavior and convergence of margins. *The Journal of Machine Learning Research*, 5:1557–1595, 2004.

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

[29] Robert E Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

[30] Ashwin Srinivasan, Stephen Muggleton, and Michael Bain. Distinguishing exceptions from noise in non-monotonic learning. Citeseer.

[31] W Nick Street, William H Wolberg, and Olvi L Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*, pages 861–870. International Society for Optics and Photonics, 1993.

[32] Aik Choon Tan and David Gilbert. Ensemble machine learning on gene expression data for cancer classification. 2003.

[33] Kinh Tieu and Paul Viola. Boosting image retrieval. *International Journal of Computer Vision*, 56(1-2):17–36, 2004.

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

[35] Gokhan Tur, Dilek Hakkani-Tür, and Robert E Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171–186, 2005.

[36] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[37] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

[38] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[39] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):408–421, 1972.

[40] Xudong Xie, Shuanhu Wu, Kin-Man Lam, and Hong Yan. Promoterexplorer: an effective promoter identification method based on the adaboost algorithm. *Bioinformatics*, 22(22):2722–2728, 2006.

[41] Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.