Adaptive Learning Algorithms and Data Cloning

Thesis by

Amrit Pratap

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2008

(Defended February, 11 2008)

Acknowledgements

I would like to thank all the people who, through their valuable advice and support, made this work possible. First and foremost, I am grateful to thank my advisor, Dr. Yaser Abu-Mostafa for his support, assistance and guidance throughout my time at Caltech.

I would also like to thank my colleagues at the Learning Systems Group, Ling Li and Hsuan-Tien Lin, for many stimulating discussions and for their constructive input and feedback . I also would like to thank Dr. Malik Magdon-Ismail, Dr. Amir Atiya and Dr. Alexander Nicholson for their helpful suggestions.

I would like to thank the members of my thesis committee, Dr. Yaser Abu-Mostafa, Dr. Alain Martin, Dr. Pietro Perona and Dr. Jehoshua Bruck, for their time to review the thesis and all the helpful suggestions and guidance.

Finally I'd like to thank my family and friends for continuing love and support.

Abstract

This thesis is in the field of machine learning: the use of data to automatically learn a hypothesis to predict the future behavior of a system. It summarizes three of my research projects.

We first investigate the role of margins in the phenomenal success of the Boosting Algorithms. AdaBoost (Adaptive Boosting) is an algorithm for generating an ensemble of hypotheses for classification. The superior out-of-sample performance of AdaBoost has been attributed to the fact that it can generate a classifier which classifies the points with a large margin of confidence. This led to the development of many new algorithms focusing on optimizing the margin of confidence. It was observed that directly optimizing the margins leads to a poor performance. This apparent contradiction has been the topic of a long unresolved debate in the machine-learning community. We introduce new algorithms which are expressly designed to test the margin hypothesis and provide concrete evidence which refutes the margin argument.

We then propose a novel algorithm for Adaptive sampling under Monotonicity constraint. The typical learning problem takes examples of the target function as input information and produces a hypothesis that approximates the target as an output. We consider a generalization of this paradigm by taking different types of information as input, and producing only specific properties of the target as output. This is a very common setup which occurs in many different real-life settings where the samples are expensive to obtain. We show experimentally that our algorithm achieves better performance than the existing methods, such as Staircase procedure and PEST.

One of the major pitfalls in machine learning research is that of selection bias.

This is mostly introduced unconsciously due to the choices made during the learning process, which often lead to over-optimistic estimates of the performance. In the third project, we introduce a new methodology for systematically reducing selection bias. Experiments show that using cloned datasets for model selection can lead to better performance and reduce the selection bias.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Adaptive Learning

The main focus of this thesis is on Adaptive Learning algorithms. In Adaptive Learning, the algorithm is allowed to make decisions and adapt the learning process based on the information it already has from the existing data and settings. We consider two types of adaptive settings, the first in which the algorithm adapts to the complexity of the dataset to add new hypothesis forming an ensemble. In the second settings, the algorithm uses the data it has to decide the optimal data-point to sample from. This is very useful in problems where the data is at premium.

In the first setting, we analyze the Adaptive Boosting algorithm [Freund and Schapire 1996] which is a popular algorithm to improve the performance of many learning algorithms. It improves the accuracy of any base learner by iteratively generating an ensemble of base hypotheses. AdaBoost maintains a set of weights for the training examples and adaptively focuses on hard examples by giving them higher weights. It has been successfully applied to many real world problems with huge success [Guo and Zhang 2001, Schapire and Singer 2000, Schwenk and Bengio 1997, Schwenk 1999]. The superior out-of-sample performance of AdaBoost has been attributed to the fact that by adaptively focusing on the difficult examples, it can generate a classifier which classifies the points with a large margin of confidence. The bounds on the out-of-sample performance of a classifier which are based on it having large margins are however very weak in practice. Any attempts to optimize the margins further has lead to worsening in performance [Li et al. 2003, Grove and Schuurmans 1998]. In this thesis, we present new variants of the AdaBoost algorithms which have been designed to test the margin explanation. Experimental results show that the margin explanation is at best incomplete.

The second setting for Adaptive learning that we consider is the more conventional approach in which the algorithm is allowed to adaptively choose a data-point. This approach has many practical implications, especially in fields where the cost of obtaining a data-point is very high. The typical learning problem takes examples of the target function as input information and produces a hypothesis that approximates the target as an output. We consider a generalization of this paradigm by taking different types of information as input, and producing only specific properties of the target as output. We present new algorithms for estimating under monotonicity constraint and adaptively selecting the next data-point.

1.2 Data Cloning

One of the major pitfalls in Machine Learning research is that of Selection Bias. This is mostly introduced unconsciously due to the choices made during the learning process which often lead to over optimistic estimates of the performance. In this chapter, we introduce a new methodology for systematically reducing selection bias. Using cloned dataset for model selection results in a consistent improvement over cross validation and its performance is much closer to the out-of-sample selection. Experimental results on a variety of learning problems shows that the cloning process results in a significant improvement in model selection over insample selection.

Chapter 2

Boosting The Margins: The Need For a New Explanation

In any learning scenario, the main goal is to find a hypothesis that performs well on the unseen examples. In recent years, there has been a growing interest in voting algorithms, which combine the output of many hypotheses to produce a final output. These algorithms take a given "base" learning algorithm and apply it repeatedly to re-weighted versions of the original dataset, thus producing an ensemble of hypotheses which are then combined via a weighted voting scheme to form a final aggregate hypothesis.

AdaBoost [Freund and Schapire 1995] is the most popular and successful of the boosting algorithms. It has been successfully applied to many real-world problems with huge success [Guo and Zhang 2001, Schapire and Singer 2000, Schwenk and Bengio 1997, Schwenk 1999]. One interesting experimental observation about AdaBoost is that it continues to improve the out-of-sample error even when the training error has converged to zero [Breiman 1996, Schapire et al. 1997]. This is a very surprising property as it goes against the principle of Occam's razor which favors simpler explanations.

The most popular explanation for this phenomenon is that AdaBoost produces classifiers which have a large margin on the training points [Schapire et al. 1997]. So, even though the classifier has correctly classified all the training points, the additional hypotheses increases the margin of confidence on those points. Mason et al.

[2000b] showed that AdaBoost does gradient descent optimization on the soft-min of the margin in a functional space. There are theoretical bounds on the out-of-sample performance of an ensemble classifier which depends on the fraction of training points with small margin. It has however been observed that directly optimizing the minimum margin leads to poor performance [Grove and Schuurmans 1998]. There has been a long debate in the machine-learning community about the validity of the margin explanation [Breiman 1994; 1996; 1998; 1999, Schapire et al. 1997, Reyzin and Schapire 2006]. We propose variants of the boosting algorithm and prove empirically that the margin explanation is at best incomplete and motivates the need for a new explanation which would lead to the design of better algorithms for machine learning.

We will first introduce the notation and then describe the AdaBoost algorithm and its generalization, AnyBoost. We discuss the bounds provided by the margin theory. We introduce two variants of AdaBoost —AlphaBoost and DLPBoost— and discuss the results, which empirically refute the margin explanation and motivate the need for a new explanation.

2.1 Notation

We assume that the examples (x, y) are randomly generated from some unknown probability distribution $\mathcal{D}$ on $\mathcal{X} \times Y$, where $\mathcal{X}$ is the input space and Y is the output space. We will only be dealing with binary classifiers, so in general we will have $Y = \{-1, 1\}$. Boosting algorithms produce a voted combination of classifiers of the form $\text{sgn}(F(x))$ where

$$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

where $f_t : \mathcal{X} \to \{-1, 1\}$ are base classifiers from some fixed hypothesis class $\mathcal{F}$, and $\alpha_t \in \mathbb{R}^+$with $\sum_{t=1}^{T} \alpha_t = 1$ are the weights of the classifiers. The class of all convex combinations of functions from the base classifiers will be called $conv(\mathcal{F})$, so $F \in conv(\mathcal{F})$.

The margin(Δ) of an example (x, y) for the classifier $\mathrm{sgn}(F(x))$ is defined as $yF(x)$. Margin is a measure of the confidence on the decision. A large positive margin implies a confident correct decision.

Given a set $S = \{(x_1, y_1), ..., (x_n, y_n)\}$ of examples drawn from $\mathcal{D}$, the goal of learning is to construct a hypothesis —in our case a voted combination of classifier— so that it minimizes the out-of-sample error which is defined as $\pi = P_{\mathcal{D}}[\mathrm{sgn}(F(x)) \neq y]$, i.e., the probability that F wrongly classifies a random point drawn from the distribution $\mathcal{D}$. The in-sample distribution over the training points would be denoted by S , with the in-sample error defined as $\nu = P_S[\mathrm{sgn}(F(x)) \neq y]$.

2.2 AdaBoost

AdaBoost is one of the most popular boosting algorithms. It takes a base learning algorithm and repeatedly applies it to re-weighted versions of the original training sample, producing a linear combination of hypothesis from the base learner. At each iteration, AdaBoost emphasizes the misclassified examples from the previous iteration, thereby forcing the weak learner to focus on the "difficult" examples. Algorithm 1 gives the pseudo-code of AdaBoost.

The effectiveness of AdaBoost has been attributed to the fact that it tends to produce classifiers with large margins on the training points. Theorem 1 bounds the generalization error of a voted classifier in terms of the fraction of points with a small margin.

Theorem 1 *[Schapire et al. 1997]*

Let S be a sample of N examples chosen independently at random according to $\mathcal{D}$. Assume that the base hypothesis space $\mathcal{F}$ has VC-dimension d, and let $\delta > 0$. Then with probability at least $1 - \delta$ over the random choice of the training set S,every weighted function $F \in lin(\mathcal{F})$ satisfies the following bound for all $\gamma > 0$

Algorithm 1 AdaBoost(S,T) [Freund and Schapire 1995]

```
• Input: $S = (x_1, y_1), ..., (x_N, y_N)$
• Input: $T$ the number of iterations
• Initialize $w_i = \frac{1}{N}$ for $i = 1, .., N$
• For $t = 1$ to $T$ do
    – Train the weak learner on the weighted dataset $(S, w)$ and obtain $h_{t:} : X \to \{-1, 1\}$
    – Calculate the weighted training error $\epsilon_t$ of $h_t$ :
          $$\epsilon_t = \sum_{i=1}^{N} w_i I[h_t(x_i) \neq y_i]$$
    – Calculate the weight $\alpha_t$ as:
          $$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$
    – Update weights
          $$w_i^{new} = w_i \exp\{-\alpha_t y_n h_t(x_n)\}/Z_t$$
      where $Z_t$ is a normalization constant
    – if $\epsilon_t = 0$ or $\epsilon_t \geq \frac{1}{2}$ then break and set $T = t - 1$
• Output $F_T(x) = \sum_{t=0}^{T} \alpha_t h_t(x)$
```

$$P_{\mathcal{D}}[sgn(F(x)) \neq y] \quad \leq \quad P_S[yF(x) \leq \gamma] + O\left(\sqrt{\frac{\gamma^{-2} d \log(N/d) + \log(1/\delta)}{N}}\right)$$

AdaBoost has been found to be particularly effective in increasing the margin of "difficult" examples (those with small margin), even at the price of reducing the margin of other examples. So, it seems that the effectiveness of AdaBoost comes from maximizing the minimum margin. Grove and Schuurmans [1998] devised an algorithm *LPBoost* which expressly maximizes the minimum margin, and achieves better minimum margin than AdaBoost. However, this lead to a worse out-of-sample performance. They concluded that no simple version of the minimum margin explanation can be complete.

2.3 AnyBoost: Boosting as Gradient Descent

Mason et al. [2000b] presents a generalized view of boosting algorithms as a gradient descent procedure in the functional space. Their algorithm, AnyBoost, iteratively minimizes the cost function by gradient descent in the functional space.

The base hypotheses and their linear combinations can be viewed as elements of an inner product space $(\mathcal{C}, \mathcal{D})$, where $\mathcal{C}$ is a linear space of functions that contain $lin(\mathcal{F})$, a generalization of $conv(\mathcal{F})$. The algorithm AnyBoost starts with the zero function F and iteratively finds a function $f \in \mathcal{F}$ to add to F so as to minimize the cost $C(F + \epsilon f)$ for some small ϵ. The new added function f is chosen such that the cost function is maximally decreased. The desired "direction" is the negative of the functional derivative of C at F, $-\nabla C(F)$, where

$$\nabla C(F)(x) := \frac{\partial C(F + \delta 1_x)}{\partial \delta}|_{\delta=0}$$

where 1_x is the indicator function of x. In general it is not possible to choose the new function as the negative of the gradient since we are restricted to picking from $\mathcal{F}$, so instead AnyBoost searches for f which maximizes the inner product $\langle f, -\nabla C(F) \rangle$

Most of the boosting algorithms use a cost function of the margin of points:

$$C(F) = \frac{1}{N} \sum_{i=1}^{N} c(y_i F(x_i))$$

where $c : \mathbb{R} \to \mathbb{R}^+$is a monotonically non-decreasing function. In this case, the inner product can be defined as

$$\langle f, g \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i) g(x_i) \tag{2.1}$$

So,

Algorithm 2 AnyBoost(C, S, T) [Mason et al. 2000b]

```
Requires:

• An inner product space $(\mathcal{X}, \langle , \rangle)$containing functions mapping $X$ to $Y$
• A class of base classifier $\mathcal{F}$
• A differentiable cost functional $C : lin(\mathcal{F}) \rightarrow \Re$
• A weak learner $\mathcal{L}(F)$ that accepts $F \in lin(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large
  value of $-\langle \nabla C(F), f \rangle$
• Input: $S = (x_1, y_1), ..., (x_N, y_N)$
• Input: $T$ is the number of iterations
    – Let $F_0(x) := 0$
    – for $t = 0$ to $T$ do
        ∗ Let $f_{t+1} := \mathcal{L}(F_t)$
        ∗ if $-\langle f, -\nabla C(F) \rangle \leq 0$
            · return $F_t$
        ∗ Choose $\alpha_{t+1}$
        ∗ Let $F_{t+1} = F_t + \alpha_{t+1} f_{t+1}$
    – return $F_{T+1}$
```

$$\langle -\nabla C(F), f\rangle = \frac{1}{N^2}\sum_{i=1}^{N} y_i f(x_i) c'(y_i F(x_i)) \tag{2.2}$$

So, maximizing $\langle -\nabla C(F), f\rangle$ is equivalent to minimizing the training error with examples weights, $D(i) \propto -c'(y_i F(x_i))$.

AdaBoost can be seen as a special case of AnyBoost with the cost function $c(yF(x)) = e^{-yF(x)}$and the inner product $\langle F(x), G(x)\rangle = \frac{1}{N}\sum_{i=1}^{N} F(x_i)G(x_i)$. Many of the most successful voting methods are special cases of AnyBoost with the appropriate cost function and step size.

Table 2.1: Voting Methods Seen as Special Cases of AnyBoost

Algorithm	Cost Function	Step Size
AdaBoost	$e^{-yF(x)}$	Line Search
ARC-X4	$(1 - yF(x))^5$	$1/t$
LogitBoost	$ln(1 + e^{-yF(x)})$	Newton-Rapson

2.4 Margin Bounds

The most popular explanation for the performance of AdaBoost is that it tends to produce classifiers which have large margins on the training points. This has lead to a great deal of development in algorithms which can optimize the margins [Boser et al. 1992, Demiriz et al. 2002, Grove and Schuurmans 1998, Li et al. 2003, Mason et al. 2000a]. Schapire et al. [1997] introduced the margin explanation and provided a bound for the generalization performance of the AdaBoost solution. Theorem 1 bounds the generalization performance by the fraction of training points which have a small margin and a complexity term. This was improved by Koltchinskii and Panchenko [2002]

Theorem 2 *With probability at least* $1 - \epsilon$ *,the following bound holds for all* $F \in$ *conv*$(\mathcal{H})$

$$P_{\mathcal{D}}[yF(x) \leq 0] \leq \inf_{\delta \in (0,1]} \left[P_S[yF(x) \leq \delta] + \frac{C}{\delta}\sqrt{\frac{V(\mathcal{H})}{n}} + \left(\frac{\log\log_2(2\delta^{-1})}{n}\right)^{1/2}\right] + \sqrt{\frac{1}{2n}\log\frac{2}{\epsilon}}$$

Experimentally it has been observed that this bound is very loose. It has been shown that as long as the VC-Dimension of the base classifier used in the boosting process is small, the margin bound can be improved. Koltchinskii et al. [2000a] introduced a new class of bounds called the γ-bounds. If the base classifiers belong to a model with a small random entropy, then the generalization performance can be further bound based on the growth rate of the entropy. Given a metric space $(\mathcal{F}, d)$, the $\epsilon-$entropy of $\mathcal{F}$, denoted by $H_d(\mathcal{F};\epsilon)$ is defined as $\log N_d(\mathcal{F};\epsilon)$, where $N_d(\mathcal{F};\epsilon)$ is the minimum number of $\epsilon-$balls covering $\mathcal{F}$. The $\gamma-$margin $\delta(\gamma; f)$ and the corresponding empirical γ-margin $\hat{\delta}(\gamma; f)$ are defined as

$$\delta(\gamma; f) = \sup \delta \in (0,1) : \delta^{\gamma} P\{yf(x) \leq \delta\} \leq n^{-1+\gamma/2}$$

$$\hat{\delta}(\gamma; f) = \sup \delta \in (0,1) : \delta^{\gamma} P_S\{yf(x) \leq \delta\} \leq n^{-1+\gamma/2}$$

Theorem 3 *Suppose that for some $\alpha \in (0,2)$ and some constant D*

$$H_{d_{P_n,2}}(conv(\mathcal{F}; u) \leq Du^{-\alpha}, u > 0\, a.s.$$

then for any $\gamma \geq \frac{2\alpha}{2+\alpha}$, for some constants $A, B > 0$ and for large enough n

$$P\left[\forall f \in \mathcal{F} : A^{-1}\hat{\delta}_n(\gamma; f) \leq \delta_n(\gamma; f) \leq A\hat{\delta}_n(\gamma; f)\right] \geq 1 - B(\log_2\log_2 n)\exp -n^{\gamma/2}/2$$

The random entropy of the convex hull of a model can be bounded in terms of its VC-dimension.

$$H_{d_{P_n,2}}(conv(\mathcal{H}; u) \leq \sup_{Q \in \mathcal{P}(S)} H_{d_{Q,2}}(conv(\mathcal{H}; u) \leq Du^{-\frac{2V-1}{V}}$$

From Theorem 3, it can be deduced that with high probability, for all $f \in \mathcal{F}$,

$$P\left[yf(x) \leq 0\right] \leq c\left(n^{1-\gamma/2}\hat{\delta}(\gamma; f)^{\gamma}\right)^{-1}$$

where $\gamma \geq \frac{2(V-1)}{2V-1}$.

The bound is the weakest for $\gamma = 1$, and at this value it reduces to the bound in Theorem 2. Smaller values of γ give a better generalization bound. So, if the base classifier has a very low VC-dimension, then this gives a much tighter bound on the generalization performance. It has been observed empirically, that the bound holds for smaller values of γ than what has been proved. The conjecture is that the classifiers produced by boosting belong to a subset of the convex hull, which has a smaller random entropy than the whole convex hull.

Koltchinskii et al. [2000a] provided new bounds for the generalization performance which uses the *dimensional complexity* of the generated classifier as a measure of complexity. The dimensional complexity measures how fast the weights given to the hypothesis decrease in the ensemble. For a function $F \in Conv(\mathcal{F})$, the approximate Δ-dimension is defined as the integer $d \geq 0$ such that there exists $N \geq 1$, functions $f_j \in \mathcal{F}$,$j = 1, .., N$ and numbers $\lambda_j \in \Re$, satisfying $F = \sum_{j=1}^{N} \lambda_j f_j$, $\sum_{j=1}^{N} |\lambda_j| = 1$ and $\sum_{j=d+1}^{N} |\lambda_j| \leq \Delta$. The Δ-dimension of F is denoted as $d(F, \Delta)$.

Theorem 4 *If $\mathcal{H} \subset Conv(\mathcal{F})$ is a class of functions such that for some $\beta > 0$*

$$\sup_{F \in \mathcal{H}} d(F, \delta) = O(\Delta^{-\beta})$$

then with high probability, for any $h \in \mathcal{H}$, the generalization error can be bounded by

$$\frac{1}{n^{1-\gamma\beta/2(\gamma+\beta)}\hat{\delta}(h)^{\gamma\beta/(\gamma+\beta)}}$$

This bound reduces to the one in Theorem 3 for $\beta = \infty$,and for smaller values of β, provides a tighter bound on the generalization performance.

The proof and discussion about these bounds can be found in Koltchinskii et al. [2000a;b] and Koltchinskii and Panchenko [2002]

2.5 AlphaBoost

In this section, we introduce an algorithm which produces an ensemble with much lower value for the cost function than AdaBoost. AlphaBoost improves on AnyBoost by minimizing the cost function more aggressively. It can achieve significantly lower values of cost function much faster than AnyBoost. AnyBoost is an iterative algorithm which is restricted to adding new hypothesis and the corresponding weight to the ensemble. It can not go back and change the weights it had assigned to a hypothesis.

2.5.1 Algorithm

AlphaBoost starts out by calling AnyBoost to obtain a linear combination of hypotheses from the base learner. It then optimizes the weights given to each of the classifier in the combination to further reduce the cost function. This is done by doing a conjugate gradient descent [Fletcher and Reeves 1964] in the weight space. Algorithm 3 gives the pseudo-code of AlphaBoost. We used conjugate gradient instead of normal gradient descent because it uses the second-order information of the cost function and so the cost is reduced faster.

The first step is to visualize the cost function as a function of the weight of the hypotheses, rather than of the aggregate hypothesis. So once we've fixed the base learners which will vote to form the aggregate hypothesis, we have a cost function $C(\alpha)$ which depends on the weight each hypothesis gets in the aggregate. We can then do a conjugate gradient descent in the weight space to minimize the cost function and thus find the optimal weights.

Suppose AnyBoost returns $F_T(x)=\sum_{i=1}^{T} \alpha_i h_i(x)$ as the final hypothesis. Then we have,

$$C(\alpha) = \sum_{i=1}^{n} c(y_i F_T(x_i)) \tag{2.3}$$

and so, the gradient can be computed as

$$\frac{\partial C(\alpha)}{\partial \alpha_t} = \sum_{i=1}^{n} y_i h_t(x_i) c'(y_i F_T(x_i)) \tag{2.4}$$

So the descent direction at stage t is computed as $d_t = -\nabla C(\alpha_t)$. Instead of using this direction directly, we choose the search direction as

$$d_t = -\nabla C(\alpha_t) + \beta_t d_{t-1} \tag{2.5}$$

where $\beta_t \in \Re$ and d_{t-1} is the last search direction. The value of β_t controls how much of the previous direction affects the current direction. It is computed using the Polak-Ribiere formula

$$\beta_t = \frac{\langle d_t, d_t - d_{t-1} \rangle}{\langle d_{t-1}, d_{t-1} \rangle} \tag{2.6}$$

Algorithm 3 uses a fixed step size to perform conjugate descent. We can also do a line search to find an optimal step size at each iteration.

2.5.2 Generalization Performance on Artificial Datasets

For analyzing the out-of-sample performance of AlphaBoost, we used AdaBoost's exponential cost function. Once we fix a cost function, AnyBoost, which reduces to AdaBoost in this case, and AlphaBoost are essentially two algorithms which try to optimize the same cost function.

To compare the generalization performance of AlphaBoost, we used the Caltech Data Engine to generate random target functions. The Caltech Data Engine [Pratap 2003] is a computer program that contains several predefined data models, such as neural networks, support vector machines (SVM), and radial basis functions (RBF). When requested for data, it randomly picks a model, generates (also randomly) pa-

Algorithm 3 AlphaBoost(C, S,T, A)

```
Requires:

• An inner product space $(\mathcal{X}, \langle,\rangle)$containing functions mapping $X$ to $Y$
• A class of base classifier $\mathcal{F}$
• Input: $S = (x_1, y_1), ..., (x_N, y_N)$
• Input : T the number of iterations of AnyBoost and $A$ is the number of conjugate
  gradient steps
• Input: $C$ is a differentiable cost functional $C : lin(\mathcal{F}) \rightarrow \Re$
    – Let $F_T(x)$=$\alpha.H(x)$ be the output of AnyBoost($C$,$S$,$T$)
    – $d_0 = 0$ and $\alpha_0 = \alpha$
    – For $t = 1$ to $A$
        * $d_t = -\nabla C(\alpha_t)$
        * $\beta_t = \frac{\langle d_t, d_t - d_{t-1}\rangle}{\langle d_{t-1}, d_{t-1}\rangle}$
        * $d\alpha_t = -d_t + \beta_t d\alpha_{t-1}$
        * $\alpha_t = \alpha_{t-1} + \eta d\alpha_{t-1}$
```

rameters for that model, and produces random examples according to the generated model. A complexity factor can be specified which controls the complexity of the generated model. The engine can be prompted repeatedly to generate independent data sets from the same model to achieve small error bars in testing and comparing learning algorithms.

The two algorithms were compared using function of varying complexity and from different models. For each target, 100 independent training sets of size 500 were generated. The algorithms were tested on an independently generated test set of size 5000. AlphaBoost was composed of 100 steps of AdaBoost, followed by 50 steps of conjugate gradient with line search, and it was compared with AdaBoost running for 150 iterations. In all the runs, AlphaBoost obtained significantly lower values of cost function. However, the out-of-sample performance of AdaBoost was significantly better.

Figure 2.1 shows the final cost and out-of-sample error obtained by the two algorithms on different targets generated from the data engine. The error bar on all the runs was less than 10^{-3} of the values, and so are not shown in the figures.

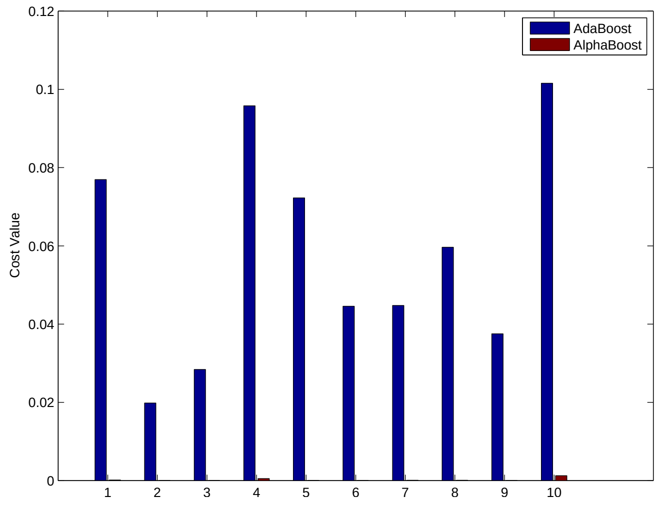
The cost function and out-of-sample error at each iteration for one such run is shown in Figure 2.2

The cost function at the end of AlphaBoost is significantly lower than the cost function at the end of AdaBoost. However, the out-of-sample error achieved by AdaBoost is significantly lower.

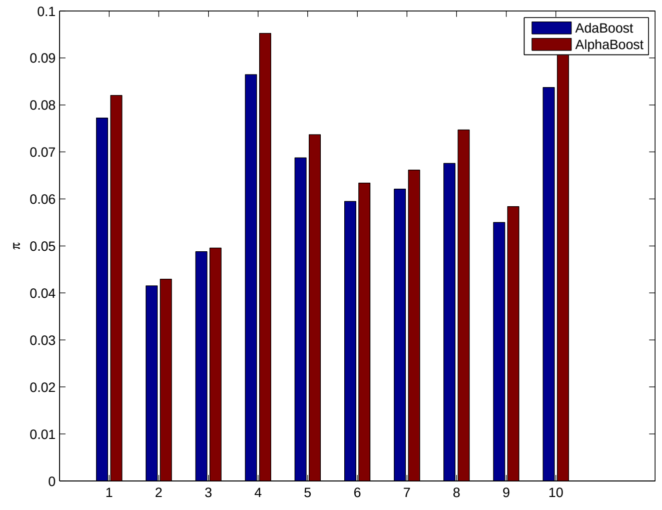
2.5.3 Experimental Results on Real World Datasets

We tested AlphaBoost on six datasets from the UCI machine learning repository[Blake and Merz 1998]. The datasets were the Pima Indians Diabetes Database; sonar database; heart disease diagnosis database from V.A. Medical Center, Long Beach; and Cleveland Clinic Foundation collected by Robert Detrano, M.D., Ph.D. Johns Hopkins University Ionosphere database, 1984; United States Congressional Voting Records Database; and breast cancer databases from the University of Wisconsin Hospitals [Mangasarian and Wolberg 1990]. Decision stumps were used as the base learner in all the experiments. For the experiments, the dataset was randomly divided into two sets of size 80% and 20% and they were used for training and testing the algorithms. AlphaBoost was composed of 100 steps of AdaBoost, followed by 50 steps of conjugate gradient with line search as in the previous section and it was compared with AdaBoost running for 150 iterations. All the results were averaged over 50 runs.

Table 2.2 shows the final values obtained by the two algorithms. As expected, AlphaBoost was able to achieve significantly lower value of the cost function. Though AdaBoost achieved better out-of-sample error than AlphaBoost, the error bars are too high in these limited datasets to make any statistically significant conclusions. Figures 2.3–2.8 show the cost function value and out-of-sample error as a function of the number of iterations. For the first 100 iterations, AdaBoost and AlphaBoost perform exactly the same, so the curves coincide for this period. The cost function takes a steep dip around iteration 101 for AlphaBoost when it starts doing conjugate

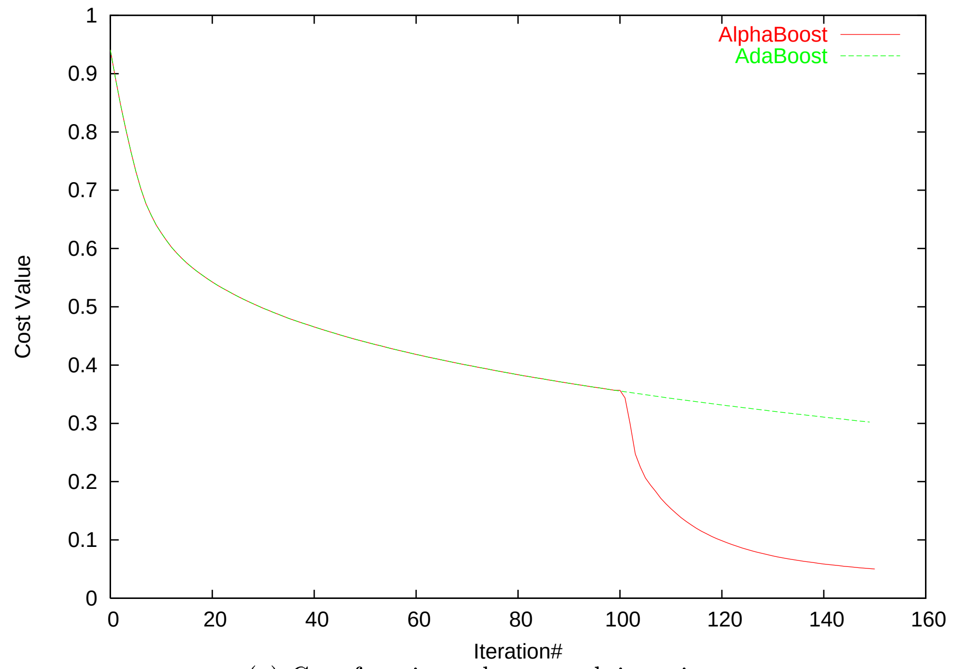


(a) Cost function value obtained by AdaBoost and AlphaBoost

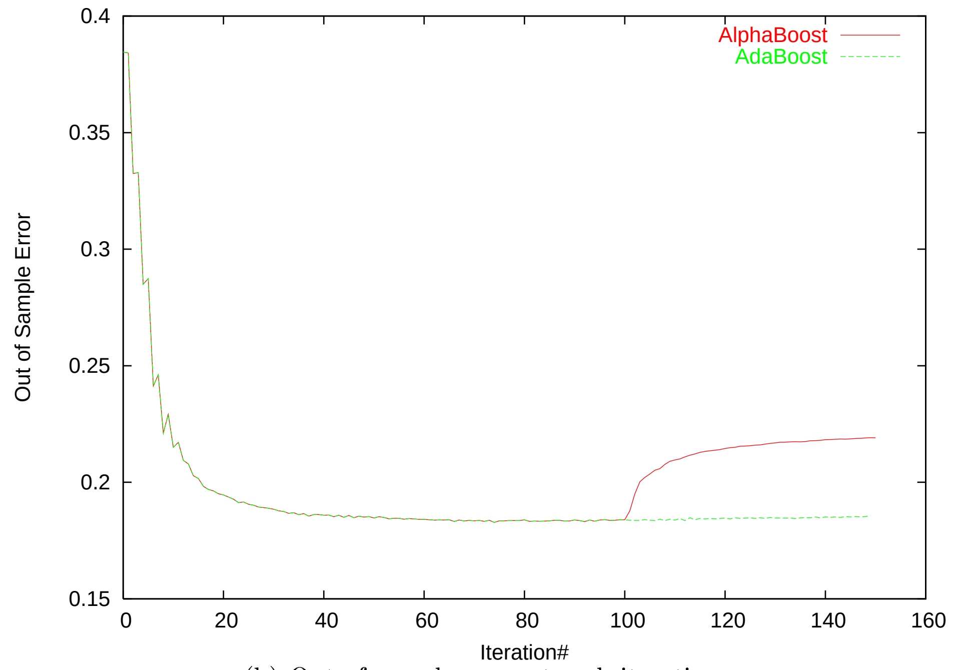


(b) Out-of-sample error obtained by AdBoost and AlphaBoost

Figure 2.1: Performance of AlphaBoost and AdaBoost on a Target Generated by DEngin



(a) Cost-function value at each iteration



(b) Out-of-sample error at each iteration

Figure 2.2: Performance of AlphaBoost and AdaBoost on a Typical Target

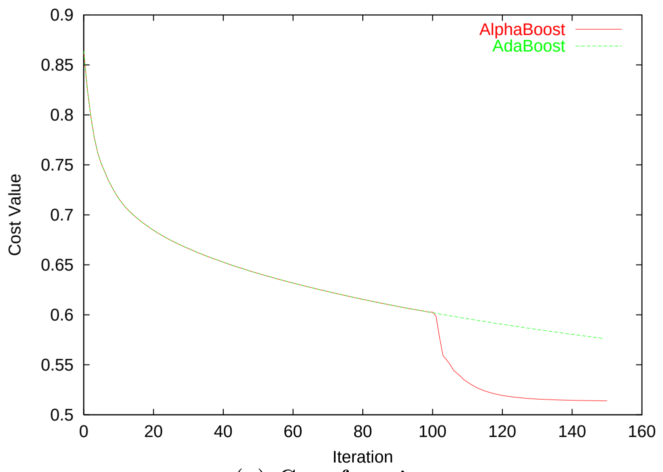
Table 2.2: Experimental Results on UCI Data Sets

Data Set	Algorithm	Cost	π	$\min(\Delta)$	$\overline{\Delta}$
Pima Indian	AlphaBoost	**0.5139**	0.2638	**−0.0979**	0.0658
	AdaBoost	0.5763	**0.2463**	−0.1576	**0.1023**
Sonar	AlphaBoost	**0**	0.1833	**0.1099**	**0.2128**
	AdaBoost	0.0003	**0.167**	0.1088	0.2099
Cleveland	AlphaBoost	**0.0386**	0.2421	**−0.183**	0.0979
	AdaBoost	0.0651	**0.2047**	−0.0747	**0.1366**
Ionosphere	AlphaBoost	**0**	0.1	**0.0652**	0.1849
	AdaBoost	0.0094	**0.0894**	0.0459	**0.1886**
Vote	AlphaBoost	**0.3576**	0.2222	**−0.1359**	0.2724
	AdaBoost	0.3756	**0.2155**	−0.1757	**0.3104**
Cancer	AlphaBoost	**0.0015**	0.0483	**−0.0052**	0.2377
	AdaBoost	0.0509	**0.0419**	−0.0511	**0.2638**

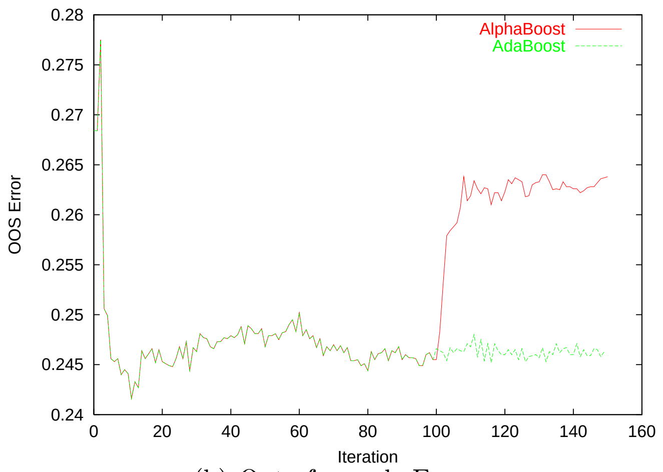
gradient descent. The out-of-sample error increases during this stage. Also shown are the distribution of the margins at the end of 100 iterations of AdaBoost, and at the end of 100 iterations of AdaBoost and 50 iterations of conjugate gradient descent. The distribution of the margins at the end of 100 iterations of AdaBoost is the starting point for both the algorithms, and from that point forward, they do different things for the next 50 iterations. AlphaBoost tends to maximize the minimum margin better than AdaBoost at the expense of lowering the maximum margin.

2.5.4 Discussion

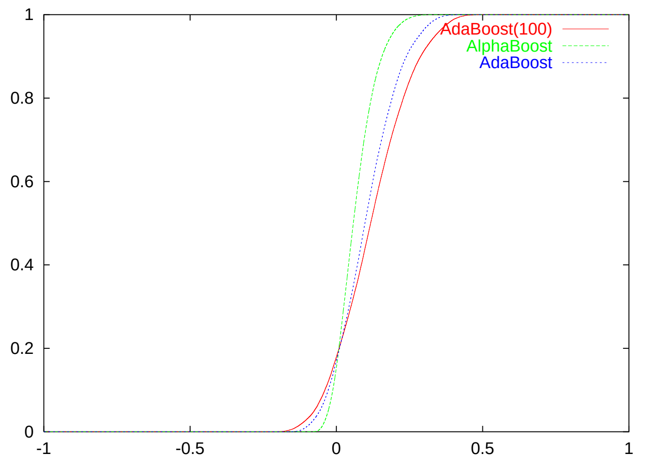
The results of this section indicate that aggressively optimizing the cost function in boosting leads to worsening of the performance. The AlphaBoost ensemble uses the exact same weak hypothesis as the AdaBoost ensemble, so there is no additional complexity that was introduced by the alpha-descent step. The only explanation for this result is that the soft-min cost function tends to overfit. These results agree with the findings of Li et al. [2003], where conjugate descent is used, but there the hypotheses used are different from that of AdaBoost. This, however, does not refute the margin theory explanation, as we are optimizing on one particular cost function of the margin. The margin explanation relies on the reasoning that large margins are



(a) Cost function

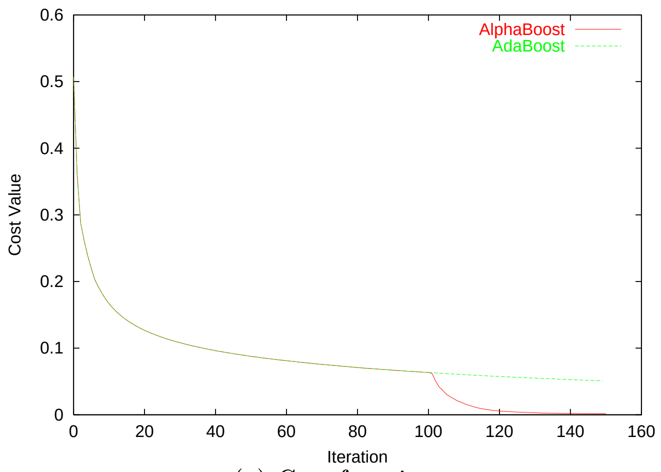


(b) Out-of-sample Error

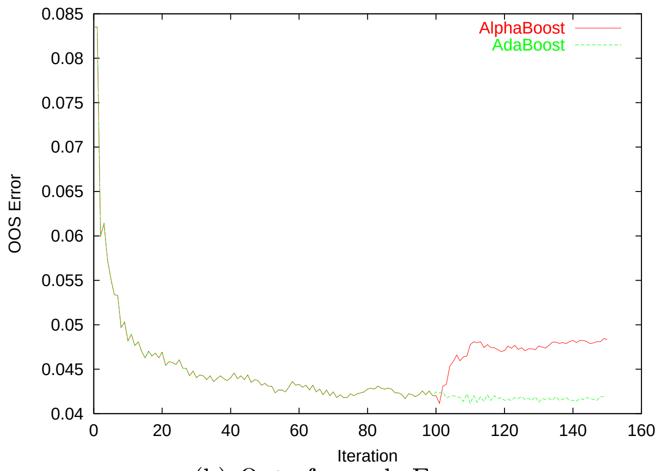


(c) Margin CDF

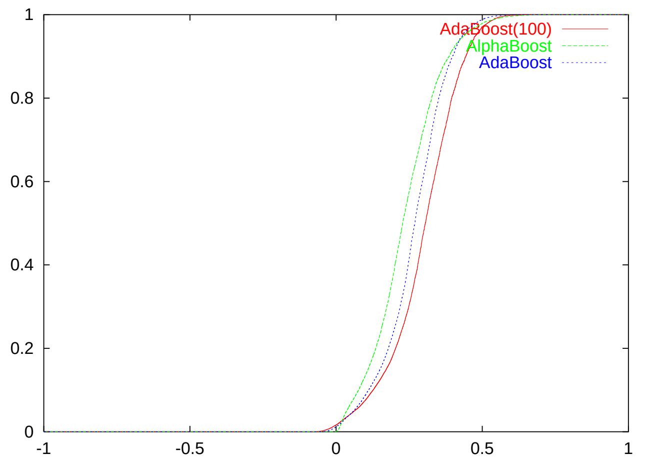
Figure 2.3: Pima Indian Diabetes DataSet



(a) Cost function

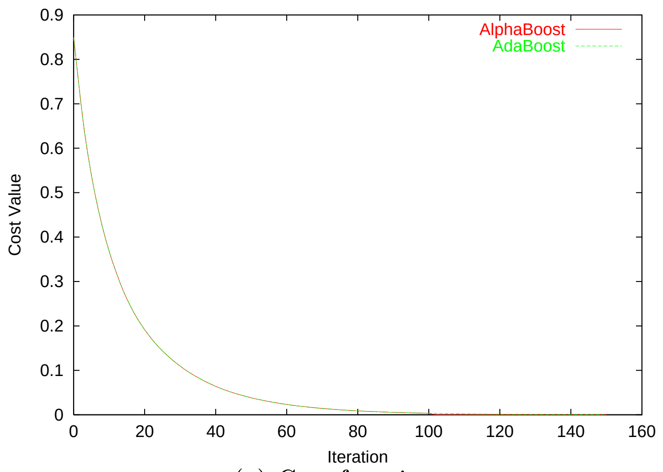


(b) Out-of-sample Error

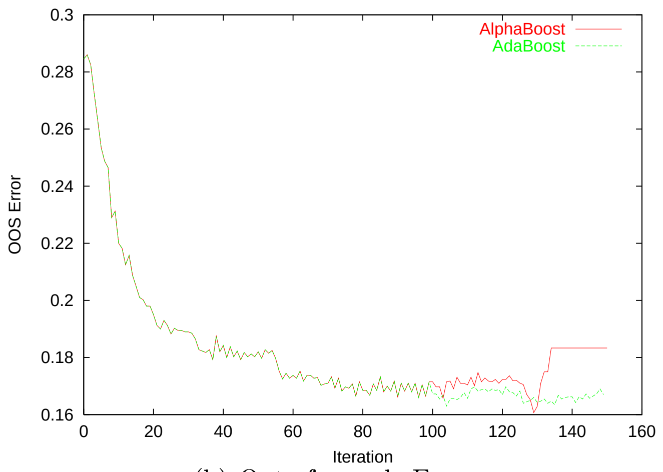


(c) Margin CDF

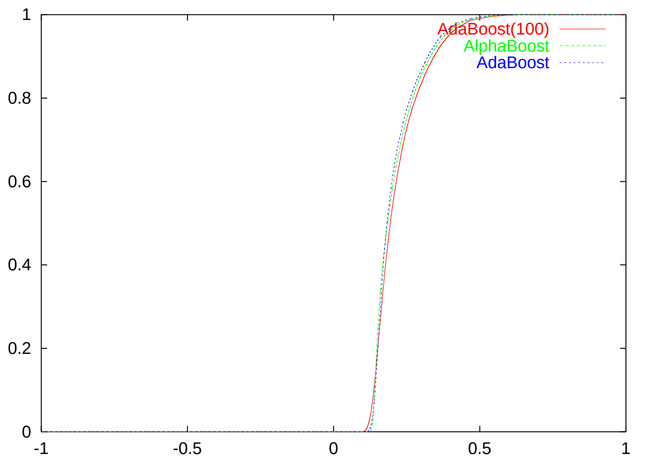
Figure 2.4: Wisconsin Breast DataSet



(a) Cost function

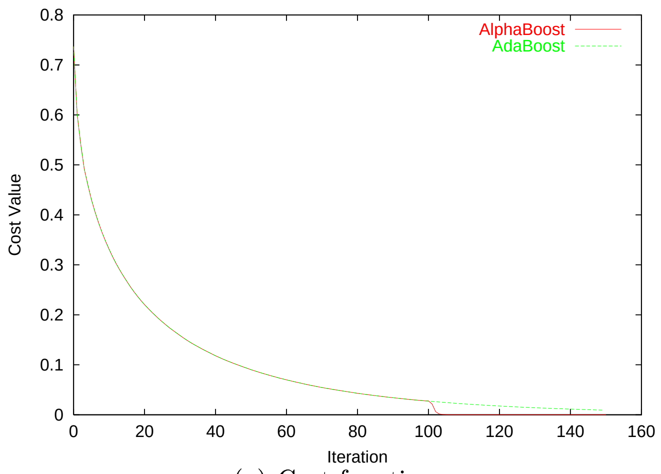


(b) Out-of-sample Error

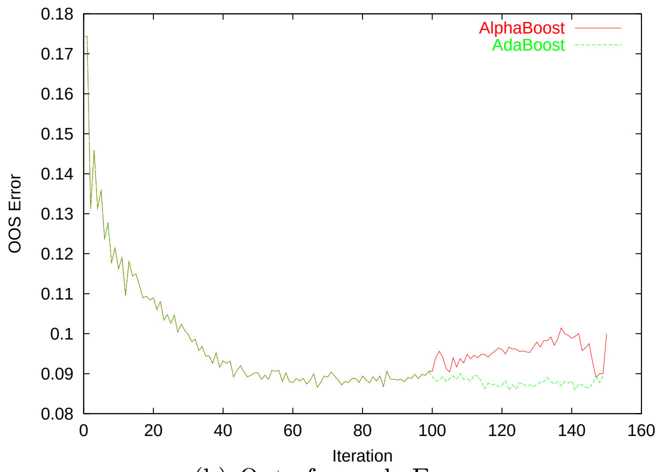


(c) Margin CDF

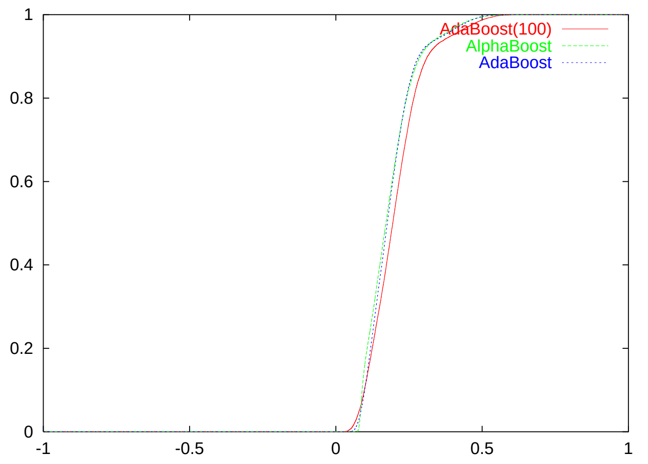
Figure 2.5: Sonar DataSet



(a) Cost function

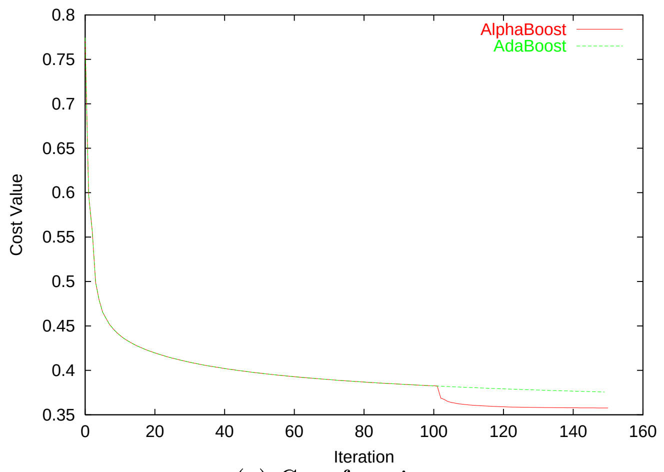


(b) Out-of-sample Error

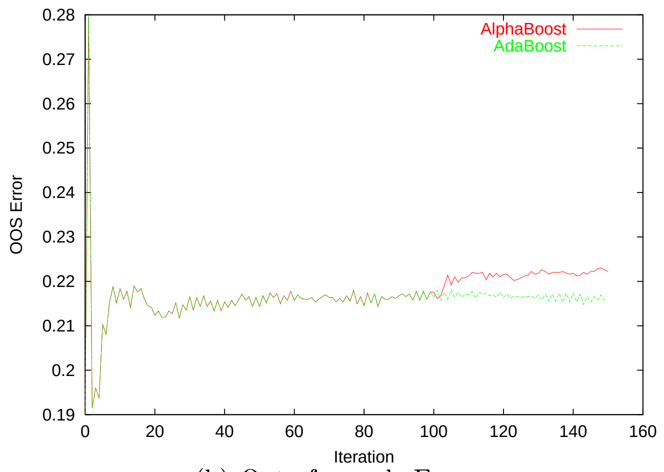


(c) Margin CDF

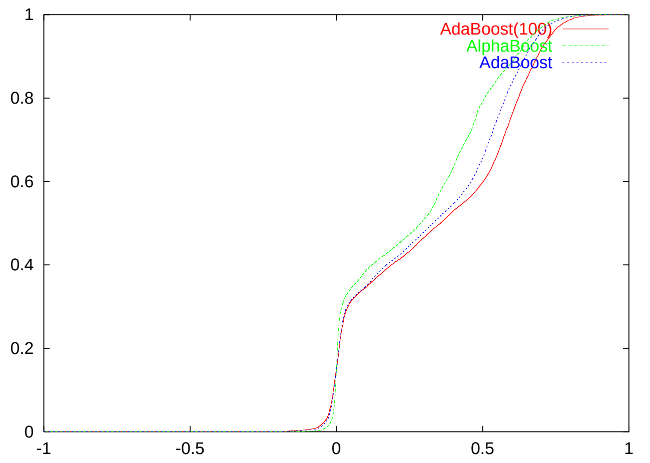
Figure 2.6: IonoSphere DataSet



(a) Cost function

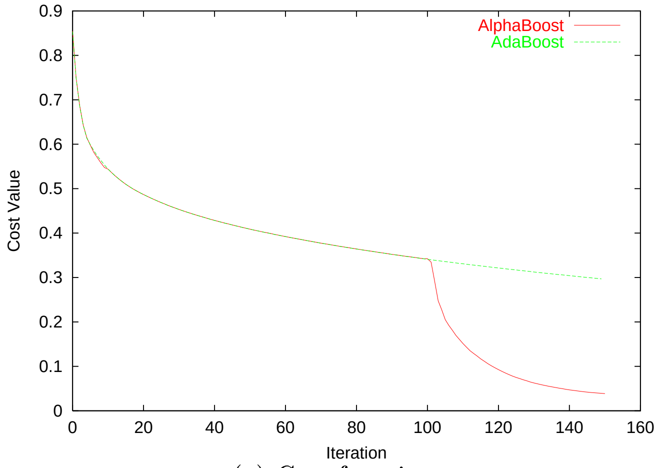


(b) Out-of-sample Error

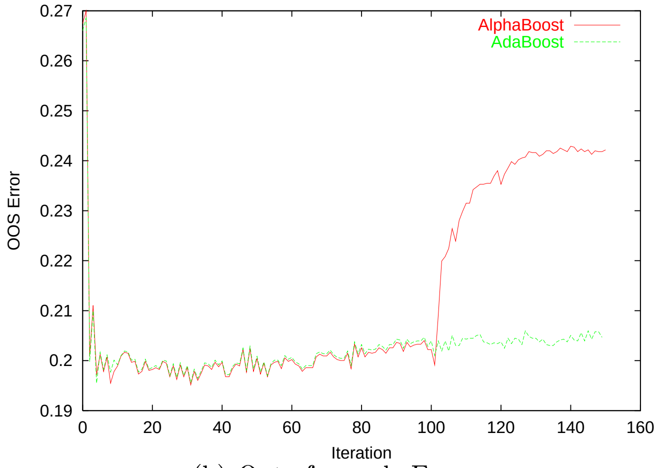


(c) Margin CDF

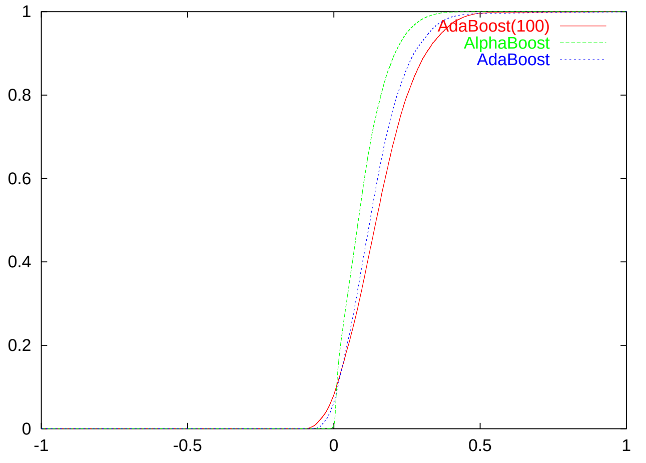
Figure 2.7: Votes84 Dataset



(a) Cost function



(b) Out-of-sample Error



(c) Margin CDF

Figure 2.8: Cleveland Heart Dataset

good for generalization. If we can show that optimizing any possible cost function of the margin would lead to bad generalization, this would refute the margin explanation.

2.6 DLPBoost

DLPBoost is an extension of AdaBoost which optimizes the margin distribution produced by AdaBoost. It has been expressly designed to test the margin explanation for the performance of AdaBoost. We have seen in the case of AlphaBoost that decreasing the cost function leads to larger minimum margin, but at the expense of decreasing the margin of some of the points. In most of the extensions and variants of AdaBoost, the margin distribution produced does not dominate the margin distribution of AdaBoost. We say that a distribution P_1 dominates another distribution P_2 if $P_1(t) \leq P_2(t), \forall t$. As a result, there are some cost functions of the margin that would be worsened by the variant. The only algorithm which could produce a better margin distribution was Arc-gv [Breiman 1998]. This algorithm was, however, criticized for producing complex hypotheses and hence the increase in error was attributed to the additional complexity used [Reyzin and Schapire 2006]. This is an important consideration in the design of DLPBoost. We want to make sure that the difference in the solutions of AdaBoost and DLPBoost can only be accounted for by the difference in the margins. It is therefore crucial to keep all other factors constant.

In DLPBoost, we provide an algorithm which consistently improves the entire distribution of the margins. This condition ensures that DLPBoost has a better margin cost than AdaBoost for any possible definition of margin cost. Thus, all the margin bounds for AdaBoost would hold for DLPBoost, and as the complexity terms are not changed, the bounds become tighter. This would lead to better performance if the margin explanation is true.

2.6.1 Algorithm

DLPBoost uses the AdaBoost solution as the starting point and optimizes the maximum margin subject to constraints on the individual margins. The general problem of

Algorithm 4 DLPBoost

```
Given $(x_1, y_1), ..., (x_N, y_N)$
Run AdaBoost/AnyBoost for $T$ iterations to get $(h_1, ..., h_T)$ and $(\alpha_1^*, ..., \alpha_T^*)$
Define $m(\alpha, i) = \sum_{t=1}^{T} y_i \alpha_t h_t(x_i)$
Solve the linear programming problem
                    $\max_\alpha \sum_{i=1}^{N} m(\alpha, i)$
                    Subject to constraints
                    $m(\alpha, i) \geq m(\alpha^*, i)$
                    $\sum \alpha_t = 1, \alpha_t \geq 0$
```

improving the margin distribution is very hard, optimization-wise, as it also involves a combinatorial component. So, instead we set up a restricted problem in which we require that the margin on none of the points is decreased. Any solution of the latter problem would be a feasible solution for the original problem, though it might not be optimal. Our goal here is to produce better margin distribution, and the restricted setup ensures that any solution we produce would have margin distributions at least as good as those of the AdaBoost solution.

Algorithm 4 illustrates DLPBoost. The base classifiers produced by DLPBoost are the exact same as the one produced by AdaBoost, and, by design, the margin of each training example is not reduced. Thus, by definition, DLPBoost does not “cheat” on the complexity and produces a solution which dominates (or at least equals) the margin distribution produced by AdaBoost.

2.6.2 Properties of DLPBoost

DLPBoost uses linear programming to find new weights for the hypotheses which would maximize the average margin, while ensuring that all the training examples have a margin at least as large as that of the AdaBoost solution. The weights generated by AdaBoost, α^*, are a feasible solution to the optimization problem in DLPBoost. The simplex procedure which is used to solve the optimization starts with a zero solution, i.e., all the weights are initially zero. It then makes each of the weights non-zero, one at a time, until it finds the optimal solution. One interesting byproduct of this procedure is that the final solution produced by DLPBoost would have a smaller number of hypotheses with non-zero weights. This is a very useful feature, both theoretically and practically. Theoretically it means that the DLPBoost solution has a smaller dimensional complexity than the AdaBoost solution. In many practical situations, like visual recognition, where real-time processing of the data is critical, the DLPBoost solution can be useful as it tends to have a smaller ensemble size.

One of the interesting aspects of the DLPBoost setup is that the function being optimized is ad hoc. The important thing is to find a non-trivial feasible solution to the constraints. The trivial solution in this case would be the AdaBoost solution in which all the inequalities are exactly satisfied. We have used the average margin as the goal, but we can use any general linear combination of the margins as the goal and still have a working setup. The most general setup would be a Multi-objective optimization setup, in which we can maximize the margin on each point. There are popular algorithms [Deb et al. 2002] to solve this kind of problem but we do not use those, as the added advantage of a sparse solution generated by linear programming is very appealing in our case. In addition, any non-trivial feasible solution would be an added improvement on the margin distribution and serve the purpose of our analysis.

2.6.3 Relationship to Margin Theory

All the bounds discussed in section 2.4 hold for DLPBoost. For DLPBoost, by definition

$$\forall \delta P_S[yF_D(x) \leq \delta] \leq P_S[yF_A(x) \leq \delta]$$

(where F_D is the DLPBoost solution and F_A is the AdaBoost solution). Hence we have that the bound on the generalization error of DLPBoost is smaller than the bound on the generalization error of AdaBoost.

The setup of DLPBoost is a very optimistic one. It is a very constrained optimization problem, as the number of constraints is of the the order of number of examples in the dataset. An improvement in all the margins would indicate that the optimization procedure used in AdaBoost is very inefficient. The design of DLPBoost ensures that we do not introduce any additional complexity in the solution. Arc-GV, which can also be used to generate ensembles with better margin distribution, has been criticized for using additional hypothesis complexity [Reyzin and Schapire 2006]. It also uses hypotheses which are different from those used by AdaBoost, and so it is plausible that the solution generated is more complicated, thus any negative results there might not contradict the margin explanation. In DLPBoost, we have isolated the margin explanation and any difference in performance can only be accounted for by the difference in the margin distributions. This would give us a fair evaluation of the margin explanation.

2.6.4 Experiments with Margins

We study the behavior of the margins and the solution produced by DLPBoost. Figures 2.9, 2.10 and 2.11 show the margins and the weights on the hypotheses obtained by AdaBoost and DLPBoost on the WDBC dataset. In this setup, we weight the margins of the points in the linear programming setup by their cost function. This weighing promotes the points with smaller margin. The out-of-sample errors for AdaBoost was 2.74% and for DLPBoost was 3.75%. The margin distribution for DLPBoost is clearly much better than that of AdaBoost, and from the final weights obtained, we can see that the hypotheses removed from the AdaBoost solution by DLPBoost had significant weights assigned to them. Figure 2.10 shows the weights assigned by the two algorithms to the weak hypotheses which are indexed by the iteration number. It can be seen that DLPBoost emphasizes some hypotheses which were generated late in AdaBoost training and it also killed some of the hypotheses which were generated early. AdaBoost is a greedy iterative algorithm. This kind of behavior is expected, as AdaBoost has no way of going back and correcting the weights of hypotheses already generated. Note that the one-norm of both the solutions is normalized to one, and so the DLPBoost solution having a smaller number of hypotheses (64, compared to the 168 of AdaBoost) has overall larger weights and this result is for a single run.

Figures 2.12, 2.13 and 2.14 show the same results for one run on the Australian

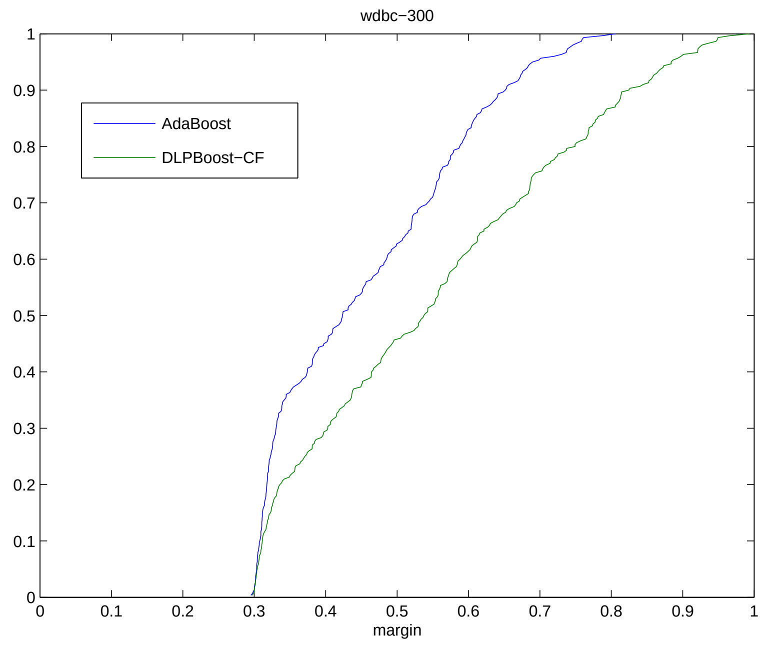


Figure 2.9: Margins Distribution for WDBC Dataset with 300 Training Points

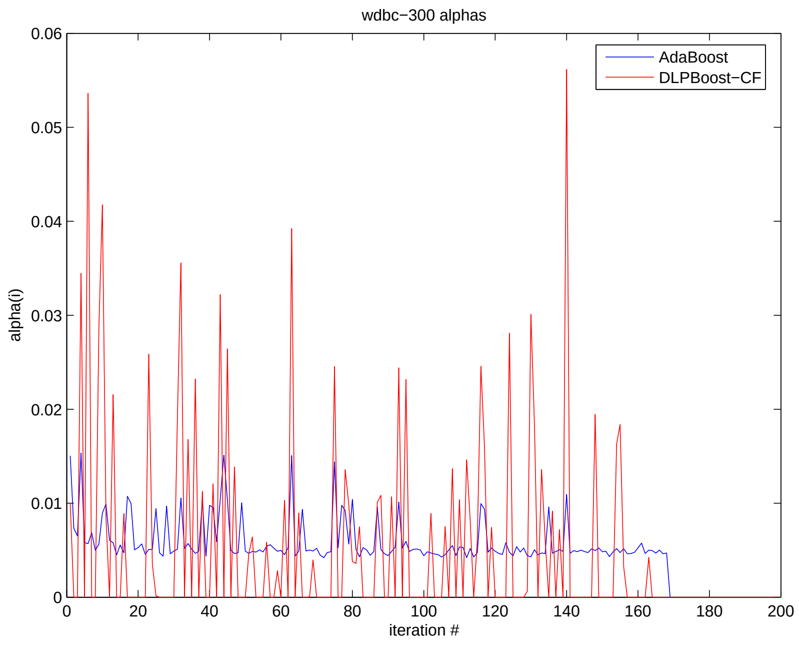


Figure 2.10: α for WDBC Dataset with 300 Training Points

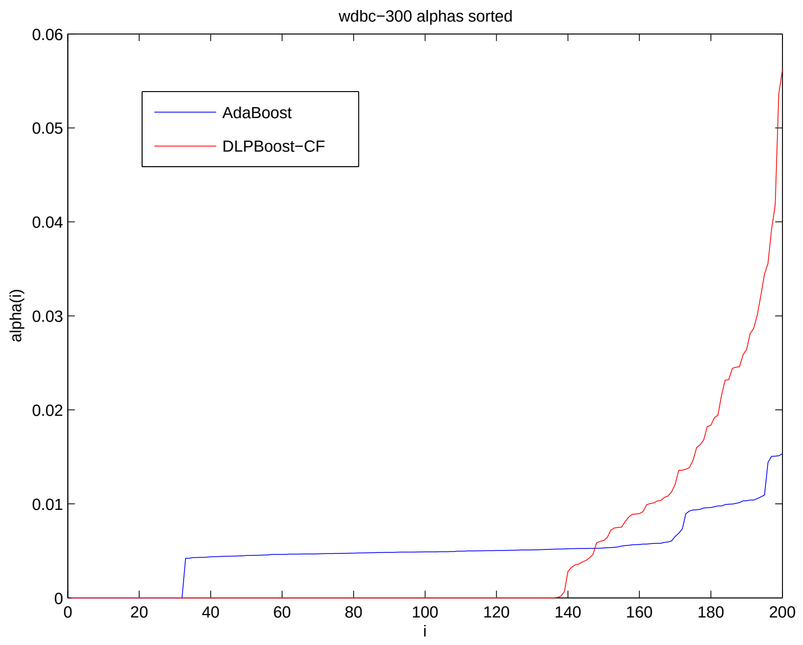


Figure 2.11: α Distribution for WDBC Dataset with 300 Training Points

dataset. Here the errors for both the algorithms were 18.8% while the number of hypotheses was reduced from 423 to 201. In this case, there is very small improvement in the margin distribution and the out-of-sample error did not change. The hypotheses killed by the DLPBoost step had significant weights.

This points to an inverse relationship between the margins and the out-of-sample performance. In the next two sections, we will investigate this relationship in detail and give statistically significant results.

2.6.5 Experiments with Artificial Datasets

We test the performance of DLPBoost and compare it with that of AdaBoost on a few artificial datasets.

2.6.5.1 Artificial Datasets Used

We used the following artificial learning problems for evaluation the performance of the cloning process.

Yin-Yang is a round plate centered at $(0, 0)$ and is partitioned into two classes. The

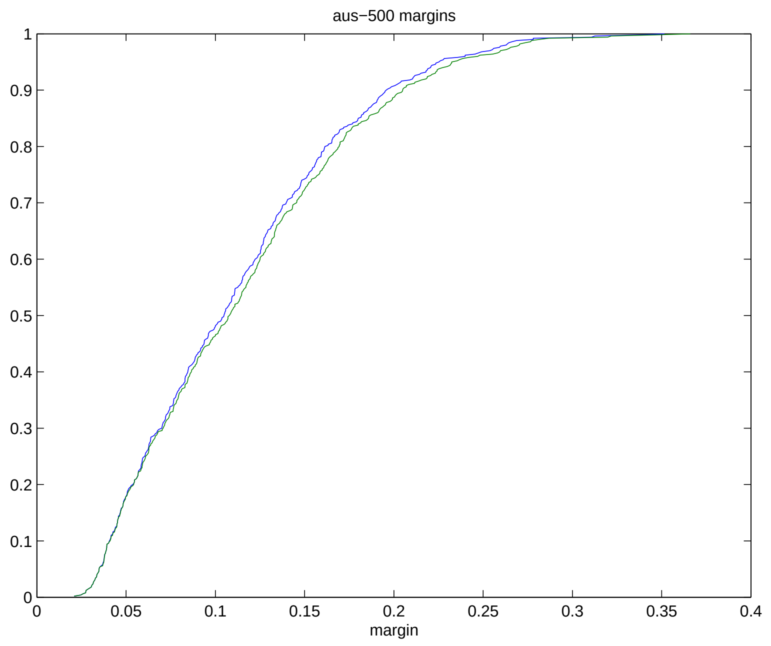


Figure 2.12: Margins Distribution for Australian Dataset with 500 Training Points

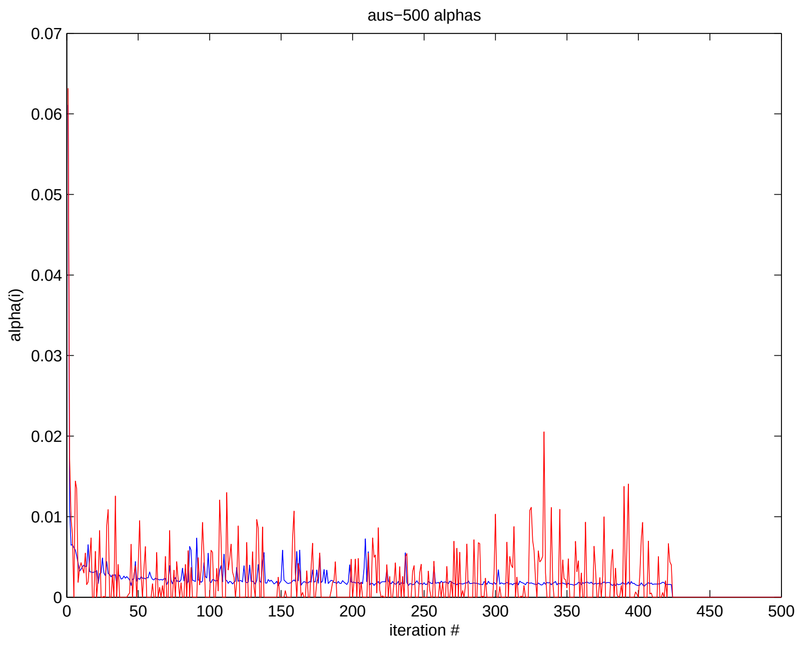


Figure 2.13: α for Australian Dataset with 500 Training Points

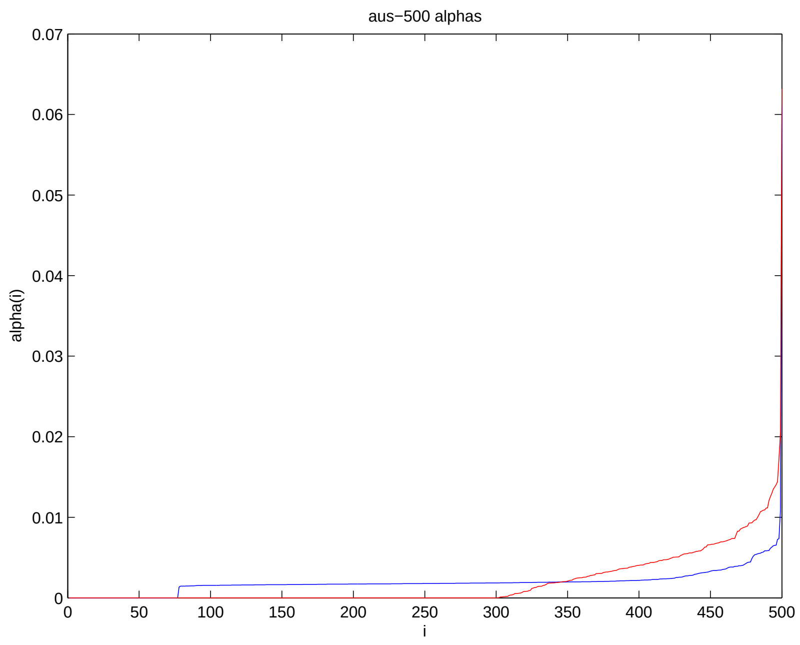


Figure 2.14: α Distribution for Australian Dataset with 500 Training Points

first class includes all points (x_1, x_2) which satisfies

$$(d_+ \leq r) \vee \left(r < d_- \leq \tfrac{R}{2}\right) \vee \left(x_2 > 0 \wedge d_+ > \tfrac{R}{2}\right),$$

where the radius of the plate is $R = 1$ and the radius of the two small circles is $r = 0.18$, $d_+ = \sqrt{(x_1 - \frac{R}{2})^2 + x_2^2}$ and $d_- = \sqrt{(x_1 + \frac{R}{2})^2 + x_2^2}$.

LeftSin ([Merler et al. 2004]) partitions $[-10, 10] \times [-5, 5]$ into two class regions with the boundary

$$x_2 = \begin{cases} 2\sin 3x_1, & \text{if } x_1 < 0; \\ 0, & \text{if } x_1 \geq 0. \end{cases}$$

TwoNorm ([Breiman 1996]) is a 20-dimensional, 2-class classification dataset. Each class is drawn from a multivariate normal distribution with unit variance. Class 1 has mean $(a, a, ..a)$ while Class 2 has mean $(-a, -a, .. -a)$, where $a = 2/\sqrt{20}$.

The artificial learning problems can be used to generate independent datasets from the same target to achieve statistically significant results.

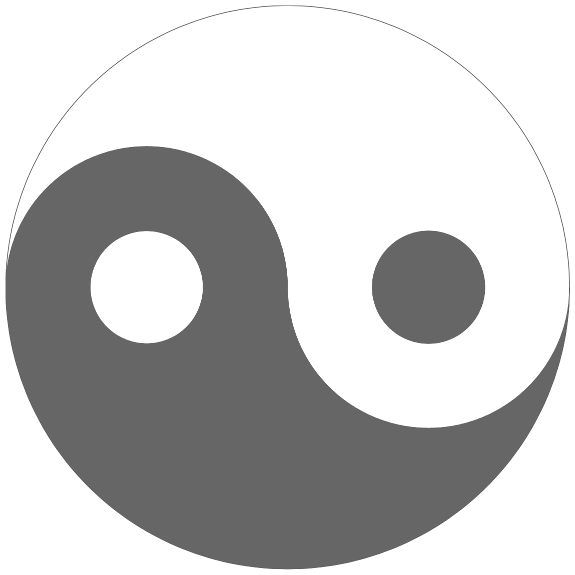


Figure 2.15: YinYang Dataset

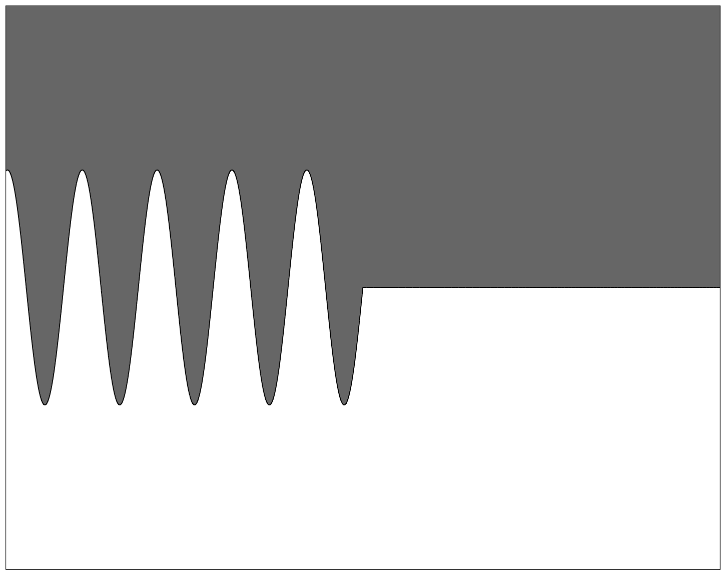
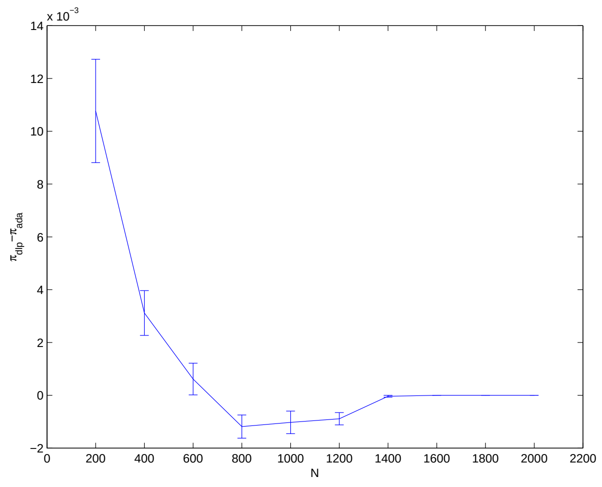
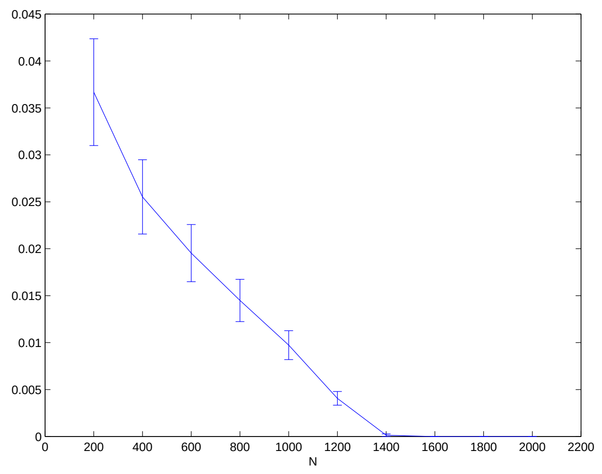


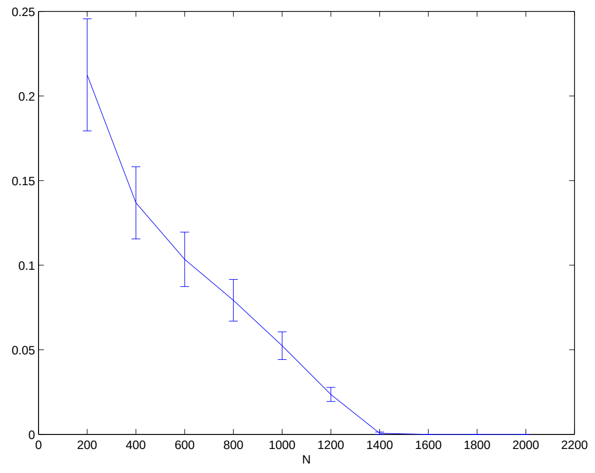
Figure 2.16: LeftSin Dataset



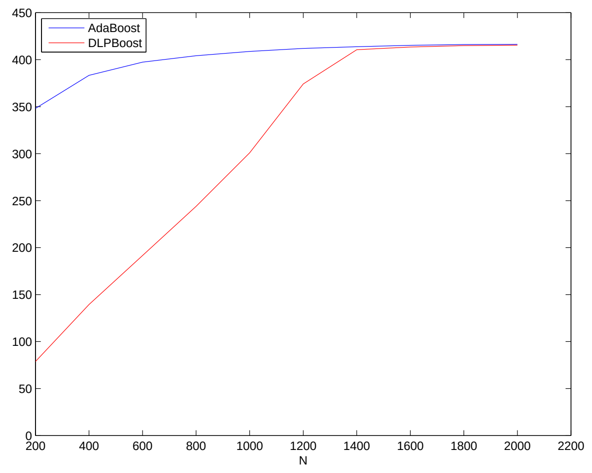
(a) Change in π



(b) Change in Average Margin



(c) Maximum Change in Margin

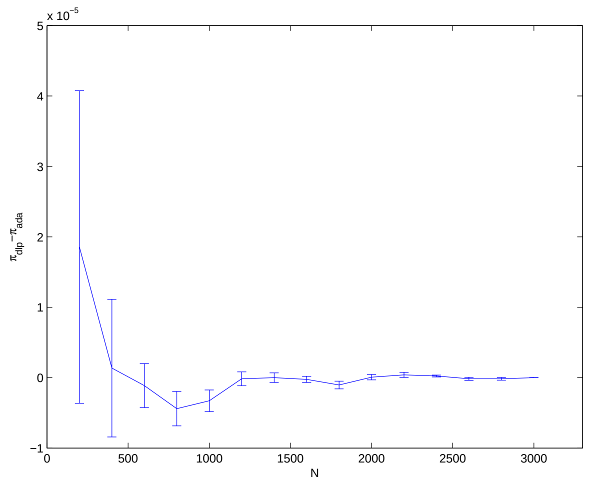


(d) Size of Ensemble

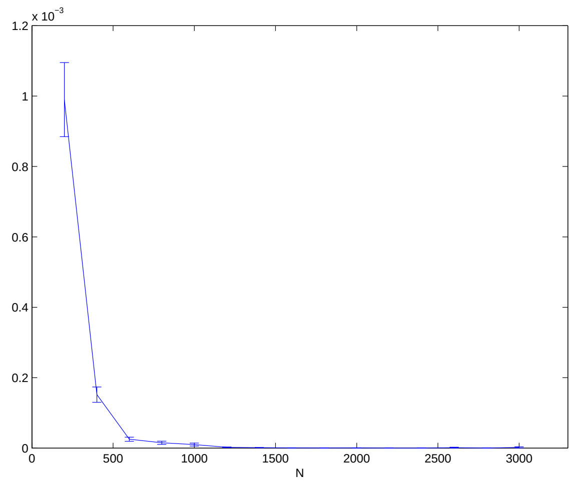
Figure 2.17: Comparison of DLPBoost and AdaBoost on TwoNorm Dataset

2.6.5.2 Dependence on Training Set Size

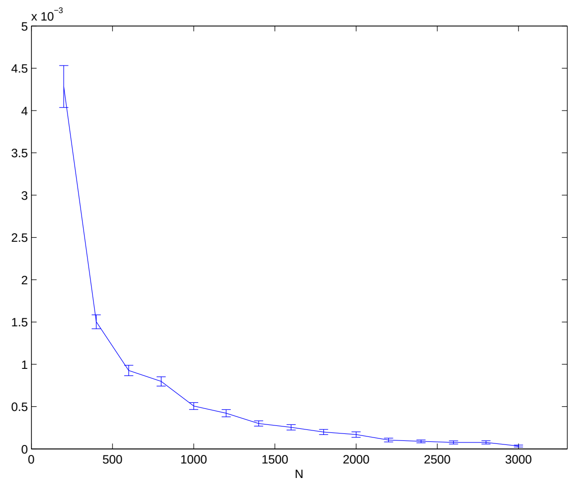
In the first test, we evaluate the difference in performance between AdaBoost and DLPBoost. We evaluate the difference in the out-of-sample error, the average change in the margin on the training points, and the maximum change in the margin. We also compare the number of hypothesis which have non-zero weight in the ensembles of AdaBoost and DLPBoost. Each of the simulations used an independent set of training and test points and was run for 500 iterations of AdaBoost with the training set size varying from 200 to 2000. All the results are averaged over 100 independent runs.



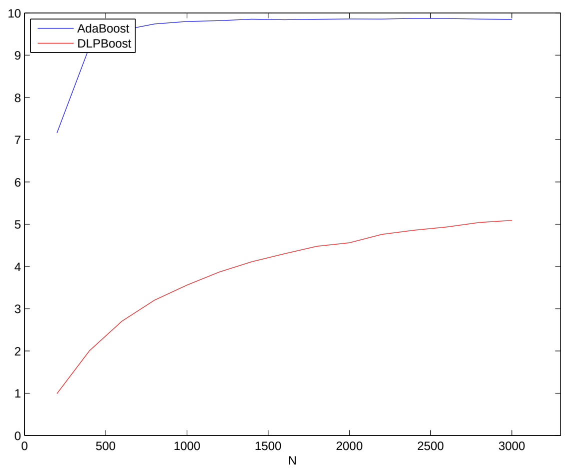
(a) Change in π



(b) Change in Average Margin



(c) Maximum Change in Margin

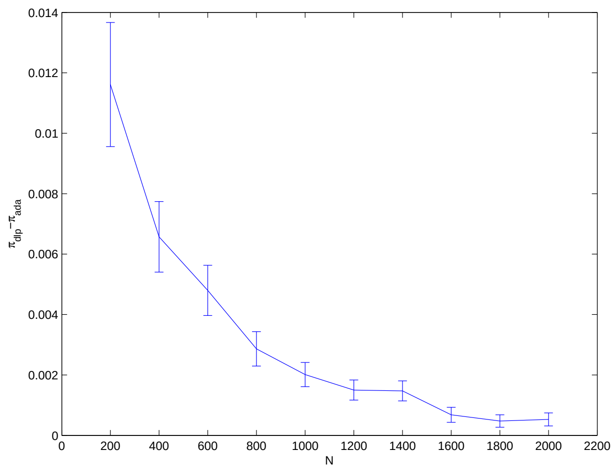


(d) Size of Ensemble

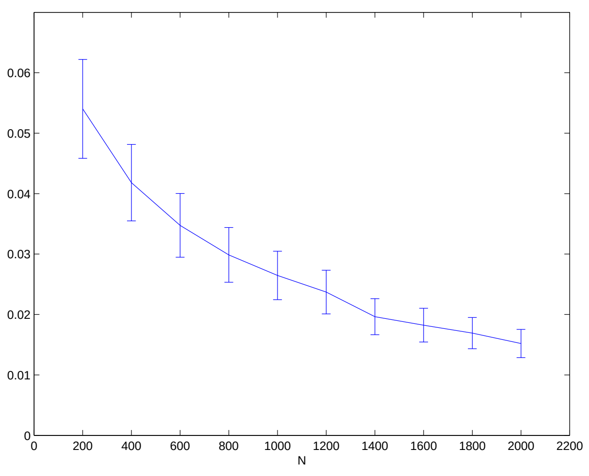
Figure 2.18: Comparison of DLPBoost and AdaBoost on YinYang Dataset

Figure 2.17 shows the results for the TwoNorm dataset, which is 20-dimensional. We observe that the out-of-sample performance of DLPBoost is consistently worse than that of AdaBoost. The difference goes down to zero as the training set size increases. The increase in the average margins and the maximum change in the margin are indications of the inefficiency of AdaBoost in optimizing the margins. We can also see the number of hypotheses used by the DLPBoost increase as we increase the training set size.

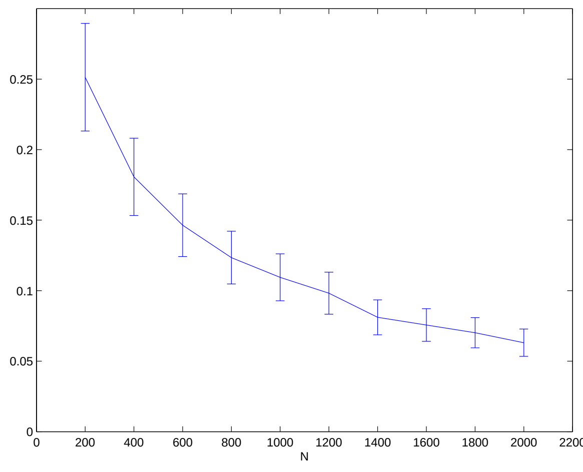
We observe similar behavior on the Yin-Yang dataset (Figure 2.18) which is 2-dimensional, except in this case the change in the out-of-sample error is very small and



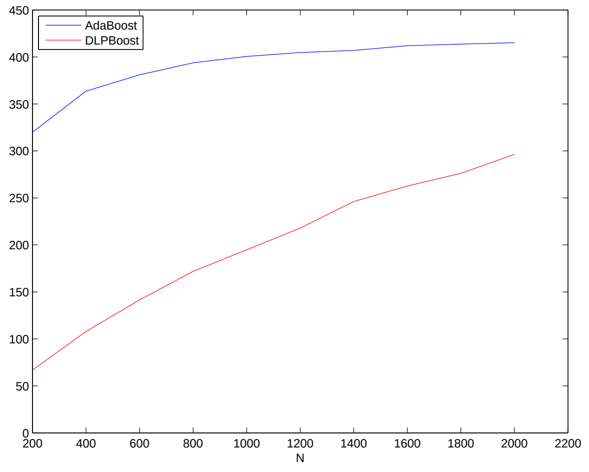
(a) Change in π



(b) Change in Average Margin



(c) Maximum Change in Margin

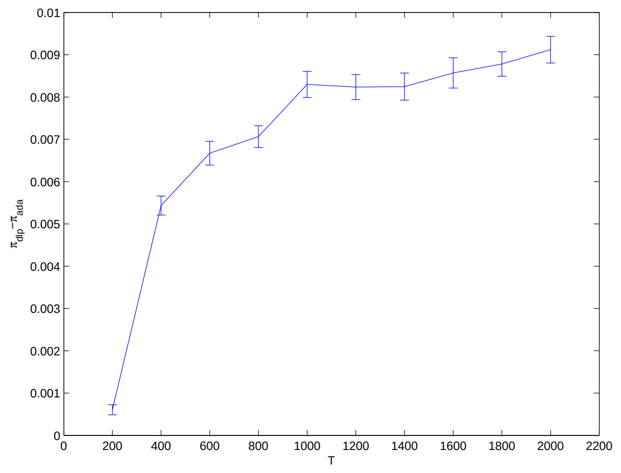


(d) Size of Ensemble

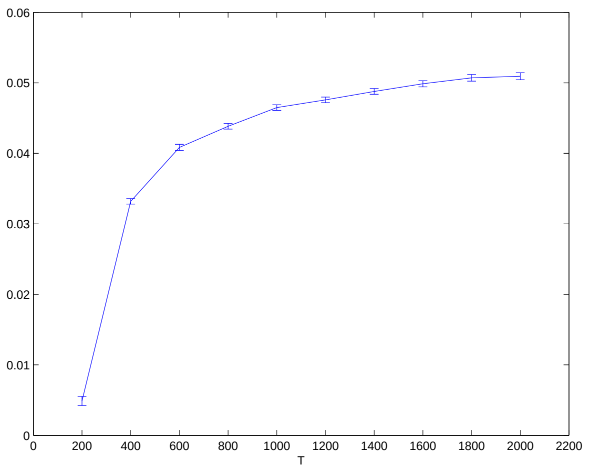
Figure 2.19: Comparison of DLPBoost and AdaBoost on RingNorm Dataset

not significantly different from 0 in most of the cases. Another observation here is that even for 3000 points in the training set, the DLPBoost solution does not use all the 500 hypotheses. In the TwoNorm case, with large training sets, the DLPBoost solution used all the 500 hypotheses in its solution. Hence the inefficiency in AdaBoost is a function of the complexity of the dataset and the number of examples in the training points. AdaBoost ended up using 500 hypotheses to explain the Yin-Yang dataset of 1000 examples or more which is a clear indication of over fitting. DLPBoost, on the other hand, adjusts its complexity based on the complexity of the dataset.

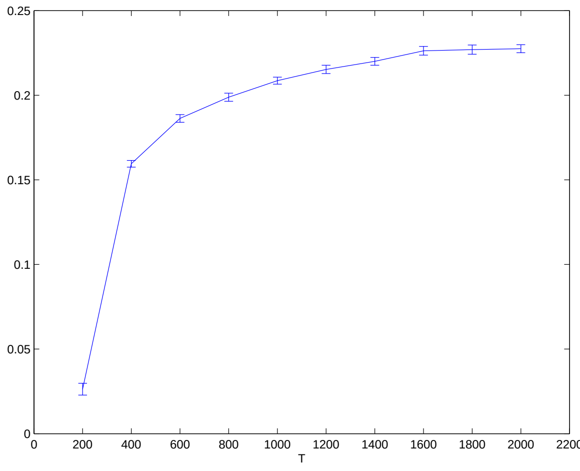
The RingNorm dataset is more complicated than the TwoNorm dataset and it is



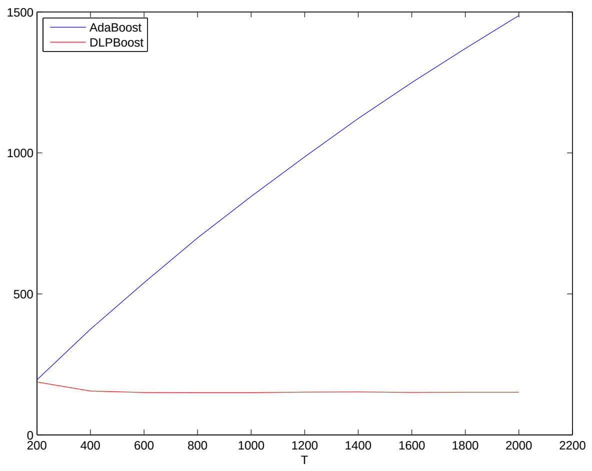
(a) Change in π



(b) Change in Average Margin



(c) Maximum Change in Margin



(d) Size of Ensemble

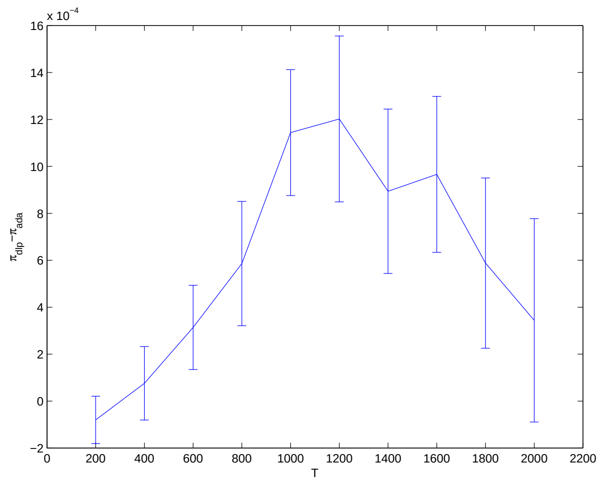
Figure 2.20: Comparison of DLPBoost and AdaBoost on TwoNorm Dataset

reflected in the results in Figure 2.19.

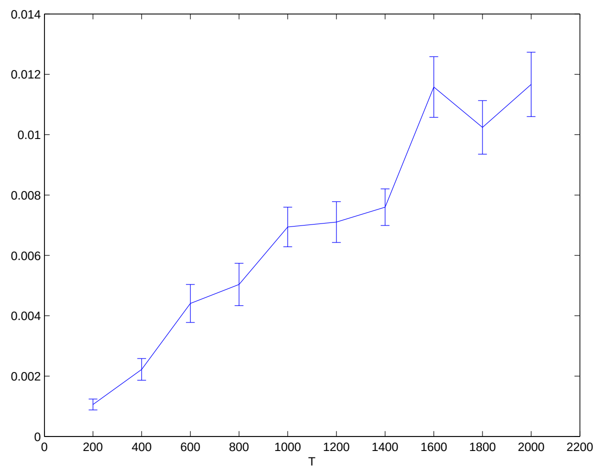
2.6.5.3 Dependence on the Ensemble Size

We ran the same experiments by fixing the ensemble size to 500 and varying the number of iterations given to AdaBoost. Figures (2.20–2.22) give the results of these experiments.

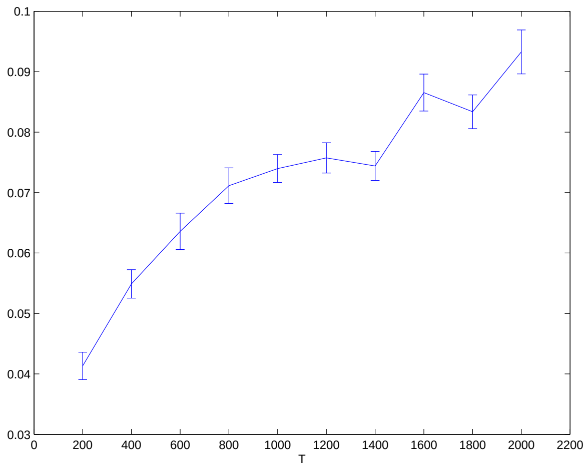
Here we observe that as we use more and more hypotheses in the AdaBoost ensemble, DLPBoost is able to get more improvement in the margins. This also leads to a consistent worsening of performance in the TwoNorm case. It is interesting to



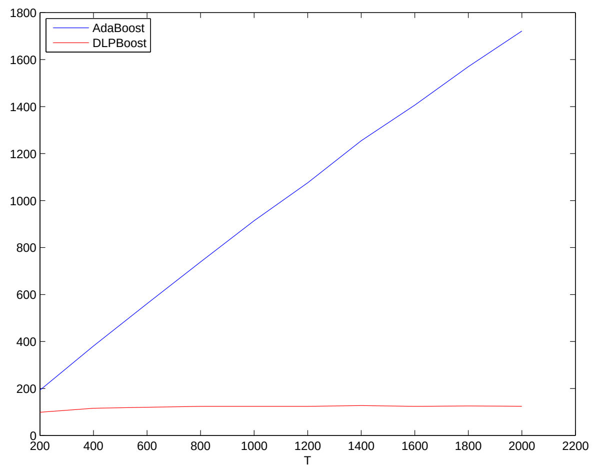
(a) Change in π



(b) Change in Average Margin

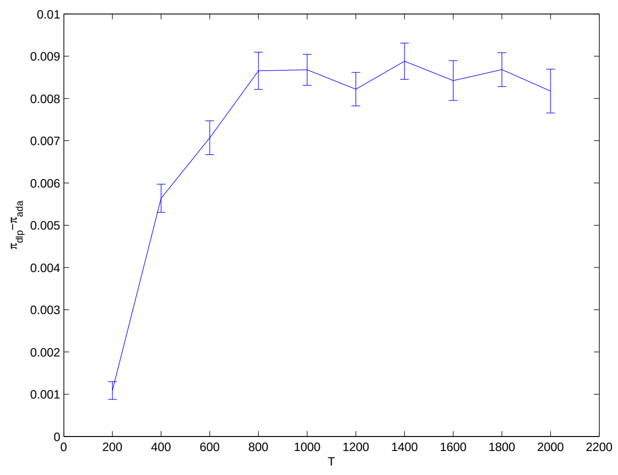


(c) Maximum Change in Margin

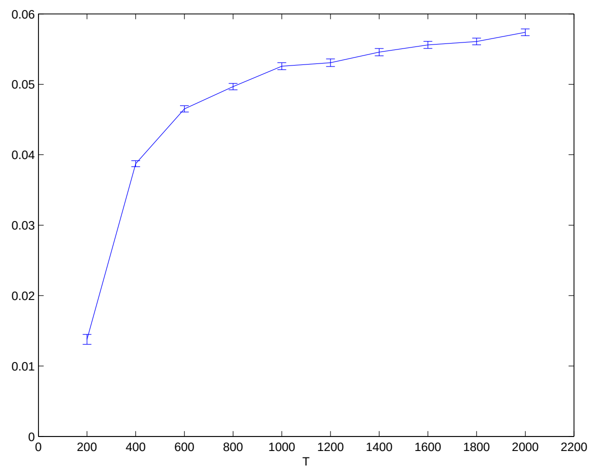


(d) Size of Ensemble

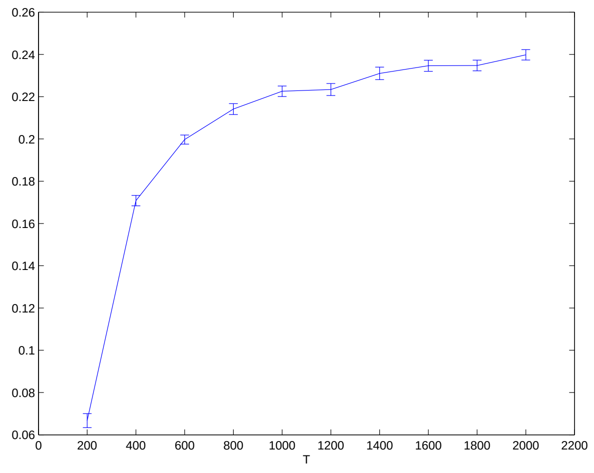
Figure 2.21: Comparison of DLPBoost and AdaBoost on YinYang Dataset



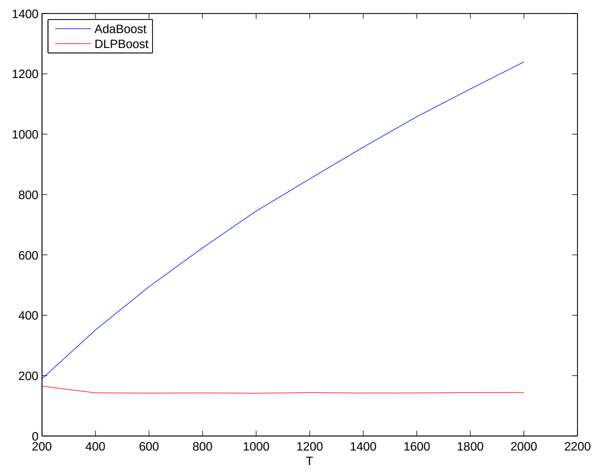
(a) Change in π



(b) Change in Average Margin



(c) Maximum Change in Margin



(d) Size of Ensemble

Figure 2.22: Comparison of DLPBoost and AdaBoost on RingNorm Dataset

Table 2.3: Experimental Results on UCI Data Sets

Data Set	Algorithm	Size of Ensemble	π	$\overline{\Delta}$	Maximum Change
Pima Indian	DLPBoost	99.12(0.06)	0.2466(0.003)	0.00829	0.0061
	AdaBoost	99.54(0.06)	0.2462(0.003)	0.0829	-
Sonar	DLPBoost	78.78(0.05)	0.1643(0.0004)	0.2221	0.1374
	AdaBoost	125.93(0.06)	0.1562(0.0005)	0.1978	-
Cleveland	DLPBoost	68.33(0.04)	0.1963(0.0005)	0.1210	0.0038
	AdaBoost	68.75(0.04)	0.1962(0.0005)	0.1201	-
Ionosphere	DLPBoost	95.47(0.07)	0.0941(0.0003)	0.1801	0.0259
	AdaBoost	107.76(0.05)	0.0904(0.0003)	0.1749	-
Vote	DLPBoost	15.31(0.04)	0.0621(0.002)	0.4457	0.0166
	AdaBoost	15.85(0.05)	0.0619(0.0002)	0.4422	-
Cancer	DLPBoost	75.37(0.08)	0.028(0.0001)	0.3797	0.1087
	AdaBoost	102.69(0.06)	0.026(0.0002)	0.3454	-

note that the DLPBoost solution uses almost the same number of hypothesis irrespective of the number of hypothesis in the AdaBoost ensemble. This also shows that the number of hypotheses required to explain the dataset is dependent only on the number of training points and the complexity of the dataset.

2.6.6 Experiments on Real World Datasets

We tested DLPBoost on the same six datasets from the UCI machine learning repository [Blake and Merz 1998] used in Section 2.5.3. Decision stumps were used as the base learner in all the experiments. For the experiments, the dataset was randomly divided into two sets of size 80% and 20%, used for training and testing the algorithms. All the results are averaged over 100 different splits to obtain the error bars. The AdaBoost algorithm was run for 250 iterations and the size of the ensemble is defined as the number of unique hypothesis in the ensemble. Table 2.3 summarizes the results. The maximum change column refers to the maximum increase in the margin of any point in the training set. The values in the parentheses denotes the size of the standard error bar.

The average and the maximum change are just two statistics of the margin distribution which are most informative to the discussion. The entire margin distribution

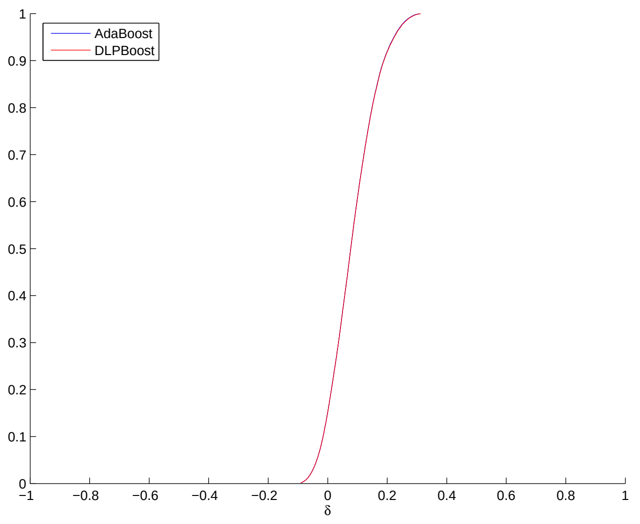


Figure 2.23: Margin Distribution on Pima Indian Dataset

obtained by the two algorithms is reported in Figures 2.23–2.28

We can observe that the improvement in margins is significant in some cases and very little in other cases. This improvement is inversely proportional to the increase in the errors. The two datasets Cancer and Sonar have a significant change in the distribution and a large maximum change. DLPBoost performs significantly worse on these two datasets.

2.7 Conclusions

AdaBoost is a gradient descent algorithm which produces an ensemble of classifiers and performs very well on many artificial and real-world problems. In this chapter, we've investigated the popular margin explanation for the success of AdaBoost. We've introduced two new extensions of AdaBoost, AlphaBoost and DLPBoost. AlphaBoost optimizes the cost function of AdaBoost more aggressively and shows that the cost function is prone to overfitting. DLPBoost tackles a more generalized problem and optimizes all the reasonable cost functions of margin. The very fact that there is room for improvement in the margin distribution shows the inefficiency of AdaBoost in terms of the margins. The performance of DLPBoost indicates that improving the margins will also lead to overfitting. It also shows that improving any particular

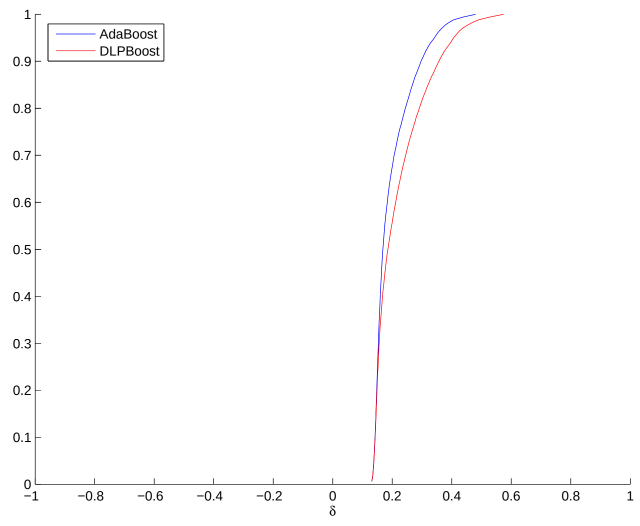


Figure 2.24: Margin Distribution on Sonar Dataset

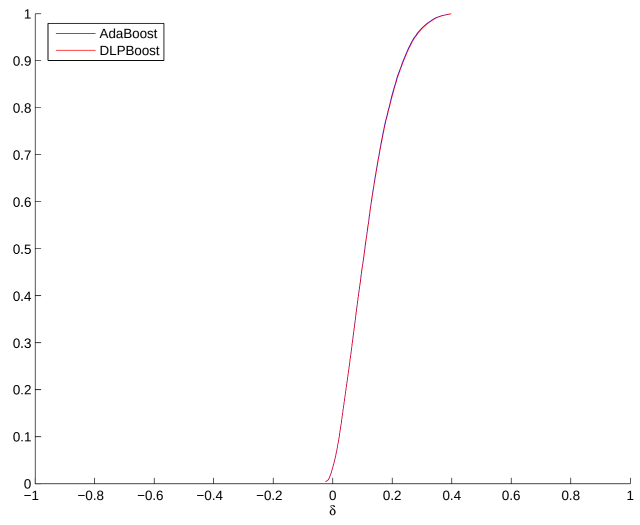


Figure 2.25: Margin Distribution on Cleveland Dataset

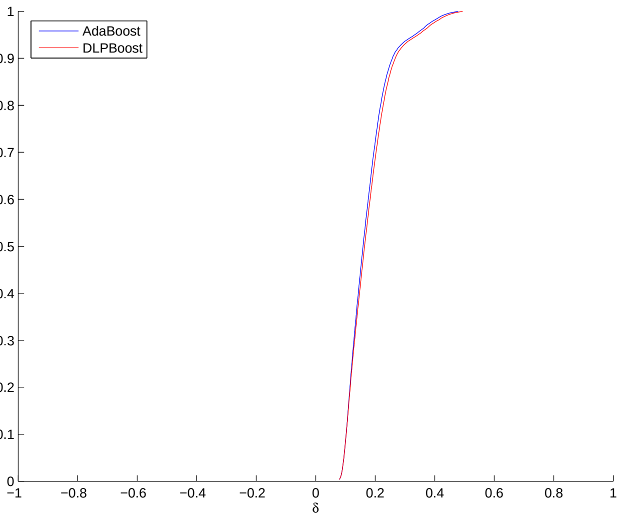


Figure 2.26: Margin Distribution on Ionosphere Dataset

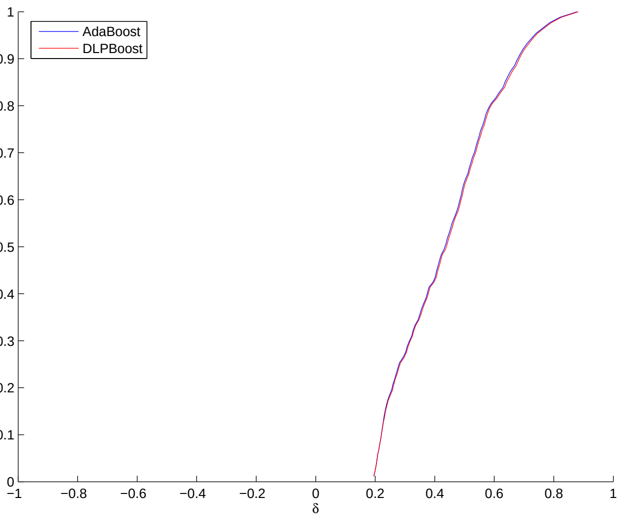


Figure 2.27: Margin Distribution on Voting Dataset

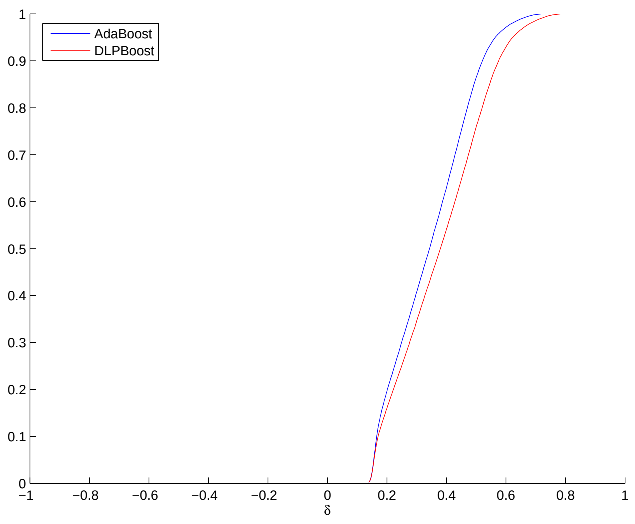


Figure 2.28: Margin Distribution on Cancer Dataset

statistic of the margin is futile and would not lead to improvement in the performance of the algorithm.

AdaBoost is improving the soft-max of the margins. This criterion gives very good performance on a variety of learning problems. The results of this chapter suggest that this criterion is only weakly correlated with the real explanation and the regularization of AdaBoost is really through poor optimization. Any further attempt to improve it has met with loss in the performance. This motivates the need for further research into the real criterion which would lead to design of better algorithms for machine learning.

Chapter 3

Adaptive Estimation Under Monotonicity Constraints

3.1 Introduction

The typical learning problem takes examples of the target function as input information and produces a hypothesis that approximates the target as an output. In this chapter, we consider a generalization of this paradigm by taking different types of information as input, and producing only specific properties of the target as output.

Generalizing the typical learning paradigm in this way has theoretical and practical merits. It is common in real-life situations that we would have access to heterogeneous pieces of information, not only input-output data. For instance, monotonicity and symmetry properties are often encountered in modeling patterns of capital markets and credit ratings [Abu-Mostafa 1995; 2001]. Input-output examples coming from historical data are but one of the pieces of information available in such applications. It is also commonplace that we are not interested in learning the entirety of the target function, and would be better off focusing on only some specific properties of the target function of particular interest and achieving better performance in this manner. For instance, instead of trying to estimate an entire utility curve, we may be only interested in the threshold values at which the curve goes above or below a critical value.

Another common feature encountered in the applications that we have considered is that the data is very expensive or time consuming to obtain. In many such cases, adaptive sampling is employed to get better performance with very small sample size [Cohn et al. 1994, MacKay 1992]. In this chapter, we will describe a new adaptive learning algorithm for estimating a *threshold*-like parameter from a monotonic function. The algorithm works by adaptively minimizing the uncertainty in the estimate via its entropy. This, combined with a provably consistent learning algorithm, is able to achieve significant improvement in performance over the existing methods.

3.1.1 Applications

This problem is very general in nature and appears in many different forms in Psychophysics [Palmer 1999, Klein 2001] , Standardized testing like the GRE [Baker 2001], adaptive pricing, and drug testing [Cox 1987]. In all these applications, there's a stimulus, and at each stimulus there is a probability of obtaining a positive response. This probability is known to be monotonically increasing.

The most popular application of these methods is in the field of drug testing. It is known that the probability of a treatment curing a disease is monotonic in the level of dosage in the treatment, within a certain range. It is desirable to obtain a correct dose level which will induce a certain probability of producing a cure. The samples in this problem are very expensive and adaptive testing is generally the preferred approach.

Standardized tests, like the SAT and the GRE, use adaptive testing to evaluate the intelligence level of students. Item Response Theory [Lord 1980] models each question that is presented in the test as an item and for each item, the probability of a student correctly answering the question is monotonic in his intelligence. Each Item has a different difficulty level, and the next item presented to the student is adaptively chosen based on his or her performance on the previous items. A student's GRE or SAT score is defined as a difficulty level at which a student has a certain probability of correctly answering the questions.

Another field where this kind of problem occurs is in determining the optimal discount level when pricing a product for sale. The probability of a customer buying a certain product increases monotonically with the amount of discount offered. In this application, it is not necessary to go after the entire price elasticity curve to determine the discount value at which the expected profit is sufficient to achieve a sales goal.

All these applications have the common theme that the target is a monotonic function from which we can adaptively get Bernoulli samples which are otherwise expensive to obtain. In each of the cases, we are not interested in the whole target but only in a critical value of the input.

3.1.2 Monotonic Estimation Setup

Consider a family of Bernoulli random variables $V(x)$ with parameters $p(x)$ which are known to be monotonic in x, i.e., $p(x) \leq p(y)$ for $x < y$. Suppose for $X = \{x_1, .., x_M\}$, we have $m(x_i)$ independent samples of $V(x_i)$. We can estimate $p(x_i)$ as the average number of positive responses y_i. If the number of samples is small then these averages will not necessarily satisfy the monotonicity condition. We will discuss the existing methods which take advantage of the monotonicity restriction on the estimation process and introduce a new regularized algorithm.

3.1.3 Threshold Function

We have a target function $p : X \rightarrow Y$ which is monotonically increasing. A *threshold* functional θ maps a function p to X. The threshold function can take many forms, the most common among them are:

Critical Value Crossing Functional. This is the most common form of the threshold. The problem here is to estimate the input value at which the monotonic function goes above a certain value. For example $\theta(h) = h^{-1}(0.612)$ measures the point at which the function crosses 0.612.

Trade-Off Functional. This takes the form $\theta(h) = argmax_x G(x)p(x)$, where $G(x)$

is a fixed known function and is usually monotonically non increasing. This kind of functional appears in the adaptive pricing problem, where the goal is to maximize expected profit. If the probability of a sale upon offering a discount rate of x is $p(x)$, then the expected profit is given by $\theta(x) = (1-x)p(x)$.

3.1.4 Adaptive Learning Setup

In the adaptive learning setup, given an input point $x \in X$, we can obtain a noisy output $y = p(x) + \epsilon(x)$, where ϵ is a zero mean input dependent noise function. For the scope of this chapter, $Y = \{0, 1\}$ and the noisy sample at x would be a Bernoulli random variable with mean $p(x)$, i.e., $y|x = B(p(x))$. This scenario is common in various real-world situations in which we can only measure the outcome of the experiment (success or failure) and wish to estimate the underlying probability of success. For adaptive sampling, we are allowed to choose the location of the next sample based on the previous data samples and any prior knowledge that we might have. In the following sections, we will introduce existing and new methods to sample adaptively for monotonic estimation.

3.2 Existing Algorithms for Estimation under Monotonicity Constraint

Most of the currently used methods employ a parametric estimation [Finney 1971, Wichmann and Hill 2001] technique for fitting a monotonic function to the data. The main advantage of this method is the ease in estimation and the high regularization ability for small sample sizes, as there are only a small number of parameters to fit.

An alternate approach is to use the non-parametric method, MonFit (also known in literature as isotonic regression [Barlow et al. 1972]) in which we do not assume anything about the target function. This method has better performance as it does not have the additional bias from the parametric assumption and is able to regularize the estimate by enforcing the monotonicity constraint. This method is not used much in practice, as the output from it is a step-wise linear function with a lot of flat regions, which is not very intuitive, although optimal in the absence of any assumptions about the setup.

3.2.1 Parametric Estimation

Parametric estimation is the most popular method used in practice. The method assumes a parametric form for the target and estimates the parameters using the data. This gives the entire target as the output. In most of the parametric methods, the target function is assumed to be coming from a 2-parameter family of functions. The parameters are estimated from the data and the threshold or the threshold-like quantity can be computed from the inverse of the parametric function. The most commonly used methods for estimating the parameters are probit analysis [Finney 1971] and maximum likelihood [Watson 1971, Wichmann and Hill 2001]. In this thesis, we will use the maximum likelihood method with the two parameter logistic family [Hastie et al. 2001] as the model.

$$\mathcal{F}_{log} = \{f(x, a, b) = 1/(1 + e^{(x-a)/b})\}$$

3.2.2 Non-Parametric Estimation

In the non-parametric approach (MonFit), [Barlow et al. 1972, Brunk 1955] the solution is obtained by a maximum likelihood estimation under the constraints that the underlying variables are monotonic. It is equivalent to finding the closest monotonic series in the mean square sense.

MonFit is an algorithm which solves the optimization problem

$$\min_{\mu} \sum_{i=1}^{M} m(x_i)\left[\mu_i - y_i\right]^2 \tag{3.1}$$

under the constraints

$$\mu_i \leq \mu_j \ \forall i \leq j$$

The MonFit estimator always exists and is unique. It has a closed-form solution [Barlow et al. 1972]

$$\hat{\mu}_i^* = \max_{s \leq i} \min_{t \geq i} Av(s,t) \tag{3.2}$$

where

$$Av(s,t) = \frac{\sum_i m(x_i) y_i}{\sum_i m(x_i)}$$

The solution can be computed in linear time by the Pool Adjusted Vector Algorithm [Barlow et al. 1972]. The algorithm starts out with a pool of points, one for each of the design points x_i. The value of the estimator for each pool is computed as the average value of the points inside the pool. If there are two consecutive pools which violate the monotonicity restriction, then these pools are merged into one. This procedure is repeated until all consecutive pools obey the monotonicity restriction.

Theorem 1 *Given examples from a family of Bernoulli random variables with monotonic probability, the MonFit estimate has a lower or equal mean square error than the naive estimate*

Proof Let us consider the case when we have one example in each bin. We will denote the target as $p_i = p(x_i)$ and assume that each bin has one sample, i.e., $m(x_i) = 1$. Suppose the first pool of the MonFit estimate contains k points, then the partial mean square error made in the first pool for the MonFit case would be

$$e_{mbt}^1 = \sum_{i=1}^{k} (\bar{y} - p_i)^2$$

where $\bar{y} = \sum_{i=1}^{k} y_i$ is the average pooled value which is assigned to each bin in the first pool.

The partial mean square error by the naive estimate would be

$$e^1_{naive} = \sum_{i=1}^{k}(y_i - p_i)^2$$

The difference between the errors can be written as

$$e^1_{naive} - e^1_{mbt} = \frac{1}{k}\sum_{i,j=1}^{k}(y_i - y_j)^2 + \frac{2}{k}\sum_{i=1}^{k}\sum_{j=i+1}^{k}(p_i - p_j)(y_j - y_i) \tag{3.3}$$

Now, since the first k points were pooled, we have that

$$y_1 \geq \frac{y_1 + y_2}{2} \geq \frac{y_1 + y_2 + y_3}{3} \geq \ldots \geq \frac{\sum_{i=1}^{k} y_i}{k}$$

From which we can conclude, that

$$y_1 \geq y_2 \geq y_3 \geq \ldots \geq y_k$$

$p_i's$ are monotonic, therefore,

$$p_1 \leq p_2 \leq p_3 \leq \ldots \leq p_k$$

So, each term in equation (3.3) is non-negative. So, we have that $e^1_{naive} \geq e^1_{mbt}$.

The mean square error would be the sum of the partial mean square errors in each pool, so $e_{naive} \geq e_{mbt}$

For the general case, when each bin has a weight $w_i = m(x_i)$, we can do similar calculation and get

$$e^1_{naive} - e^1_{mbt} = \frac{1}{k}\sum_{i,j=1}^{k} w_i w_j (y_i - y_j)^2 + \frac{2}{k}\sum_{i=1}^{k}\sum_{j=i+1}^{k} w_i w_j (p_i - p_j)(y_j - y_i)$$

where, in this case,

$$e^1_{mbt} = \sum_{i=1}^{k} w_i (\bar{y} - p_i)^2$$

where

$$\bar{y} = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

and

$$e^1_{naive} = \sum_{i=1}^{k} w_i (y_i - p_i)^2$$

and the ordering on $y_i' s$ still holds, so each term is again positive.

□

Theorem 1 proves the advantage in enforcing the monotonicity constraint. The naive estimate is unbiased, and by enforcing the additional constraints the MonFit estimator becomes biased. However, the variance of the MonFit estimator is reduced. This result proves that overall, the increase in the bias is less than the corresponding decrease in the variance, making MonFit a better option for minimizing the mean square error.

The MonFit solution only gives the target values at the design points. To estimate the threshold, the function can be interpolated in any possible way. The most common approach is to interpolate it linearly in between the design points, although more sophisticated approaches are possible (like monotonic splines [Ramsay 1998]). We use linear interpolation because of its simplicity and the fact that we do not have any a priori knowledge to choose one over another. Theorem 2 proves the consistency property of the MonFit estimate with constant interpolation but it can also be extended to linear interpolation.

Theorem 2 *Let $f(x)$ be a continuous non-decreasing function on (a, b). Let $\{x_n\}$be a sequence of points dense in $(a, b)$with one observation made at each point. Let the variance of the observed random variables be bounded. Let $\hat{f}_n(x)$be the estimate based on the first n observations, defined to be constant between observation points and continuous from the left. If $c > a$ and $d < b$, then*

$$P\left[\lim_{n\to\infty} \max_{c \le t \le d} \left| f(t) - \hat{f}(t) \right| = 0\right] = 1$$

Proof See Brunk [1958]

3.3 Hybrid Estimation Procedure

The existing methods which we have discussed have their own advantages and disadvantages. We introduce a hybrid estimation technique, which is a combination of the parametric and the nonparametric approaches that tries to combine the advantages of both into one approach. We developed a regularized version of MonFit, which we call *Bounded Jump Regression,* that uses jump parameters to smoothen and regularize the MonFit solution. Bounded jump regression with a heuristic for parameter selection is used here for estimation in the hybrid estimation procedure.

3.3.1 Bounded Jump Regression

One of the main drawbacks of the non-parametric approach is that the solution it produces is a step function, which for many applications is non-smooth and tends to have a lot of flat regions. Practically, this kind of behavior is unacceptable [Wichmann and Hill 2001]. In this section, we introduce a new approach to estimating the solution which introduces a parameter to the MonFit algorithm in order to make it smooth.

The non-parametric estimation technique solves a least squares regression problem under the monotonicity constraint. The monotonicity constraint ensures that the difference between two consecutive estimates is positive. In this section we introduce a generalization of this problem where the difference between two consecutive estimates is bounded above and below by certain parameters.

3.3.1.1 Definition

We define the *Bounded Jump Regression* problem as follows

$$\min_{\mu} \sum_{i=1}^{N} (\mu_i - y_i)^2 \tag{3.4}$$

Subject to the constraints that

$$\delta_i \leq \mu_{i+1} - \mu_i \leq \Delta_i \, \forall i = 0..N \tag{3.5}$$

where μ_0 and μ_{N+1} are fixed constants and $\{\delta_i, \Delta_i\}_{i=1}^{N}$are parameters for the model.

3.3.1.2 Examples

Many of the regression problems can be expressed as a special case of the bounded jump regression problem. For regression of binomial probabilities which we are dealing with in this chapter, we fix the boundary points $\mu_0 = 0$ and $\mu_{N+1} = 1$.

- When $\delta_i = -\infty$ and $\Delta_i = \infty$, the constraints are vacuous and the problem reduces to normal mean squares regression.
- When $\delta_i = 0$ and $\Delta_i = \infty$, the constraints enforce the monotonicity on $\mu_i' s$, and so, the problem reduces to monotonic regression.
- When $\delta_i = 1/N + 1$, there is only one feasible solution and so there is no regression problem to solve.

3.3.1.3 Solving the BJR Problem

The BJR problem can be reduced to a quadratic optimization problem under box constraints by using suitable variable transformations. If we substitute

$$z_i = \frac{p_{i+1} - p_i - \delta_i}{1 - \sum_{t=0}^{N} \delta_t}$$

and

$$C_i = \frac{\Delta_i - \delta_i}{1 - \sum \delta_t}$$

and

$$Q_{ij} = min(N + 1 - i, N + 1 - j)$$

and

$$y_i^{new} = \sum_{t=1}^{i} \left(\frac{\bar{y}_t}{1 - \sum_{r=0}^{N} \delta_r} - \delta_t \right)$$

then, BJR can be reduced to

$$\min \frac{1}{2} z^T Q z - z^T y^{new}$$

under the constraints that

$$0 \leq z \leq C$$

and

$$z^T \mathbf{1} = 1$$

3.3.1.4 Special Cases

For the case when we have $\Delta_i = \infty$, the BJR can be reduced to monotonic regression problem and hence can be solved in linear time.

By using the transform

$$z_i = \frac{p_i - \sum_{t=1}^{i} \delta_t}{1 - \sum_{t=0}^{N} \delta_t}$$

and

$$y_i^{new} = \frac{y_i - \sum_{t=1}^{i} \delta_t}{1 - \sum_{r=0}^{N} \delta_r}$$

The BJR reduces to

$$\min_{\mu} \sum_{i=1}^{N} (z_i - y_i^{new})^2 \tag{3.6}$$

Subject to the constraints that

$$z_{i+1} - z_i \geq 0 \, \forall i = 0..N \tag{3.7}$$

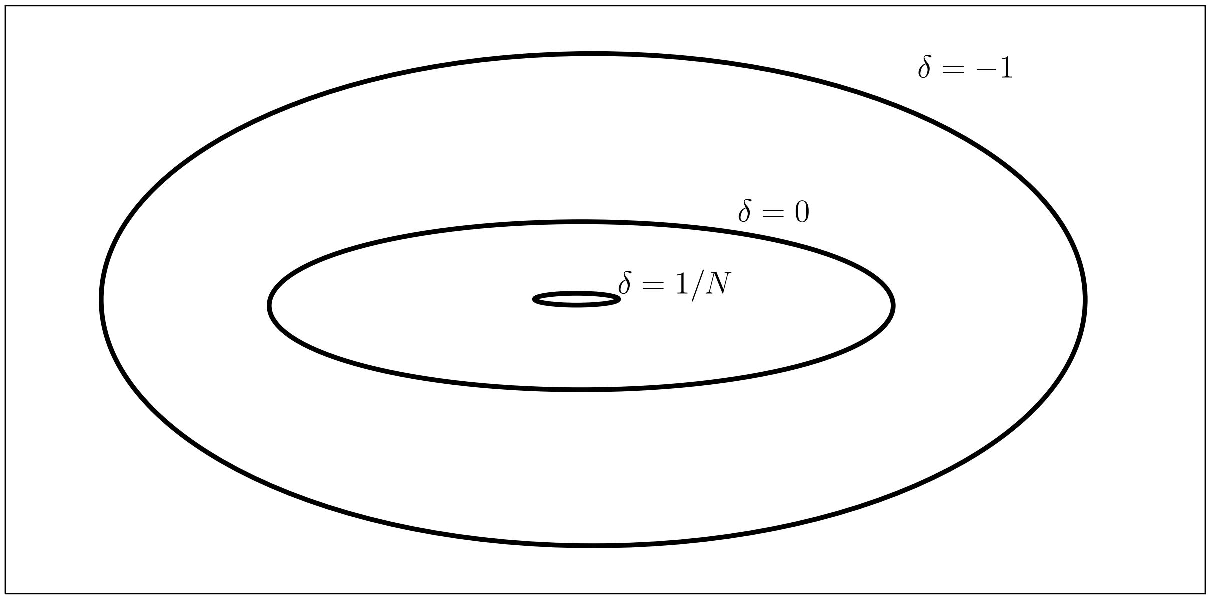


Figure 3.1: δ as a Complexity Parameter Controlling the Search Space

3.3.1.5 δ as a Complexity Parameter

Consider the case, when we have $\Delta_i = \infty$, i.e., we're only bounding the minimum jump between two consecutive bins, and $\delta_i = \delta$ for all i. For the case when we have $\delta = -\infty$, BJR reduces to normal regression and the set of feasible solution is the entire $\mathbb{R}^N$ plane (or if we are working with probabilities, it will be $[0,1]^N$). As we increase δ, the feasible region keeps shrinking. At $\delta = 0$, the feasible region is the set of all monotonic functions, and when $\delta_i = 1/N + 1$, the feasible region contains just one solution. So, δ can be seen as a measure of complexity of the solution space and increasing delta will reduce the variance while increasing the bias. The number of independent parameters in the feasible region can also be considered as a function of δ. For $\delta = -\infty$, the number of independent parameters is N, and for $\delta = 1/N + 1$, it is zero. This gives us a continuous control over the bias and variance trade-off. $\delta = -\infty$ corresponds to the zero bias solution and $\delta = 1/N + 1$ corresponds to the zero variance solution.

3.3.2 Hybrid Estimation Algorithm

In the hybrid estimation algorithm we use bounded jump regression with the lower bound δ as a parameter. By controlling δ, we are effectively controlling the size of the feasible region and thus giving better generalization performance, especially when we have a small number of examples. The jump parameter denotes the minimum value of the gradient of the target function. As we have seen in 3.3.1.5 , it controls the search space in which the non-parametric algorithm searches for the optimal solution. However, we have the same number of parameters as the number of bins If we estimate all of them from the data, it would really be a non-parametric procedure as the number of parameters vary according to the data. To limit this, we use only a single parameter and scale it differently for each bin. Estimating δ requires knowledge about the nature of the target function and we estimate δ using the parametric fit.

The δ parameter is the lower bound on the jump between bins, i.e. $p_{i+1} - p_i \geq \delta_i$. This can be estimated by the gradient of the target function. We have

$$p_{i+1} - p_i \geq (x_{i+1} - x_i) \min(p'(x))$$

We estimate the gradient of the target function as the minimum of the gradient of the parametric solution. We set

$$\delta_i \propto (x_{i+1} - x_i) \min_j \frac{\hat{p}(x_{j+1}) - \hat{p}(x_j)}{x_{j+1} - x_j}$$

The MonFit estimation algorithm does not take into account the change in input value between the bins. The parametric method exploits the dependence nicely and returns a function which depends on the location of the bins. This heuristic allows us to introduce this dependence into the estimation procedure.

Ideally, if we have small number of samples then we should choose a smaller space to search in; as we get more and more examples, we can afford to expand the search space. We therefore decrease the jump parameter as we get more samples. In the limit as we increase the number of samples, the hybrid estimation converges to the

Algorithm 5 HybridFit(x, y, m)

```
• Input: $x = (x_1, .., x_N)$, $y = (y_1, ..., y_N)$, and $m = (m_1, ..., m_N)$, where $y_i$ is the
  average number of positive responses out of $m_i$ samples which were taken at $x_i$
    – Let $f_l(x)$ be the best logistic fit for the data $(x, y, m)$
    – Calculate $d = \min \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}$, the minimum gradient estimate
    – Let $\delta_i = d \frac{x_{i+1} - x_i}{m_i}$
    – Define $y_i^{new} = \frac{y_i - \sum_{j=1}^{i} \delta_j}{1 - \sum_{j=1}^{N} \delta_j}$
    – Let $z = MonFit(y, m)$
    – Return $\mu_i^{hybrid} = z_i(1 - \sum_{j=1}^{N} \delta_j) + \sum_{j=1}^{i} \delta_j$
```

same solution as the monotonic regression. This allows us to derive the asymptotic properties for the hybrid estimation, like consistency and convergence. We scale the gradient at any point by the inverse of the number of samples at that point.

3.3.3 Pseudo-Code and Properties

Algorithm 5 describes the hybrid estimation algorithm. The hybrid estimation algorithm inherits the asymptotic statistical properties from the MonFit estimator. As the number of samples increase densely in the region of interest, the δ parameter goes down to zero.

Corollary 1 *Let $f(x)$ be a continuous non-decreasing function on (a, b). Let $\{x_n\}$be a sequence of points dense in $(a, b)$with one observation made at each point. Let the variance of the observed random variables be bounded. Let $\hat{f}_n(x)$be the hybrid estimate based on the first n observations, defined to be constant between observation points and continuous from the left. If $c > a$ and $d < b$, then*

$$P\left[\lim_{n\to\infty} \max_{c \le t \le d} \left| f(t) - \hat{f}(t) \right| = 0\right] = 1$$

PROOF Since, f is a continuous non-decreasing function on (a, b), the minimum of the gradient of the parametric estimate used in the estimation of δ would be bounded. The scaling factor is proportional to the difference between the consecutive sample points. Since the points are dense in the region, the minimum jump parameter would converge to 0 as $n \to \infty$. With these observations, the Corollary follows directly from Theorem 2. $\square$

3.3.4 Experimental Evaluation

We compared the hybrid estimation algorithm with the MonFit and the parametric fit algorithms. We compare the performance for various shapes of the target function as shown in Table 3.1.

Table 3.1: Various Shapes of Target Function Used in the Comparison

Linear	$f(x) = x$
Weibull	$f(x) = (1 - e^{-x^2})$
NormCDF	$f(x) = \Phi(\frac{x-0.5}{0.25})$
Student-t	Student's t distribution with 10 dof
Quadratic	$f(x) = x^2$
TR	$f(x) = \begin{cases} 2x^2 & x \leq 0.5 \\ 1 - 2(1-x)^2 & x > 0.5 \end{cases}$
Exponential	$f(x) = \frac{1}{1+e^{\frac{a-x}{b}}}$,$a = 0.5, b = 0.05$
Exponential2	$f(x) = \frac{1}{1+e^{\frac{a-x}{b}}}$,$a = 0.4, b = 0.1$

For each of the targets, we compute the deviation in the estimates which is defined as

$$d(y, m) = \sum_{i=1}^{N} m(x_i)(y_i - p_i)^2 / \sum_{i=1}^{N} m(x_i)$$

Figure 3.2 shows the deviation when we have one sample per bin (i.e. $m(x_i) = 1$) and Figure 3.3 shows the result when more samples are taken in the center than in

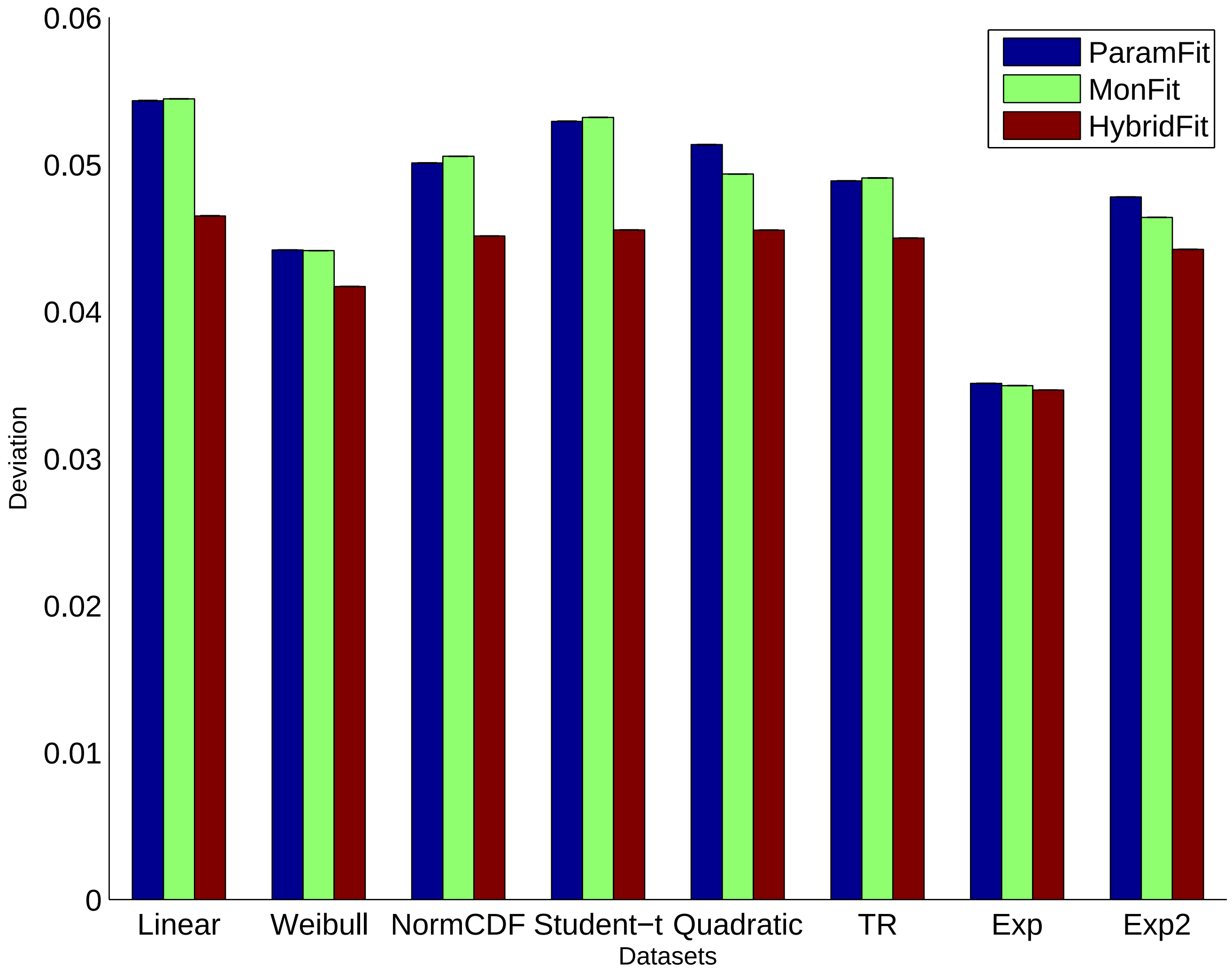


Figure 3.2: Comparison of Various Monotonic Estimation Methods with 1 Sample Per Bin

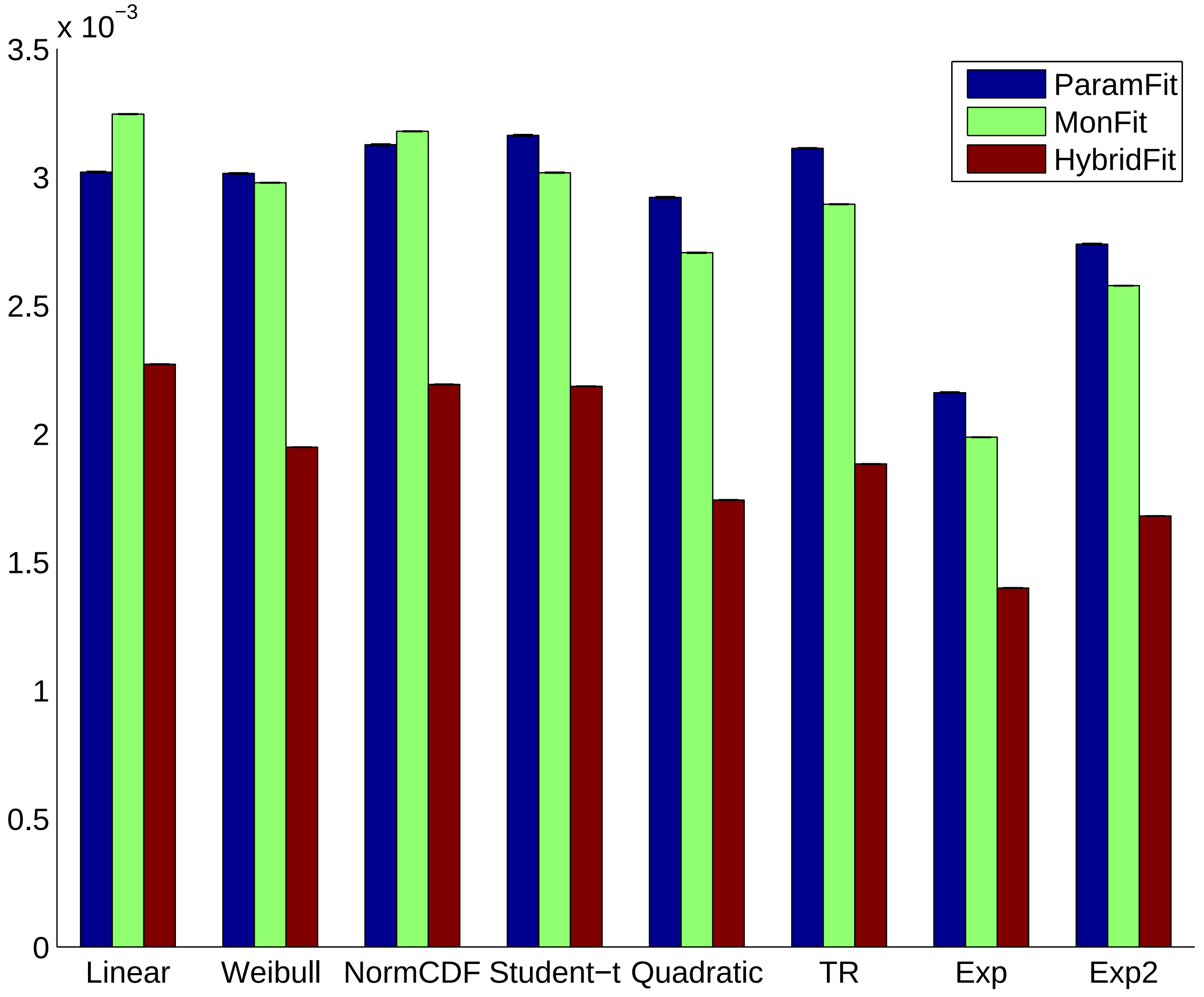


Figure 3.3: Comparison of Various Monotonic Estimation Methods with $m(x_i) = \min(i, N - i)$

the corners (i.e. $m(x_i) = \min(i, N - i)$).

We can see that hybrid fit generally outperforms both parametric and MonFit on all the targets, as it uses information from both the parametric and MonFit to compute its estimate. The performance ParamFit depends on how close the assumptions made in the parametric model hold for the real target function. For some cases it outperforms the MonFit method, but in those cases, there is not much difference in the performance of the two methods.

3.4 Adaptive Algorithms For Monotonic Estimation

In most of the problems we have described, the samples are very few and expensive to obtain. In these cases, whenever possible, it is beneficial to adaptively select the location of an example based on the known information. This way we can avoid samples which provide little or no information and speed up the estimation process. The problems described in the introduction fit into this framework. Computerized standardized testing uses adaptive estimation to zero in on a person's score, similarly most of the drug testing research is done adaptively. There are many existing algorithms for estimating the threshold of a monotonic function. Two of the most common approaches follow.

3.4.1 Maximum Likelihood Approach

One of the most popular approaches for adaptive sampling is the maximum likelihood approach which is very similar to the Best PEST [Taylor and Creelman 1967, Pentland 1980](Parametric Estimation with Sequential Testing). As the name suggests, it assumes a simple parametric form for the underlying function and estimates the parameters using the maximum likelihood approach. It adaptively places the new sample at the current value of the threshold [Leek 2001]. This approach has many advantages, like very simple estimation procedure and a simple rule for adaptive sampling. The major drawback of this method is that it assumes a parametric form for the underlying target. This assumption introduces a bias in the estimation which can be very large if the actual target is far from the model.

3.4.2 Staircase Procedure

Staircase procedures [Cornsweet 1962] are the simplest and most commonly used procedure for estimating the threshold of a monotonic function with Bernoulli sampling. They start out sampling at an arbitrary value and depending on the outcome of the current sample, they either increase the input value or decrease it. Using this procedure, one can estimate the 50% threshold value. This procedure can be modified to target a few other threshold values. For example, if we do a two up, one down procedure, in which we increase the input after two consecutive negative samples and decrease it after every positive sample, this procedure would target the 70.7% threshold value. The variations of this procedure would use different number of consecutive samples before increasing or decreasing the input. A major drawback of this algorithm is that it can only target a few threshold values and so cannot be used for a general threshold functional.

3.5 MinEnt Adaptive Framework

The MinEnt adaptive framework tries to place the new samples with an aim of minimizing the entropy of the threshold distribution. It uses a technique similar to the one described in Fedorov [1972], and at each stage assumes the current estimate of the underlying function as the true function and decides where to sample next based on that.

3.5.1 Dual Problem

Adaptive sampling can be viewed as a dual problem of learning, if we assume the current estimate to be the true target function. In learning, given the datasets, we would like to estimate the true function. In sampling, we are trying to get the best sample which would have the maximal information about the function in general or the threshold in particular. Consider a target function $f : X \rightarrow Y$, and a learning model $\mathcal{H} = \{h : X \rightarrow Y\}$, and a known functional θ which maps a function h to an element in X. Our goal is to estimate $\theta(f)$ for an the function f. Given the locations $\{x_1, x_2, ..., x_n\}$, we can get noisy samples from $\{y_1, y_2, ..., y_n\}$ where $y_i = f(x_i) + \epsilon(x)$. Using the combined input-output pairs, we can estimate the underlying function f or its threshold $\theta(f)$, which now will be a random variable dependent on the noise ϵ. In MinEnt setup, we try to minimize the dependence of the estimate on the underlying noise. This is done by adaptively minimizing the entropy of the estimated threshold.

3.5.2 Algorithm

There are four main steps in the MinEnt adaptive sampling algorithm :

Estimation Given the current samples, the framework first uses the learning algorithm to obtain an estimate of the target function.

Re-sampling We assume that the current estimate to be the actual target. Now, the problem is reduced to the question in Section 3.5.1: given the current sample locations, where in X should we sample next to minimize the entropy or uncertainty in the threshold estimate? This is done by repeated sampling from the estimated target and increasing the sample size by one at each of the possible locations. Assuming that we have N possible locations $\{x_1, ..., x_N\}$to sample from and we currently have $\mathbf{m} = \{m_1, ..., m_N\}$ number of samples from each of the possible locations, and the current estimate learned from the samples is $h(x)$. Let $e_i = \{0, .., 0, 1, 0, ..., 0\}$ with a 1 in the i^{th}row denoting a sample at x_i. In the re-sampling step, we generate random samples from $h(x)$ at $\{x_1, ..., x_n\}$ of size $\mathbf{m} + e_i$. From each of these samples, we get an estimate of the threshold.

Computation For each possible location, we can then compute the distribution of the threshold estimate when we take an additional sample at that location.

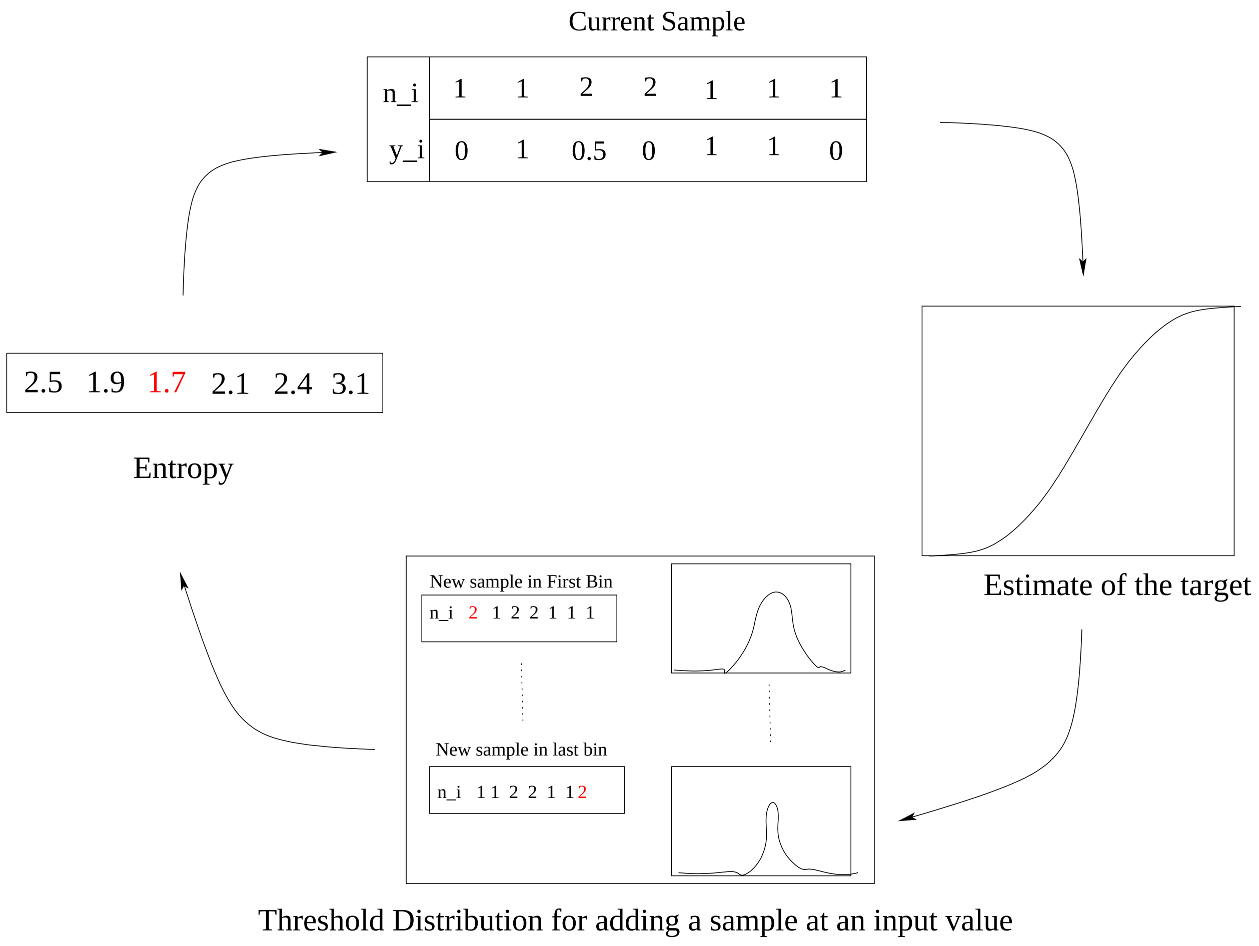


Figure 3.4: MinEnt Adaptive Sampling Algorithm

Evaluation From the threshold distribution at each location, we can compute the entropy of the threshold estimate and take the next sample at the location level which minimizes the entropy of the threshold.

The framework is shown in Figure 3.4

In practice, we first take one sample in each of the possible locations and then adaptively choose the rest. As we continue to take more samples, the estimate of the target would be closer to the real target. This way, the decision of the adaptive sampling would become more accurate as we take more samples.

3.5.3 Extensions of the Framework

The MinEnt framework can be easily extended to accommodate sampling at any input value or any cost function. For sampling at any stimulus level, during the re-sampling step, we also compute for the case when we sample in between two of the input values. In the Evaluation step, we can replace the entropy of the threshold by any other statistic, like variance. This way we can focus on different aspects as dictated by the application.

3.6 Comparison

We compared the MinEnt adaptive algorithm with the Staircase method [Cornsweet 1962] and the Best PEST procedure [Pentland 1980]. We compare the performance for various shapes of the target function, as shown in Table 3.2, and for various values. The testing is done only on known target functions, as it is not possible to use a fixed dataset for adaptive testing. The target functions that we use provide us with a model for the problem. There are other factors in the real world applications, for example the concentration of the subject being tested in the GRE test can vary over time. We do not consider these factors in the tests.

Linear	$f(x) = x$
Weibull	$f(x) = (1 - e^{-x^2})$
NormCDF	$f(x) = \Phi(\frac{x-0.5}{0.25})$
Student-t	Student's t distribution with 10 dof
Quadratic	$f(x) = x^2$
TR	$f(x) = \begin{cases} 2x^2 & x \leq 0.5 \\ 1 - 2(1-x)^2 & x > 0.5 \end{cases}$
Exponential	$f(x) = \frac{1}{1+e^{\frac{a-x}{b}}}, a = 0.5, b = 0.05$
Exponential2	$f(x) = \frac{1}{1+e^{\frac{a-x}{b}}}, a = 0.4, b = 0.1$

Table 3.2: Various Shapes of Target Function Used in the Comparison

Table 3.3: Comparison of MinEnt with Existing Methods for Estimating 50% Threshold

	Equal-Logfit	Equal-MonFit	Staircase	PEST	MinEnt
Linear	0.01(0.07)	0.011(0.037)	0.004(0.25)	0.003(0.04)	0.026(0.02)
Weibull	0.01(0.06)	0.016(0.032)	0.001(0.23)	0.073(0.012)	0.025(0.012)
NormCDF	0.017(0.03)	0.011(0.01)	0.001(0.25)	0.053(0.09)	0.003(0.01)
Student-t	0.005(0.01)	0.006(0.02)	0.005(0.24)	0.059(0.01)	0.036(0.01)
Quadratic	0.079(0.04)	0.079(0.01)	0.06(0.23)	0.116(0.01)	0.08(0.01)
TR	0.004(0.04)	0.008(0.01)	0.001(0.25)	0.041(0.01)	0.012(0.01)
Exponential	0.006(0.01)	0.003(0.01)	0.001(0.25)	0.043(0.002)	0.002(0.001)
Exponential2	0.003(0.03)	0.003(0.005)	0.003(0.2)	0.005(0.005)	0.003(0.004)

The Staircase method only works for certain values and form of the threshold, and so it is not included in all the comparisons. We test the performance of the various methods in estimating the 50% threshold and the 61.2% threshold. For simplicity, all the methods were restricted to sample from a fixed grid of equally spaced locations and were given a budget of three times the number of locations. For baseline comparison, we also compare with the simple method, which allocates equal number of samples in each location and then uses parametric estimation and non parametric estimation.

The results of the comparison are shown in Tables 3.3 and 3.4. The two values in each cell of the table represent the bias and the variance of the estimate of the threshold obtained by each algorithm. The variance is given in the parenthesis and the total error would be composed of the two parts. To compare the algorithms on a target function, we need to compare both the bias and the variance. Lower variance denotes convergence in the estimate, while low bias denotes consistency.

MinEnt achieves lower variance for most of the comparisons. Its bias is at least comparable if not better than the existing methods. For the 50% threshold estimation, the Staircase method has a very low bias. This is because the method is unbiased, however its variance is significantly higher than the other methods, leading to a high error value.

In Figures 3.5-3.12, we compare the mean square error in the estimate of the threshold computed by PEST and MinEnt algorithm as we obtain additional samples as dictated by the algorithms. The starting point for both the algorithms is the same,

Table 3.4: Comparison of MinEnt with Existing Methods for Estimating 61.2% Threshold

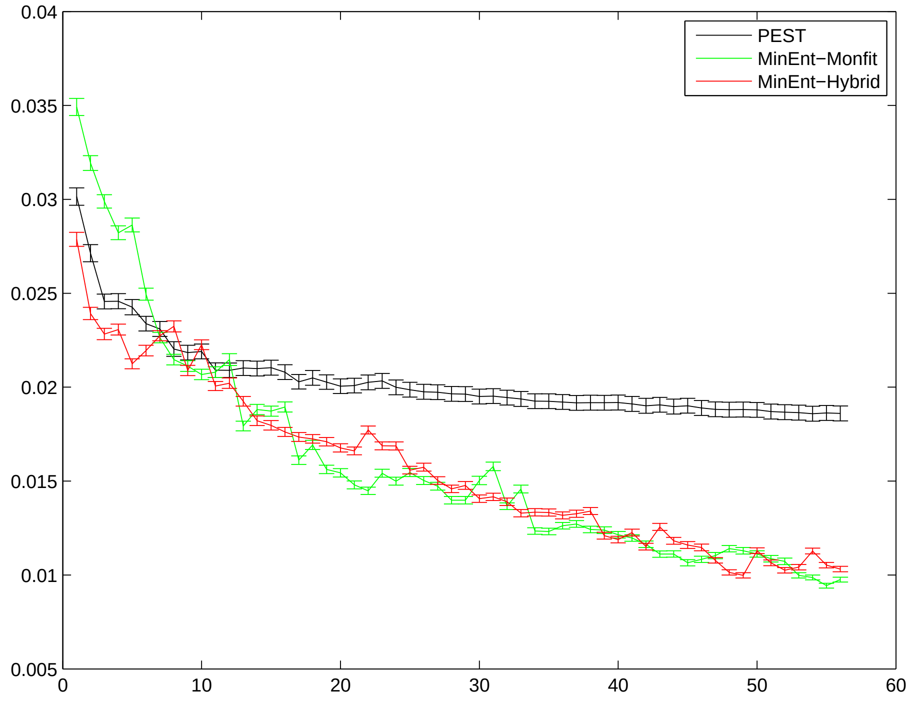
	Equal-Logfit	Equal-MonFit	PEST	MinEnt
Linear	0.015(0.016)	0.0158(0.027)	0.06(0.009)	0.014(0.0137)
Weibull	0.0154(0.008)	0.0236(0.016)	0.0589(0.01)	0.0118(0.0048)
NormCDF	0.0103(0.006)	0.0143(0.017)	0.07(0.013)	0.0076(0.009)
Student-t	0.005(0.01)	0.022(0.018)	0.0292(0.013)	0.0174(0.01)
Quadratic	0.0956(0.005)	0.0899(0.008)	0.1017(0.008)	0.0601(0.007)
TR	0.0106(0.004)	0.0076(0.008)	0.0690(0.01)	0.0123(0.009)
Exponential	0.0086(0.002)	0.0024(0.002)	0.0429(0.004)	0.0044(0.0015)
Exponential2	0.0241(0.004)	0.0084(0.005)	0.0637(0.006)	0.0132(0.0032)

one sample per bin and the x-axis denotes the number of additional samples used.

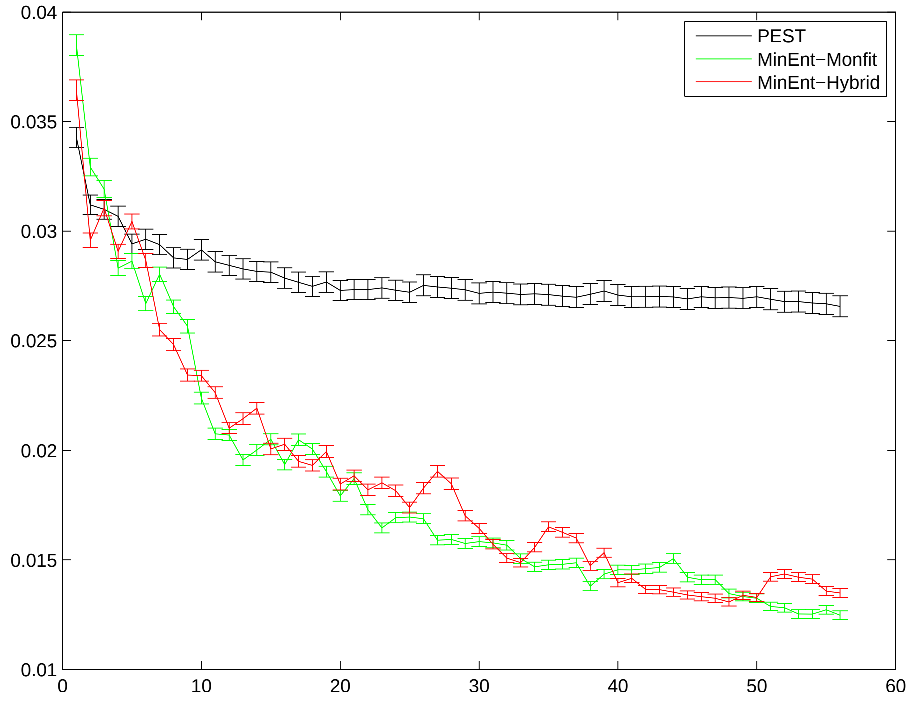
We can see that in some targets the error of PEST algorithm at the beginning is much lower than the other methods. This is partly due to the fact that PEST being a curve-fitting procedure has an advantage in cases where the target is either in or very close to the model used in PEST. MinEnt, however has a much better rate of convergence and is quickly able to focus on the threshold and reduce the errors there. In almost all of the cases, MinEnt with MonFit and HybridFit achieves lower error and better convergence.

3.7 Conclusions

We have described a new algorithm for using adaptive sampling in the non parametric estimation of a threshold-like parameter. This new algorithm allows us to use the power and consistency of the non-parametric methods in an adaptive setting. The performance of this new algorithm is better than the existing adaptive algorithms for parametric estimation. It achieves quick convergence and has low bias in its estimates, thereby making it ideally suitable for applications where the samples are very expensive.

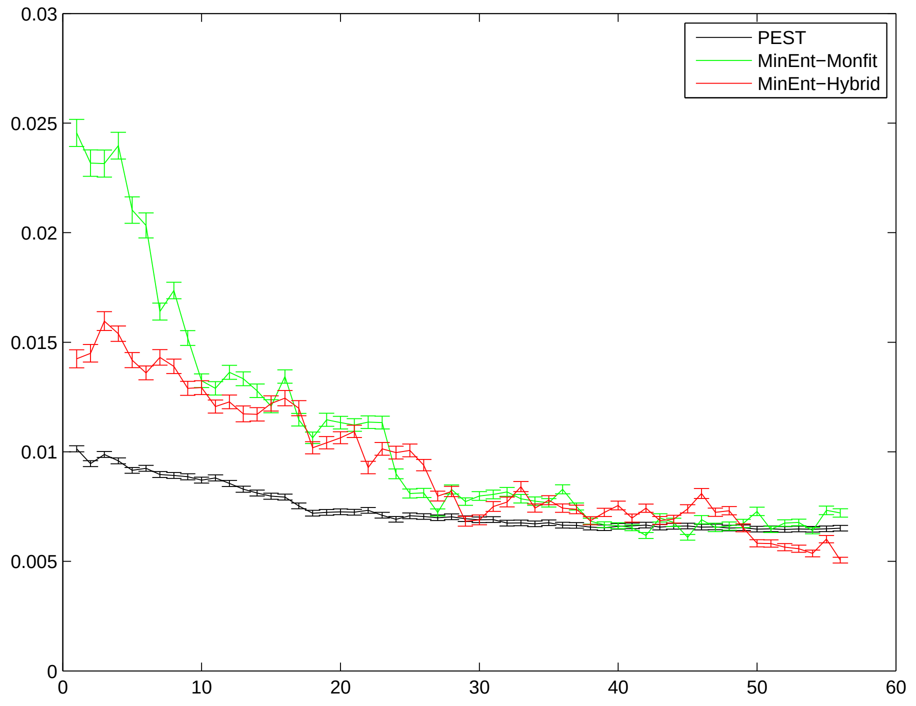


(a) MSE on 0.5 Threshold Estimation

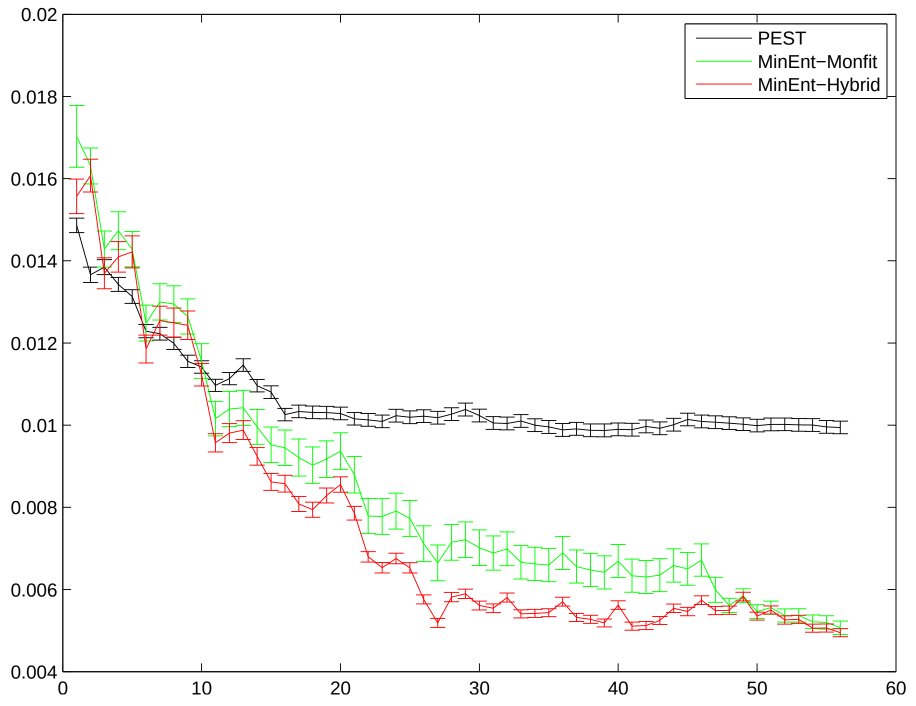


(b) MSE on 0.612 Threshold Estimation

Figure 3.5: Results on Linear Target

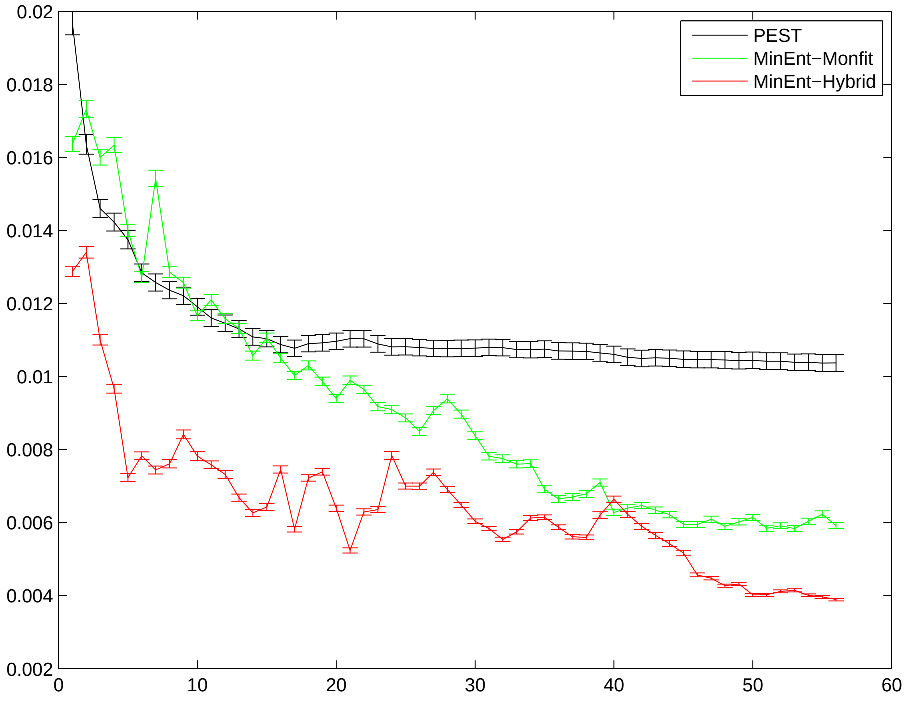


(a) MSE on 0.5 Threshold Estimation

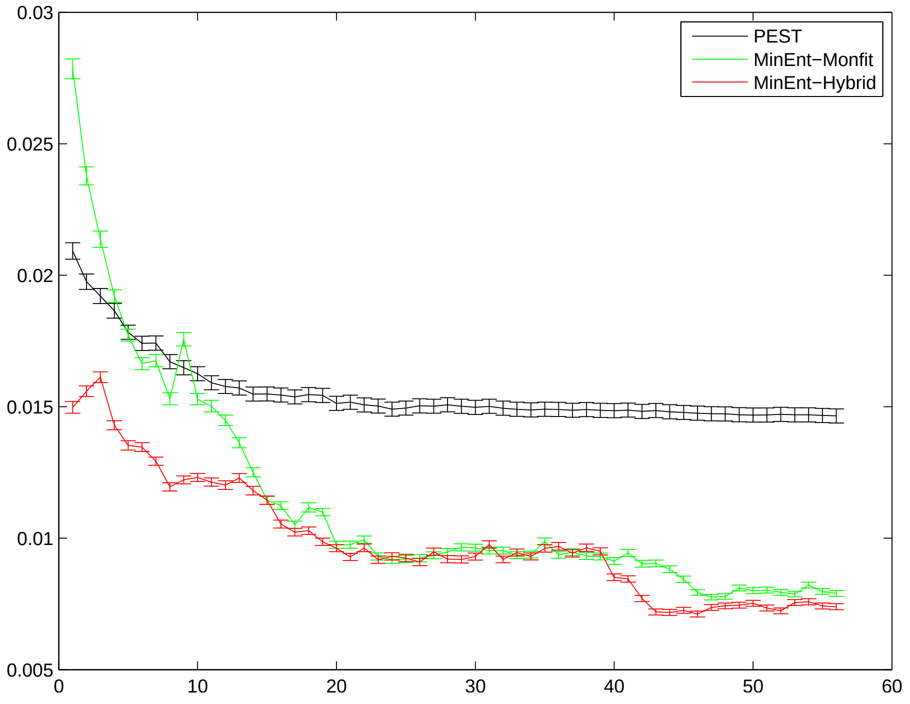


(b) MSE on 0.612 Threshold Estimation

Figure 3.6: Results on Weibull Target

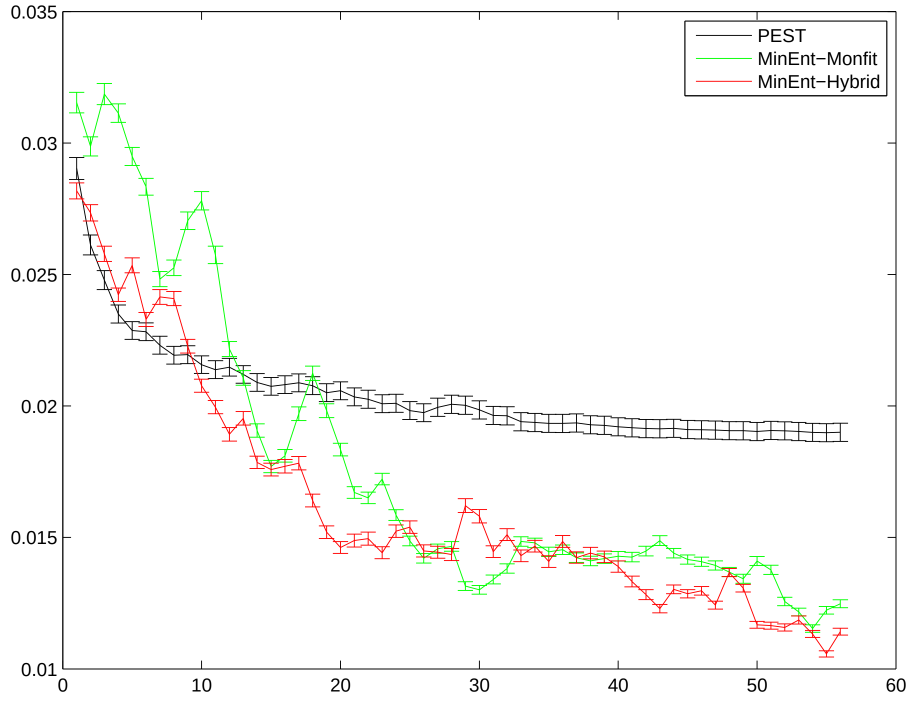


(a) MSE on 0.5 Threshold Estimation

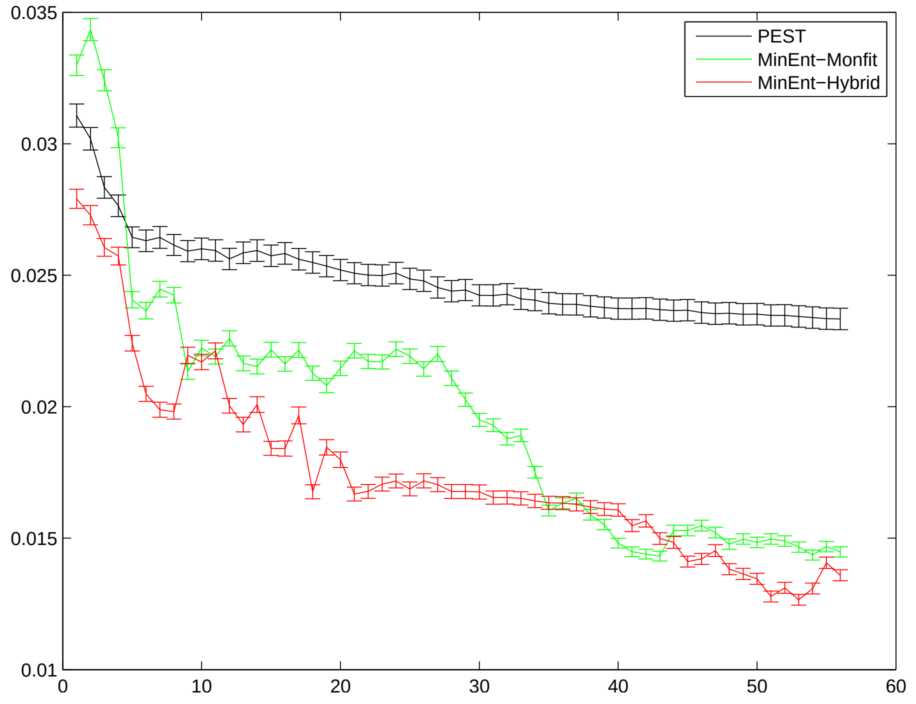


(b) MSE on 0.612 Threshold Estimation

Figure 3.7: Results on Normal Target

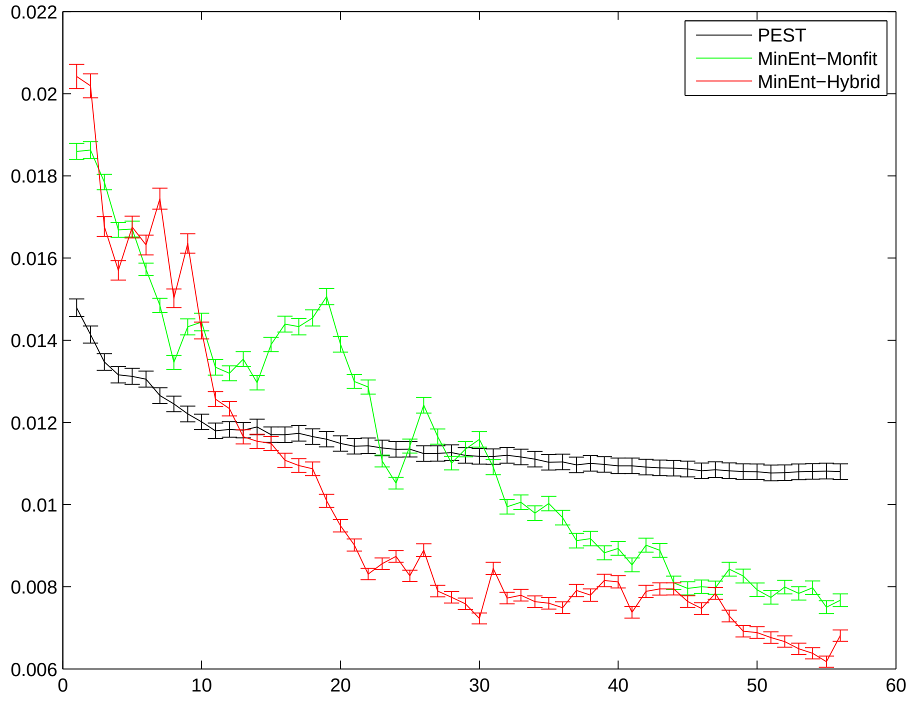


(a) MSE on 0.5 Threshold Estimation

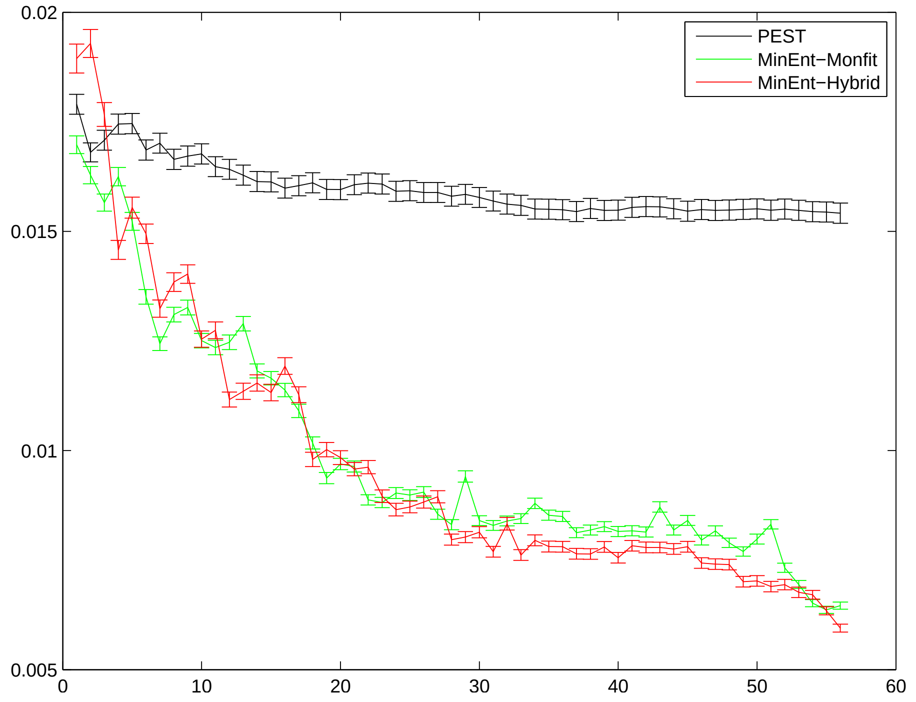


(b) MSE on 0.612 Threshold Estimation

Figure 3.8: Results on Student-t Distribution Target

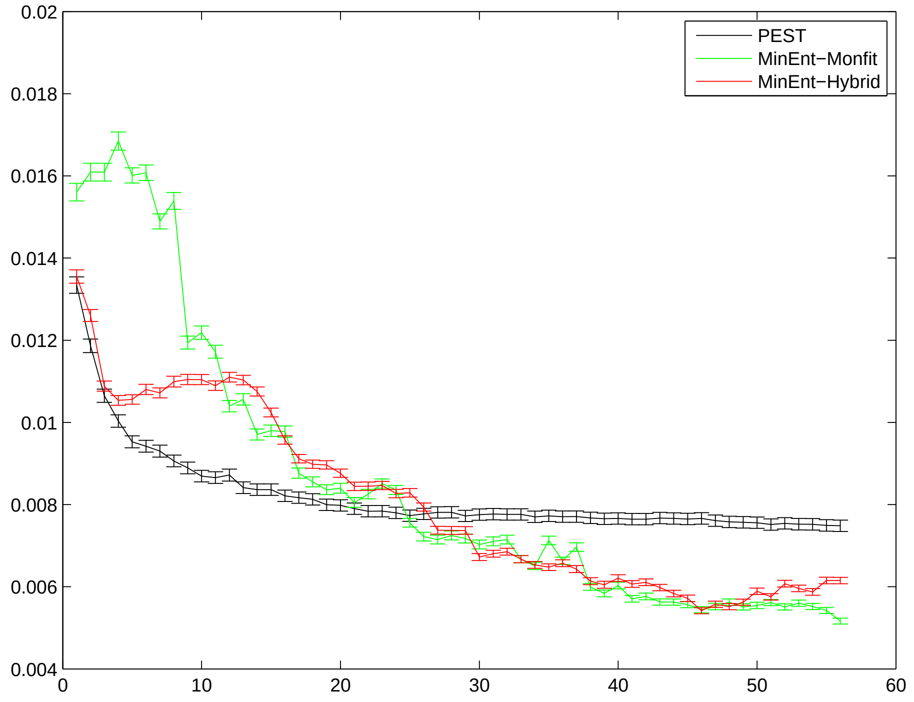


(a) MSE on 0.5 Threshold Estimation

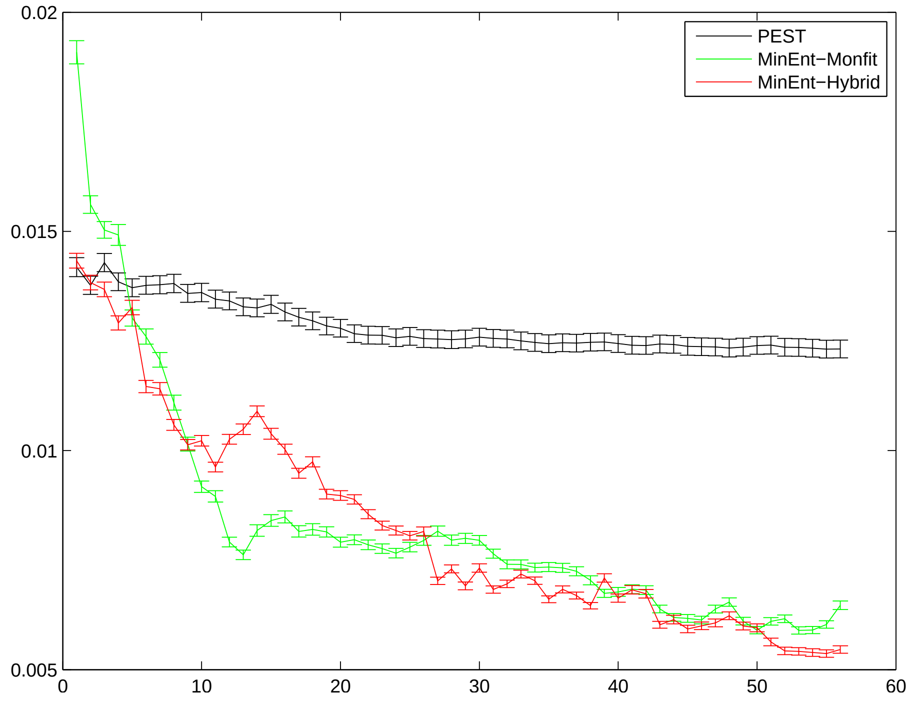


(b) MSE on 0.612 Threshold Estimation

Figure 3.9: Results on Quadratic Target

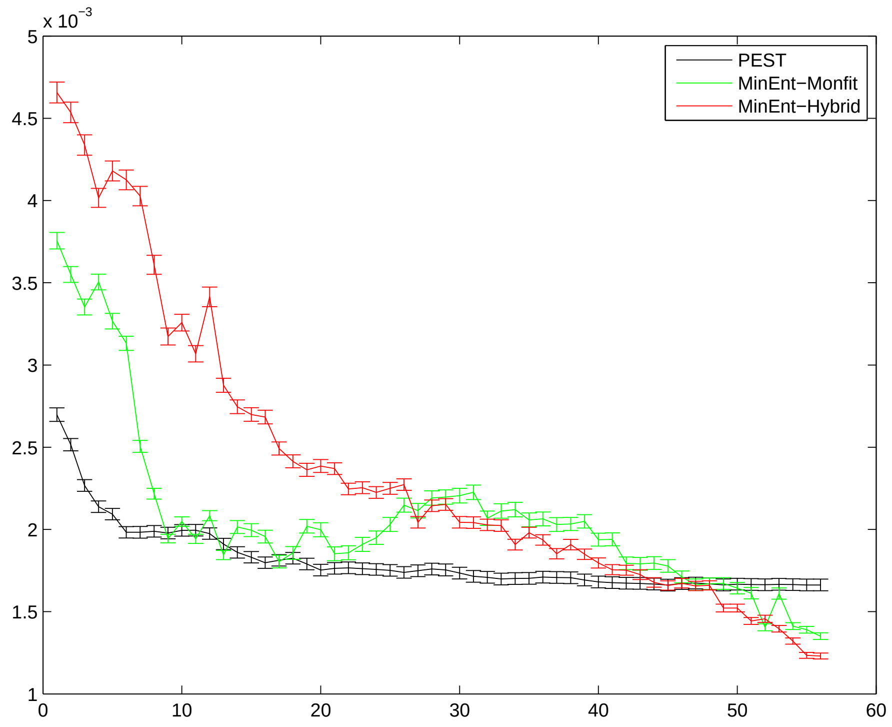


(a) MSE on 0.5 Threshold Estimation

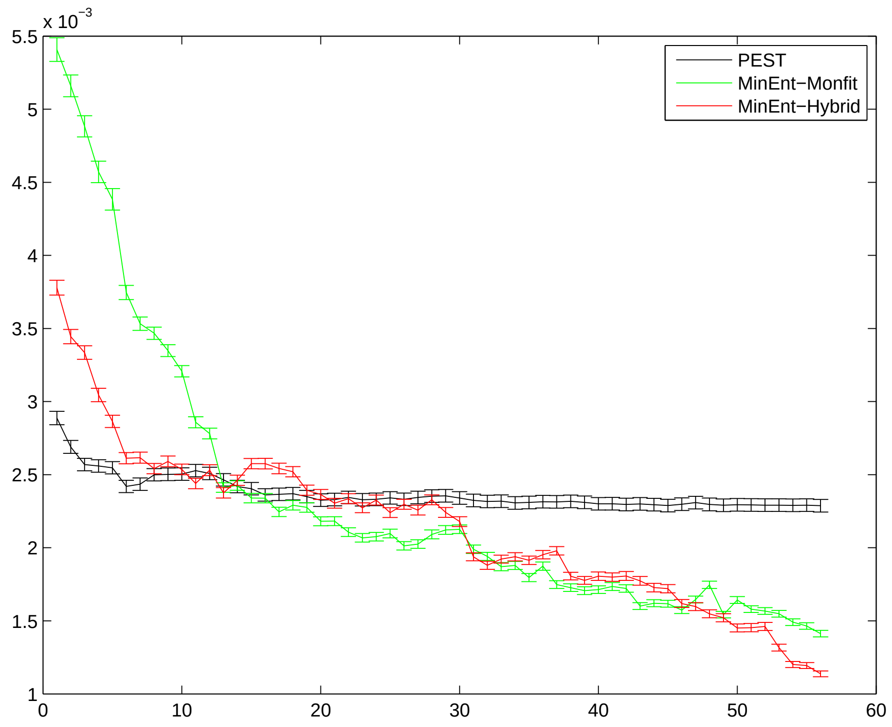


(b) MSE on 0.612 Threshold Estimation

Figure 3.10: Results on TR Target

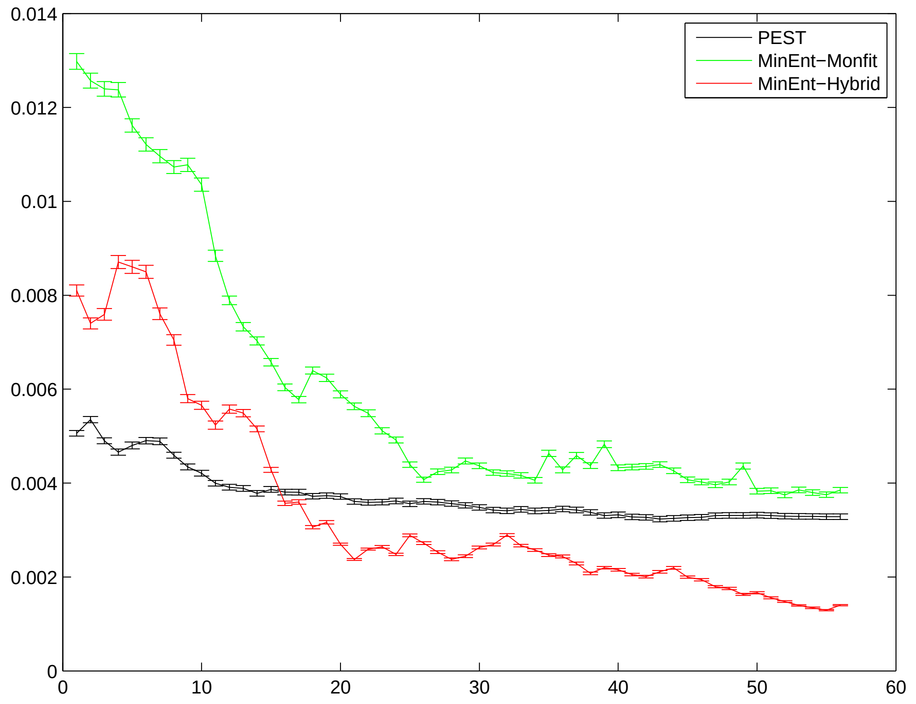


(a) MSE on 0.5 Threshold Estimation

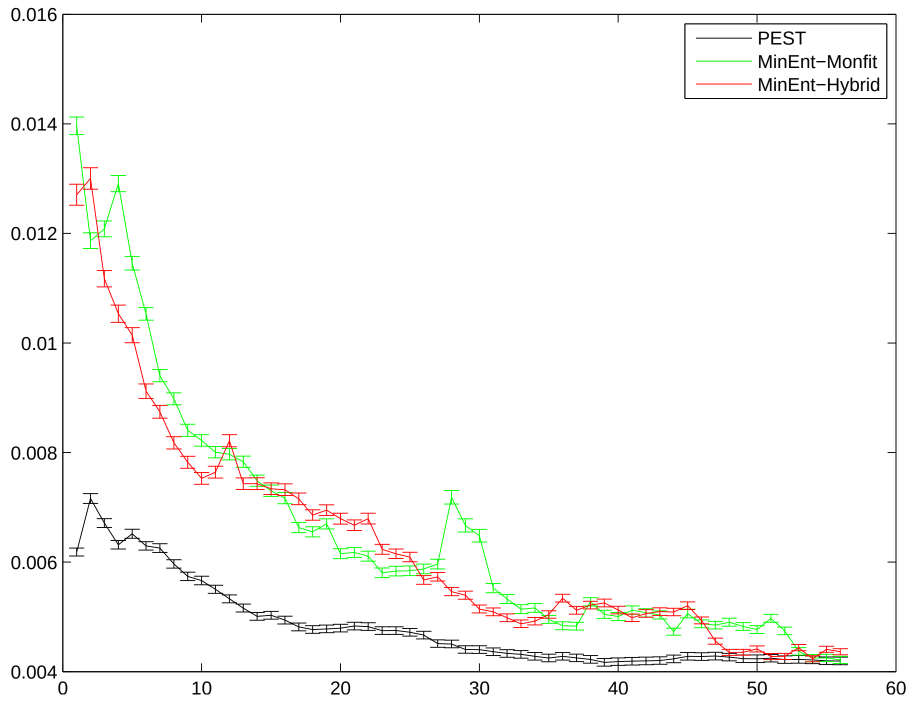


(b) MSE on 0.612 Threshold Estimation

Figure 3.11: Results on Exp Target



(a) MSE on 0.5 Threshold Estimation



(b) MSE on 0.612 Threshold Estimation

Figure 3.12: Results on Exp2 Target

Chapter 4

Data Cloning for Machine Learning

One of the major pitfalls in machine learning is that of selection bias. This is mostly introduced unconsciously due to the choices made during the learning process which often lead to over-optimistic estimates of the performance. We introduce a new methodology for *cloning* a dataset which can systematically reduce the selection bias.

Cloning produces a dataset of similar complexity and sufficiently independent from the original dataset. Using cloned dataset for model selection results in a consistent improvement over cross validation and its performance is much closer to the out-of-sample selection. Experimental results on a variety of learning problems shows that the cloning process results in a significant improvement in model selection over insample selection.

4.1 Introduction

While applying machine learning techniques to a real world problem, many decisions have to be made. These include which learning algorithm to use and how to select the parameters for the learning algorithm. This fine tuning of the learning process can be done based on domain-specific heuristics and prior knowledge about the problem. A lot of these decisions are, however, made based on the available data. This can lead to a "contamination" of the dataset and introduce selection bias in the learning process [Salzberg 1997].

This issue also arises while designing generic algorithms in machine learning research. Here the decision to be made is whether the new algorithm is better than existing methods. The "No Free lunch theorem" [Schaffer 1994, Rao et al. 1995, Wolpert 1996] states that no algorithm can universally be better than any other algorithm for all problems. It is therefore important to test the algorithms on a variety of problems coming from different fields in order to evaluate their effectiveness on commonly encountered types of problems. One such repository of real world datasets is the UCI Machine Learning Repository [Blake and Merz 1998]. The archive has been widely used by researchers all over the world as a primary source of machine learning data sets and has been cited over 1000 times. The main limitation of a dataset generated from a real world problem is that the number of samples are small, as they are expensive to obtain and label. The comparison between algorithms is usually done by taking different bootstrapped samples to get statistically significant results. This can introduce a great deal of selection bias especially since there are very few datasets and these have been used for a very long time now for comparing algorithms.

We introduce a methodology to replicate a dataset, i.e., to generate a dataset similar in difficulty to the original dataset but independent from it, so testing on them would not introduce additional selection bias. The cloned dataset can also be used for model selection while working on a real world application to avoid contamination of the available dataset. In this work, we do not focus on generating new unlabeled samples for the cloned dataset, but only on generating labels for them based on the existing dataset. [Shakhnarovich 2001] and [Shakhnarovich et al. 2001] looked at generating new unlabeled samples by learning the distribution of the dataset. These methods can be used along with our cloning techniques, but for the purpose of this work, we will focus on generating labels for new data-points.

4.2 Complexity of a Dataset

The complexity of a dataset is a measure of the difficulty it presents to a learning algorithm. It can be defined in terms of the complexity of a classifier required to describe the dataset. Li [2006] provides a framework for defining the complexity of the dataset with respect to a fixed learning model which allows for in-sample errors and accounts for them in the complexity.

Definition 1 *For a given learning model $\mathcal{H}$ and a positive constant λ, the data complexity of a dataset D with a lookup table is defined as*

$$C_{\mathcal{H},\lambda}(D) = \min |h| + \lambda e_D(h) : h \in \mathcal{H}$$

The constant λ is the complexity of implementing one lookup table with the learning model. This method is very closely related to the principle of minimum description length (MDL) [Rissanen 1978].

If we use support vector machine [Vapnik 2000] as our learning model, then the complexity of the dataset can be defined as the sum of the number of support vectors in the learned model and the number of in-sample points that were misclassified. This provides a practical measure for the complexity which can be easily computed. The other factors in the description of the hypothesis which include the description of the bias and the kernel computation program can be ignored when comparing the complexities as they remain constant for a fixed model.

4.3 Cloning a Dataset

The cloning process is a two step procedure. The first step is to create a *cloner function* which will generate examples which are similar to the ones generated by the target function that generated the original dataset. The second step is to generate new unlabeled points, classify them using the cloner function, and select points which would make up a dataset of similar complexity to the original dataset. As mentioned earlier, we will assume that we can generate new unlabeled points and so we will only focus on selecting points for the new dataset. We will discuss each of these steps in detail in the following sections.

4.4 Learning a Cloner Function

In this step, we will use a fixed learning model to learn a classifier from the dataset and then use it to create cloned copies. We will present two different approaches to generating a cloner.

4.4.1 AdaBoost.Stump

AdaBoost [Freund and Schapire 1995] is a popular and very successful machine learning algorithm. It is described in Algorithm 1. We use AdaBoost, along with decision stumps as the base learner, as a model to generate a cloner. The decision stump is a very simple learning model which bases its output on thresholding one input variable of the dataset. The advantage of using decision stump is that the hypothesis will be independent of the scale factor in the individual attributes. The learned hypothesis can be used to determine the labels for the generated dataset. This is a model free approach as we do not need to know the learning algorithm that would be used on the replicated dataset. Using a fixed but powerful learning model has an additional benefit of not introducing additional selection bias, as the model is chosen before looking at the dataset.

The performance of a learning algorithm on the cloned dataset gives an indication of the performance on the original dataset. We can bound the generalization performance which can be expected on the original dataset by the generalization performance of the classifier learned on the cloned dataset.

Theorem 5 *Given a dataset S generated from a distribution $\mathcal{D}$ and a target f_t, a cloner $f_c \in \mathcal{F}_c$ and for all hypothesis $f \in \mathcal{F}$, with high probability,*

$$P_{\mathcal{D}}[f(x) \neq f_t(x)] \leq P_{\mathcal{D}}[f(x) \neq f_c(x)] + P_S[f_t(x) \neq f_c(x)] + Complexity(\mathcal{F}_c)$$

The Theorem follows from the generalization bound for the cloner function and the triangle inequality, Theorem 5 bounds the out-of-sample error of a hypothesis learned on the cloned function. If the cloner is appropriately chosen such that it has a small in-sample error from a fixed class of functions, then the difference between the out-of-sample performance on the target becomes close to that on the cloner. One interesting observation here is that the bound is independent of the complexity of the hypothesis class $\mathcal{F}$, thereby allowing us to do model selection between a large number of models and incur a fixed and quantifiable amount of bias.

4.4.2 ρ-learning

ρ-learning [Nicholson 2002] is an algorithm which uses a learning model for data valuation and uses that valuation to classify new points. The ρ-value of a point for a given learning model is defined as the correlation between the performance on that point of a function with the overall generalization error of the function. Given that the points are generated according to a distribution $\mathcal{D}$ and a probability distribution over the learning model $\mathcal{G}$, $P_{\mathcal{G}}$, the ρ-value of a data point denotes the correlation between the error a function makes on that point and the overall out-of-sample error of that function.

$$\rho((x_0, y_0)) = corr_{g \in \mathcal{G}} \left([g(x_0) \neq y_0], P_{\mathcal{D}}[g(x) \neq y] \right)$$

A negative ρ-value for a data point implies that for better out-of-sample performance, it is optimal to misclassify that point. ρ-learning computes both $\rho(x, 1)$ and $\rho(x, -1)$ and decides the label based on the larger of the two values. The correlation is calculated with respect to the original dataset S, giving the classification rule as

$$y|x = sign \left(corr_g([g(x) \neq 1], P_{\mathcal{S}}[g(x) \neq y]) \right)$$

If we choose points with high correlation to the dataset to make up a new set, it will have a good correlation with the original dataset.

Theorem 6 *Given two datasets, $S_1$and S_2, and a negation symmetric learning model*

$\mathcal{G}$ (i.e., $P_{\mathcal{G}}[g] = P_{\mathcal{G}}[-g]$,$\forall g \in \mathcal{G}$) we have

$$Corr_g(\nu_{S_1}, \nu_{S_2}) = \frac{1}{|S_1|^2} \sum_{x \in S_{11}} Corr_g(\nu_x, \nu_{S_2})$$

PROOF For general random variables $\{X_i\}_{i=1}^N$and Y, we have

$$Corr(\sum_{i=1}^{N} X_i, Y) = \frac{\sum \sigma_{X_i} Corr(X_i, Y)}{\sum \sigma_{X_i}} \tag{4.1}$$

For negation symmetric learning models, we have that $P_{\mathcal{G}}[\nu_x = 1] = 1/2$, therefore $Var_g(\nu_x) = 1/4$.

$$\nu_{S_1} = \frac{1}{|S_1|} \sum_{x \in S_1} \nu_x$$

By 4.1, we have the theorem.

□

A high correlation between the datasets would roughly imply that if a learning algorithm does well on the replicated dataset, it should do well on the original dataset as well and vice-versa. This means that the replicated dataset has approximately the same amount of "difficulty" as the original dataset. The choice of the learning model $\mathcal{G}$ is ad-hoc and in general it can be different from the model used for learning. If we use the same learning model for learning and replication, then we ensure that the replicated set is highly correlated with the original set. This would, however, introduce additional bias into the cloning process. Li et al. [2005] used ρ-values for categorizing the data and found that the concept of correlation holds independently of the model. We will use neural networks with 5 neurons in the hidden layer as our model for estimating the ρ-values.

4.5 Data Selection

The cloner function provides a mechanism which can be used to generate as many examples as needed. It is vital to choose the new data points in such a way that they represent the characteristics of the original dataset. It is always possible to generate a dataset of very high or very low complexity from a function by choosing the data points appropriately.

One of the important properties of a learning algorithm is its handling of noise in the dataset. It is therefore important to estimate the noise level in the original dataset and introduce the appropriate noise in the cloned dataset. We consider a generalization of this paradigm which was introduced by Li et al. [2005]. They categorized a dataset into three categories instead of the usual two, according to their usefulness to the learning process. A data point could either be *noisy* (i.e., it is misclassified), or *typical* (which implies that it is noiseless but not very informative about the target), or *critical* (which means that it should be given a higher priority in the learning process as it carries important information about the target). We can extend this concept and do categorization at various levels instead of just three. This would allow us to get a finer handle on the information content of the examples. We can then select the new points such that each category has an equal or almost equal representation in both the cloned and the original dataset. Li et al. [2005] introduced three practical methods for categorizing data, two of which are directly applicable to our cloning techniques.

4.5.1 ρ-values

The ρ-value of a point was primarily introduced by Nicholson [2002] as a measure of data valuation, i.e., measuring its importance to the learning process. A negative ρ-value implies that the point is mislabeled. A small positive value indicates that the point will be a critical point and a large value would indicate that the example is typical. In this approach, we select a new set which has similar distribution of ρ-values as the original dataset. Since we know the cloner function, we can easily compute the correlation coefficient based on this knowledge, instead of the usual cross-validation approach which is generally used for computing the correlation. This step ensures that the cloned dataset contains roughly the same amount of information about the cloner target as the original dataset does about the actual target, therefore the same amount of complexity.

4.5.2 AdaBoost Margin

AdaBoost [Freund and Schapire 1996] improves the accuracy of any base learner by iteratively generating an ensemble of base hypotheses. During its iterations, some examples are more likely to be misclassified than others, and are thus “hard” to learn [Merler et al. 2004]. AdaBoost maintains a set of weights for the training examples and gradually focuses on hard examples by giving them higher weights. At iteration t, the ensemble $\tilde{f}_t(x) = \sum_{s=1}^{t} \alpha_s h_s(x)$ is constructed, where h_s is a base hypothesis and α_s is the coefficient for h_s. The data weight $w_i^{(t)}$, proportional to $e^{-y_i \tilde{g}_t(x_i)}$, is thus tightly related to the ensemble confidence margin $y_i \tilde{g}_t(x_i)$, and shows how hard it is to get an example correct at iteration t [Freund and Schapire 1996]. Noisy examples tend to get misclassified a lot by base hypotheses and would have very large weights for most of the iterations. Easy examples, on the other hand, are almost always classified correctly and would have small weights. Thus, the average weight over different iterations can be used for data categorization.

The margin of a point is defined as the magnitude of the real output of the ensemble and is a measure of the confidence of the ensemble in the binary output. It is also an indication of the usefulness of the point to the learning process. In this approach, we select points which have similar distribution of AdaBoost margins as the original dataset. This ensures that the cloned dataset contains roughly the same amount of information about the cloner target as the original dataset does about the actual target function.

4.6 Data Engine

The Caltech Data Engine [Pratap 2003] is a computer system that contains several predefined models, such as neural networks, Support Vector Machines (SVM), and radial basis functions (RBF). When requested for data, it randomly picks a model, generates (also randomly) parameters for that model, and produces examples according to the generated target function. A complexity factor can be specified which controls the complexity of the generated target function. The engine can be prompted repeatedly to generate independent data sets from the same model to achieve small error bars in testing and comparing learning algorithms.

This approach has been used for comparing the performance of learning approaches [Li et al. 2003]. It allows us to obtain statistically significant comparison. DEngin is the first step in data cloning. It generates data according to the specified complexity. Genuine data cloning should however uses the existing data to replicate its complexity and other characteristics. We will use the DEngin to generate typical learning problems in order to evaluate the effectiveness of the cloning process.

4.7 Experimental Evaluation

4.7.1 Artificial Datasets

We used the datasets described in 2.6.5 along with the following artificial datasets for evaluating the performance of the cloning process.

Ring Norm [Breiman 1996] is a 20-dimensional, 2-class dataset. Each class is drawn from a multivariate normal distribution. Class 1 has mean zero and covariance 4 times the identity. Class 2 has mean $(-a, -a, .. -a)$ and unit covariance, where $a = 2/\sqrt{20}$.

DEngin [Pratap 2003] DEngin generates multiple target functions for evaluating the effectiveness of the cloning process over a variety of targets.

4.7.2 Complexity of Cloned Dataset

In this test, we test the relationship between the complexity of the cloned dataset and the complexity of the original dataset. Ideally we would expect the cloned dataset to have a similar complexity to the original dataset. We also test the usefulness of the data selection step in the cloning process.

4.7.2.1 Experimental Setup

We use the artificial targets to generate multiple datasets from a fixed target and compute the histogram of the complexity using the support vector machine model with Gaussian kernel. We compare this to the histogram of the complexity of a cloned dataset.

4.7.2.2 Results

Figures 4.1-4.8 show the histogram of the complexities of the original problem and the cloned datasets. We normalize the complexity by the number of samples in the original dataset so that it is a number between 0 and 1. In each of the figures, the top plot shows the distribution of the complexity of a dataset generated from the original target. The middle and the bottom figure shows the distribution of the complexities of cloned datasets generated from one original dataset. The expected complexity of the cloned dataset is very close to that of the dataset used to generate the cloner. We observe that the complexity of the cloned dataset are distributed in a very similar way to the original problem.

The complexity of the TwoNorm and RingNorm datasets is quite low, which is expected as the two datasets are generated from a Gaussian distribution, which is the model used for the estimation of the complexities. This kind of behavior is possible but highly unlikely for a real-world dataset.

Figure 4.9 shows the histograms of the cloned dataset for the YinYang problem without the data selection step. It clearly shows the advantage of the data selection step, since the randomly selected dataset has a much lower complexity than the original dataset. .

4.7.3 Cloning for Model Selection

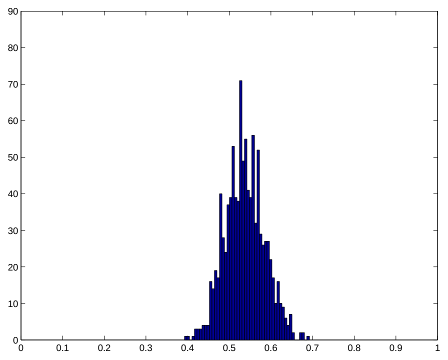
To evaluate the effectiveness of the cloning procedure for model selection, we compare it with existing methods for model selection and compare the generalization performance under the two methods. Cross validation [Kohavi 1995] is the most popular method for model selection in machine learning. The optimal method would be to do an out-of-sample model selection. However this is not practical, as it would require additional training samples which can be better used in the training stage. Since we are testing on artificial datasets, we can evaluate the Oracle solution and use it for baseline comparison.

4.7.3.1 Experimental Setup

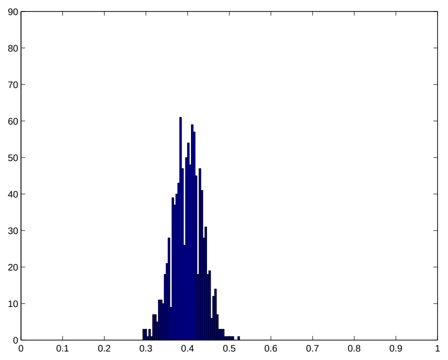
The model selection problem that we consider is the selection of the optimal parameters for a SVM classifier with the RBF kernel. There are two parameters that need to be selected and this is usually done by using cross validation. The Oracle method uses extra training samples for selecting the optimal parameters, as discussed it is not practical, but we will use it as a baseline for comparison. In the cloning approach, an AdaBoost of decision stumps is generated from the dataset, and that is then used to generate multiple cloned copies of the training set and a large test set. The optimal parameters are then chosen using the cloned dataset, similar to the Oracle procedure. The average value of the generalization error which is defined as the difference between the out-of-sample error and the in-sample error, is then computed.

4.7.3.2 Results

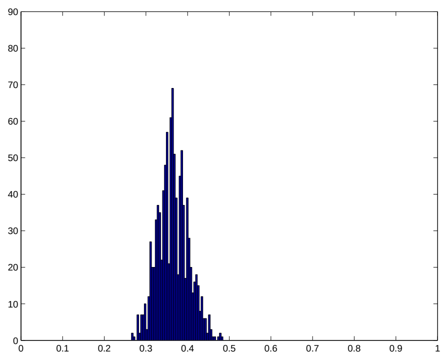
Table 4.1 summarizes the results of LeftSin dataset using the AdaBoost-Stump and ρ-learning as cloners. All results are averaged over 50 independent runs and the generalization errors are reported in percentage. We compare the 4 methods by varying the number of models that they have to select from. The models are nested,



(a) Histogram of Complexity of Original Dataset

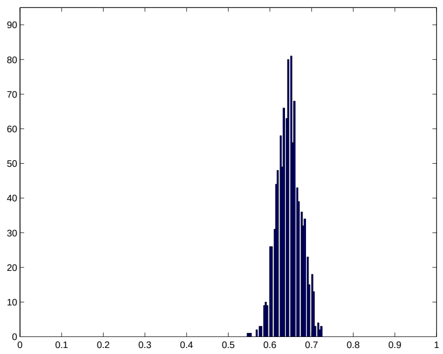


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.4265

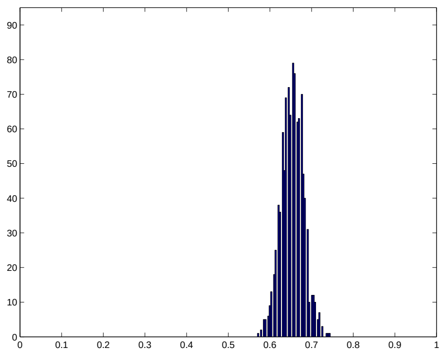


(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.4625

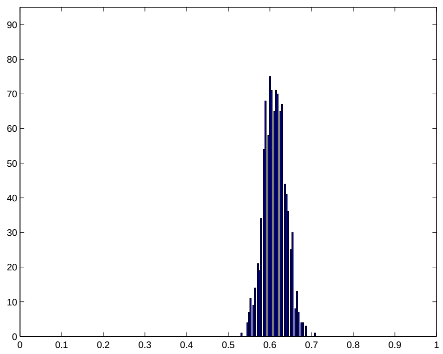
Figure 4.1: LeftSin with 400 Examples



(a) Histogram of Complexity of Original Dataset

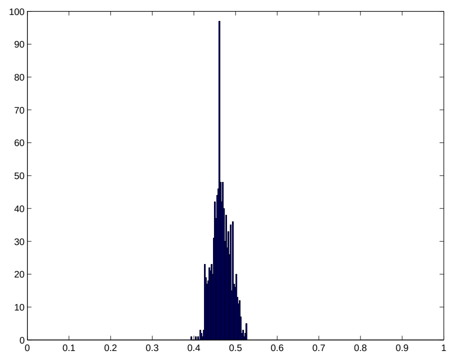


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.69

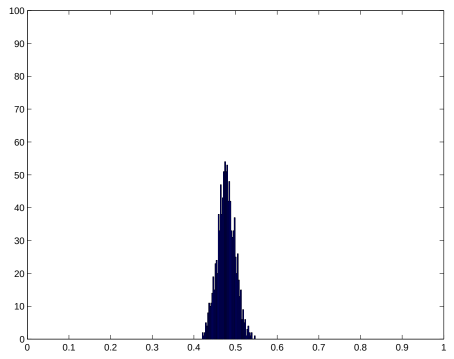


(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.69

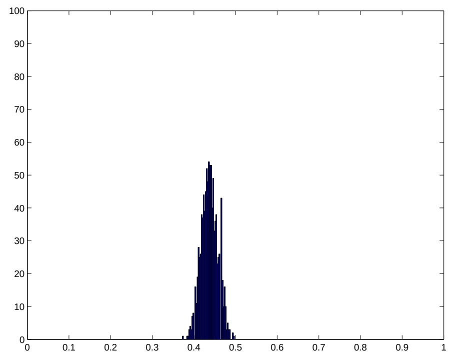
Figure 4.2: LeftSin with 200 Examples



(a) Histogram of Complexity of Original Dataset

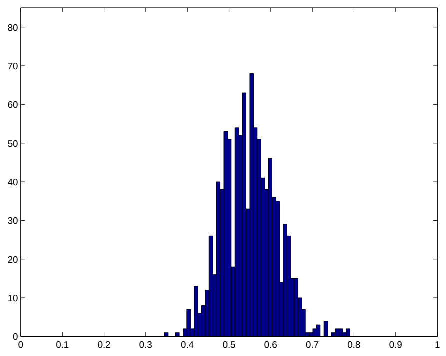


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.4523

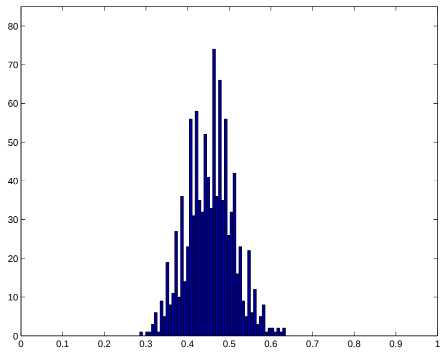


(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.45235

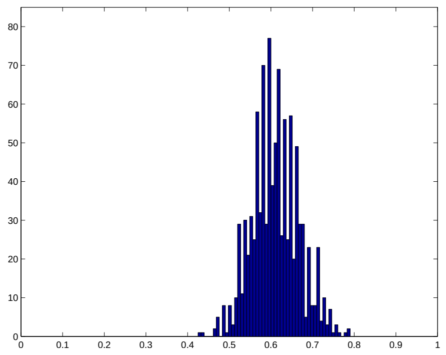
Figure 4.3: YinYang with 400 Examples



(a) Histogram of Complexity of Original Dataset

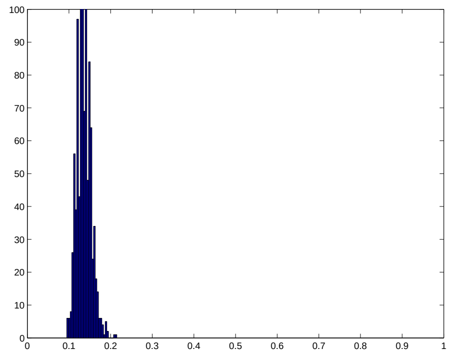


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.475

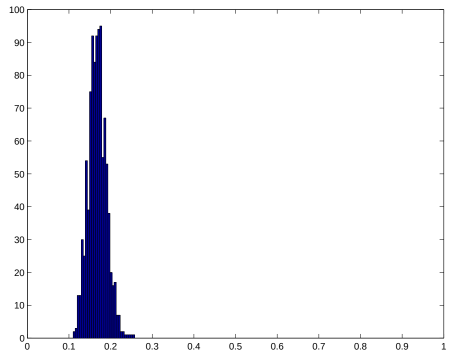


(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.475

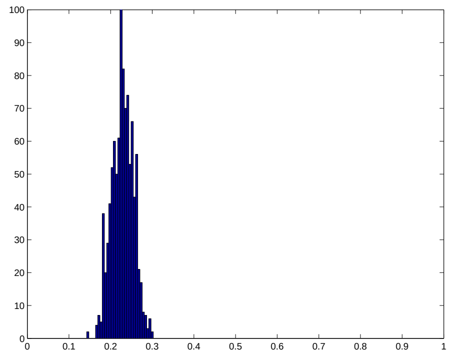
Figure 4.4: YinYang with 200 Examples



(a) Histogram of Complexity of Original Dataset

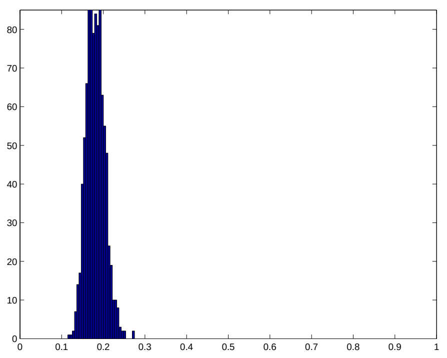


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.125

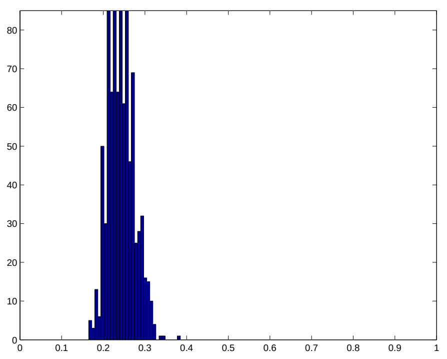


(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.125

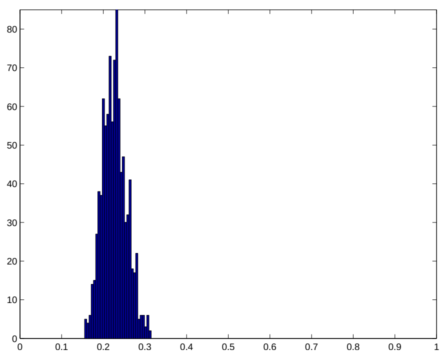
Figure 4.5: TwoNorm with 400 Examples



(a) Histogram of Complexity of Original Dataset

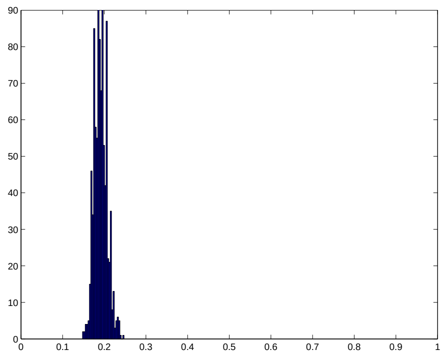


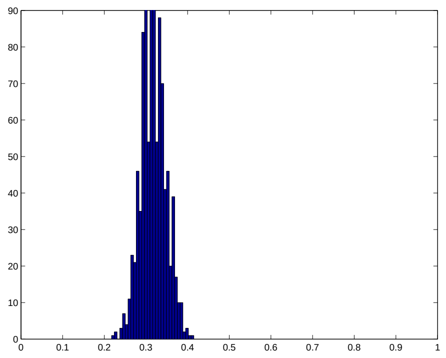
(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.19



(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.19

Figure 4.6: TwoNorm with 200 Examples


(a) Histogram of Complexity of Original Dataset


(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.1875

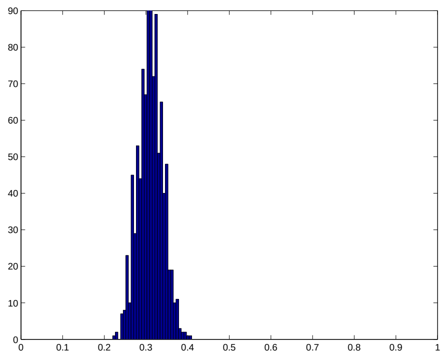
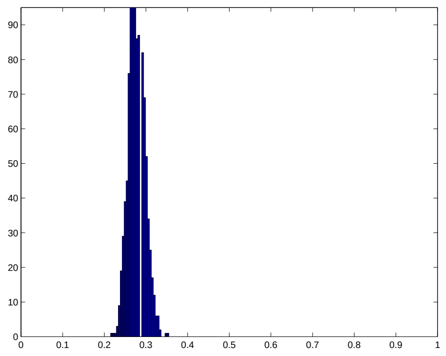
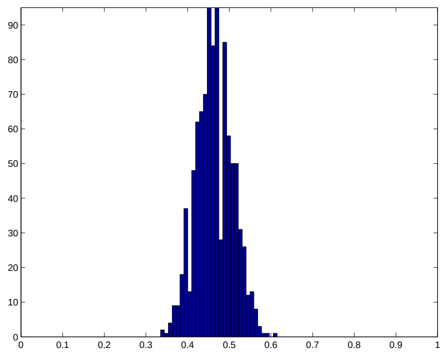

(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.1875

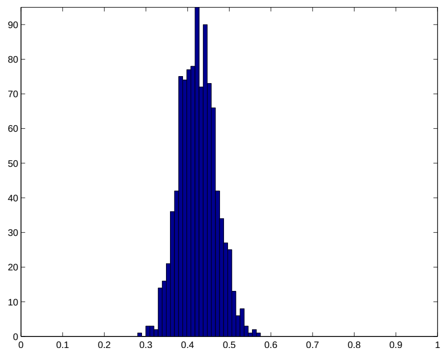
Figure 4.7: RingNorm with 400 Examples



(a) Histogram of Complexity of Original Dataset



(b) Histogram of Complexity of Dataset cloned by AdaBoost-Cloner from a Dataset of complexity 0.32



(c) Histogram of Complexity of Dataset cloned by ρ-Cloner from a Dataset of complexity 0.32

Figure 4.8: RingNorm with 200 Examples

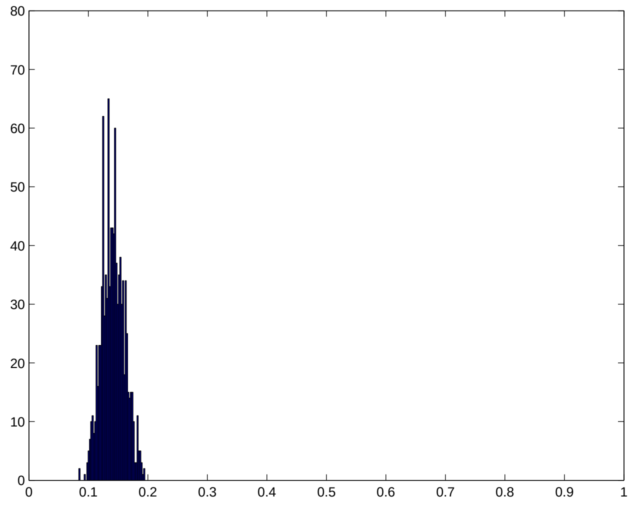


Figure 4.9: YinYang with 400 Examples without Data-Selection

Table 4.1: Performance of Cloning on LeftSin Dataset

Method/# choices	5	10	15	20
In-Sample	9.2217 ± 0.1298	14.8387 ± 0.09	15.6 ± 0.074	15.4587 ± 0.0941
CV	3.8450 ± 0.2436	5.2630 ± 0.4621	4.83 ± 0.4662	3.643 ± 0.301
ρ-cloning	2.2160 ± 0.1445	2.3808 ± 0.1082	2.723 ± 0.259	1.6102 ± 0.0936
AdaBoost-Cloning	2.7311 ± 0.1612	2.9412 ± 0.1340	3.1211 ± 0.421	2.012 ± 0.132
Oracle	1.803 ± 0.1316	2.0871 ± 0.0537	1.4557 ± 0.0695	1.0637 ± 0.0604

Table 4.2: Performance of Cloning on RingNorm Dataset

Method/# choices	5	10	15	20
In-Sample	2.3976 ± 0.0024	2.4836 ± 0.0018	2.4226 ± 0.002	2.5324 ± 0.0027
CV	1.6166 ± 0.0122	1.7744 ± 0.0126	1.5580 ± 0.0089	1.7348 ± 0.0071
ρ-cloning	1.4540 ± 0.0133	1.6772 ± 0.0151	1.3112 ± 0.0095	1.3591 ± 0.0074
AdaBoost-Cloning	1.4121 ± 0.021	1.6213 ± 0.0241	1.2336 ± 0.0067	1.2864 ± 0.0076
Oracle	1.1544 ± 0.009	1.1114 ± 0.0094	1.1101 ± 0.0063	1.0864 ± 0.0059

i.e., all choices available in the 5-model problem are available for selection in the 10-model problem, and are of increasing complexity. The 10-model problem contains all the models in the 5-model problem, plus 5 additional models which are more powerful.

We can see a consistent improvement by using cloning over cross validation, and the results are much closer to the Oracle approach, indicating that the cloned datasets are of similar complexity and sufficiently independent from the original dataset. The in-sample selection is consistently worsening as we increase the number of choices, which is expected. The cross-validation approach follows the same trend as the Oracle approach, i.e., its bias decreases as the bias of the Oracle approach decreases. The cloning approach, although not following the exact trend in the Oracle approach, gives results closer to the Oracle approach.

Table 4.2 summarizes the results of the 20-dimensional RingNorm dataset. We observe results similar to that of the LeftSin dataset, although the magnitude of errors here is much smaller.

4.7.4 Cloning for Choosing a learning algorithm

Here we test the cloning approach in choosing between two learning algorithms. We use the same artificial datasets and use cloned datasets to select between two learning algorithms. In an ideal setting, if we had access to the entire target function, we would then choose the algorithm which would have a lower out-of-sample error upon learning from that dataset. We will use this concept as the baseline for comparison.

4.7.4.1 Experimental Setup

We use the Caltech Data Engine to generate different target functions from various learning models. For each target, we generate a dataset of size 400. We can then clone each dataset and get a cloned set of the same size. We train a $2-5-1$ neural network and support vector classifier on the original and cloned datasets, and report how often the performance on the cloned set matches the Oracle decision (which uses "illegal" information about the target function to choose a model). We also report the generalization performance of a system which uses the cloning process to decide which algorithm to use.

Given a target function f and a dataset $S_1, ..., S_N$ generated from f, we generate the cloners $f_1^c, ..., f_N^c$ and cloned datasets $S_1^c, ..., S_N^c$. Given two learning algorithms $A_1$and A_2 which map a dataset S to a hypothesis $h = A_i(S)$ for $i = 1, 2$. Define $\pi_{ij} = e(A_i(S_j), f)$ and $\pi_{ij} = e(A_i(S_j^c), f_j^c)$, as the out-of-sample performances of the two models. We define the hit-rate of a cloning process as

$$h = \frac{1}{N} \sum_{i=1}^{N} ([\pi_{i1} < \pi_{i2}] = [\pi_{i1}^c < \pi_{i2}^c])$$

The hit-rate measures how often the decision of an algorithm on the cloned dataset matches the actual Oracle decision. The hit-rate of the in-sample process can also be similarly defined.

The performance of the Oracle chooser is defined as

$$Or = \frac{1}{N} \sum_{i=1}^{N} min\,(\pi_{i1}, \pi_{i2})$$

and the performance of the cloner is defined as

$$Cln = \frac{1}{N} \sum_{i=1}^{N} (\pi_{i1}[\pi_{i1}^c < \pi_{i2}^c] + \pi_{i2}[\pi_{i1}^c \geq \pi_{i2}^c])$$

A similar performance measure can be calculated for the selection which is done in-sample

$$IS = \frac{1}{N}\sum_{i=1}^{N}\left(\pi_{i1}[\nu_{i1} < \nu_{i2}] + \pi_{i2}[\nu_{i1} \geq \nu_{i2}]\right)$$

These quantities measure the overall performance of an algorithm which uses the methods to select a learning model. The Oracle method will have the lowest error measure, but it is not possible to implement that method.

4.7.4.2 Results

The cloning procedure was evaluated on 20 target functions generated using the date engine from different models. Figure 4.10 shows the hit-rates of the cloning method against the in-sample selection. A random choice would give a hit-rate of 50%. The cloning process has a higher hit-rate than the in-sample selection and is always better than random. Figure 4.11 summarizes the performance of the cloning process in terms of the average error obtained by choosing an algorithm based on the cloned dataset. There is a significant improvement in the selection process by using the cloner over the in-sample method.

4.8 Conclusion

We have presented a methodology to create cloned copies of datasets which can inherit the properties and structure of the original dataset, while minimizing the contamination of the original dataset. We showed that the cloned datasets have a similar complexity to the original datasets. We tested the cloning approach for the purpose of model selection and choosing a learning algorithm. The experimental results indicate that the technique can be very useful in machine learning.

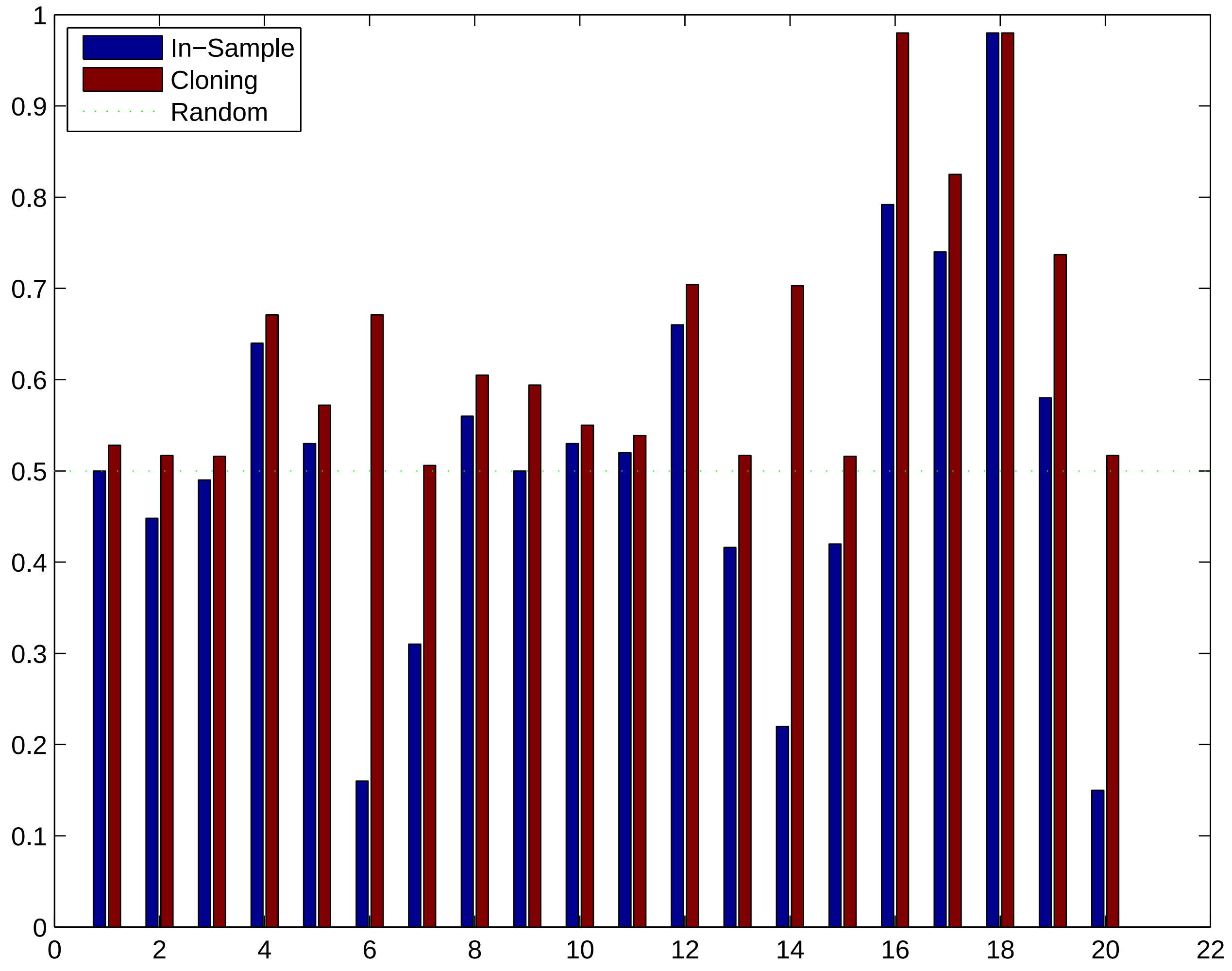


Figure 4.10: Comparison of Hit-Rate

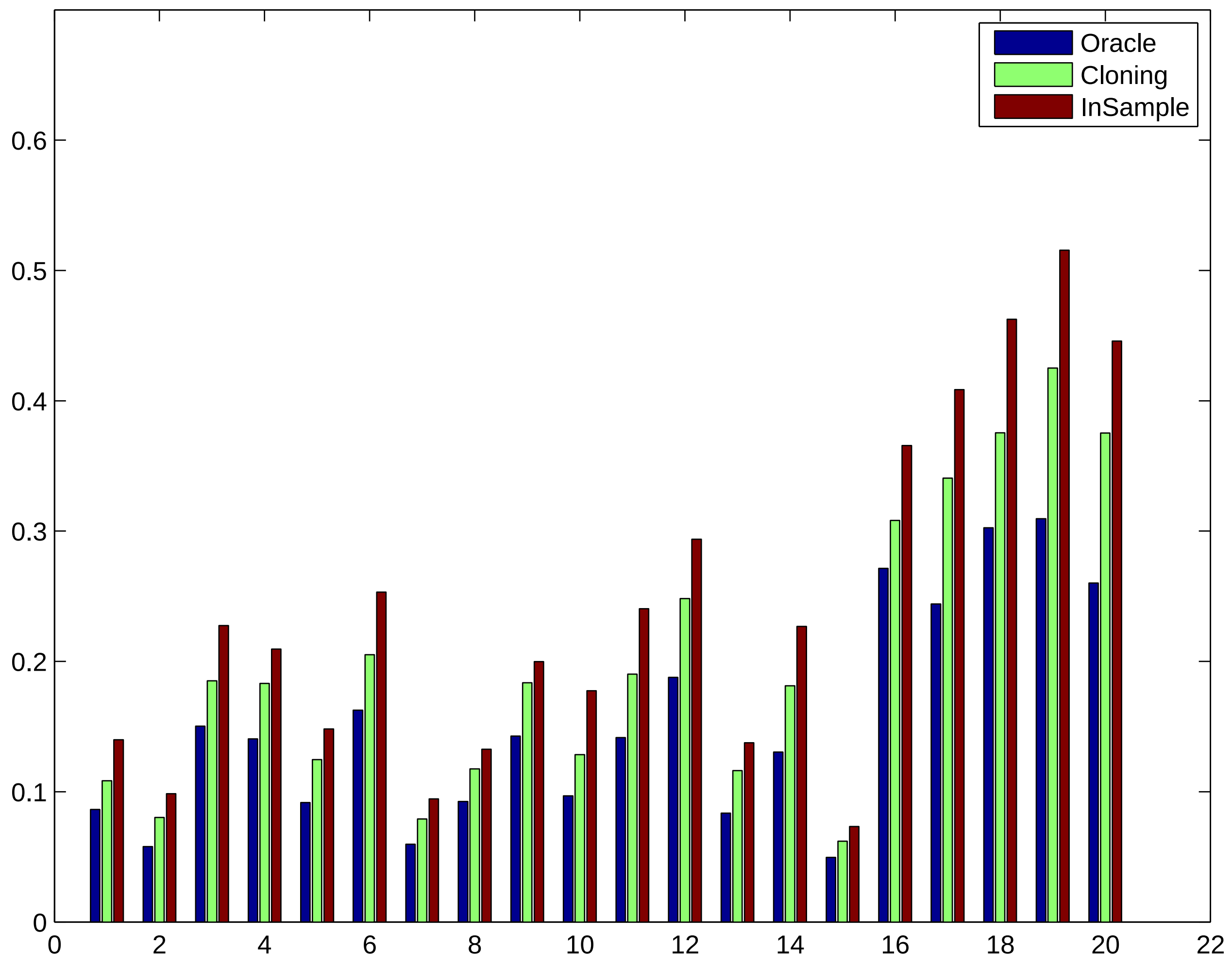


Figure 4.11: Comparison of Average Error by Selection Procedures

Bibliography

Y.S. Abu-Mostafa. Financial model calibration using consistency hints. *IEEE Transactions on Neural Networks*, 12(4):791–808, July 2001.

Y.S. Abu-Mostafa. Hints. *Neural Computation*, 7:639–671, 1995.

F. B. Baker. *The Basics of Item Response Theory.* Clearinghouse, 2001.

R.E. Barlow, D.J. Bartholomew, J.M. Bremner, and H.D. Brunk. *Statistical Inference Under Order Restrictions.* New York: Wiley, 1972.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992. ISBN 0-89791-497-X.

L. Breiman. Bagging predictors. Technical Report 421, Department of Statistics,, Sep 1994. URL `ftp://ftp.stat.berkeley.edu/pub/users/breiman/bagging.ps.Z`.

L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Department of Statistics, Apr 1996. URL `ftp://ftp.stat.berkeley.edu/pub/users/breiman/arcall96.ps.Z`.

L. Breiman. Arcing classifiers. *Annals of Statistic*, 26(3):801–824, Jun 1998.

L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7): 1493–1517, Oct 1999.

H.D. Brunk. Maximum likelihood estimates of monotone parameters. *The Annals of Mathematical Statistics*, 26(4):607–616, Dec 1955.

H.D. Brunk. On the Estimation of Parameters Restricted by Inequalities. *The Annals of Mathematical Statistics*, 29(2):437–454, June 1958.

D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, May 1994.

T.N. Cornsweet. The staircase method in psychophysics. *American Journal of Psychology*, 75:485–491, 1962.

C. Cox. Threshold dose-response models in toxicology. *Biometrics*, 43(3):511–23, Sep 1987.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2):182–197, 2002. ISSN 1089-778X. doi: 10.1109/4235.996017.

A. Demiriz, K.P. Bennett, and J. Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1–3):225–254, 2002.

V.V. Fedorov. *Theory of optimal experiments*. Academic Press, 1972.

D.J. Finney. *Probit Analysis*. Cambridge University Press, 1971.

R. Fletcher and C.V Reeves. Function minimization by conjugate gradients. *Computer Journal*, pages 149–154, 1964.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

A. J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pages 692–699, 1998.

G. Guo and H. Zhang. Boosting for fast face recognition. In *Proc. IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, 2001.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

S. Klein. Measuring, estimating, and understanding the psychometric function: A commentary. *Perception and Psychophysics*, 63(8):1421–1455, 2001.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1137–1143. Morgan Kaufmann, 1995.

V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30(1):1–50, February 2002.

V. Koltchinskii, D. Panchenko, and F. Lozano. Bounding the generalization error of convex combinations of classiers: balancing the dimensionality and the margins, 2000a.

V. Koltchinskii, D. Panchenko, and Fe. Lozano. Some new bounds on the generalization error of combined classifiers. In *NIPS*, pages 245–251, 2000b.

M.R. Leek. Adaptive procedures in psychophysical research. *Perception and Psychophysics*, 63(8):1279–1292, November 2001.

L. Li. *Data Complexity in Machine Learning and Novel Classification Algorithms*. PhD thesis, California Institute of Technology, Pasadena, CA, May 2006.

L. Li, Y.S. Abu-Mostafa, and A. Pratap. CGBoost: Conjugate gradient in function space. Computer Science Technical Report CaltechCSTR:2003.007, California Institute of Technology, Pasadena, CA, Aug 2003. URL `http://resolver.caltech.edu/CaltechCSTR:2003.007`.

L. Li, A. Pratap, H.-T. Lin, and Y. S. Abu-Mostafa. Improving generalization by data categorization. In A. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, volume 3721 of *Lecture Notes in Artificial Intelligence*, pages 157–168. Springer-Verlag, 2005. ISBN 3-540-29244-6.

F.M. Lord. *Applications of item response theory to practical testing problems.* Lawrence Erlbaum, 1980.

D.J.C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4:590–604, 1992.

O.L. Mangasarian and W.H. Wolberg. Cancer diagnosis via linear programming. *SIAM Nesw*, 23(5), September 1990.

L. Mason, P. L. Bartlett, and J. Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38(3):243–255, Mar 2000a.

L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In S.A. Solla, T.K. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems 12*, pages 512–518. MIT Press, 2000b.

S. Merler, B. Caprile, and C. Furlanello. Bias-variance control via hard points shaving. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(5):891–903, 2004.

A. Nicholson. *Generalization error estimates and training data valuation.* PhD thesis, California Institute of Technology, Pasadena, CA, May 2002.

S.E. Palmer. *Vision Science: Photons to Phenomenology.* Bradford Books/MIT Press, 1999.

A. Pentland. Maximum likelihood estimation: The best pest. *Perception and Psychophysics*, 28:377–379, 1980.

A. Pratap. *Data Engine for Machine Learning Research.* California Institute of Technology, 2003. URL `http://www.work.caltech.edu/dengin`.

J.O Ramsay. Monotone Regression Splines in Action. *Statistical Science*, 3(4):425–441, 1998.

R. B. Rao, D. Gordon, and W. Spears. For every generalization action is there really an equal and opposite reaction? analysis of the conservation law for generalization performance. In *Proc. 12th International Conference on Machine Learning*, pages 471–479. Morgan Kaufmann, 1995.

L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd International Conference on Machine Learning.* ICML, 2006.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

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

R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

R. E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

H Schwenk. Using boosting to improve a hybrid hmm/neural network speech recogniser. In *Proceedings of ICASSP'99*, pages 1009–1012, 1999.

H. Schwenk and Y. Bengio. Adaboosting neural networks: Application to on-line character recognition. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *International Conference on Artificial Neural Networks*, volume 1327 of *Lecture Notes in Computer Science*, pages 967–972. Springer, 1997.

G. Shakhnarovich. Statistical Data Cloning for Machine Learning. Master's thesis, Technion, 2001.

G. Shakhnarovich, R. El-Yaniv, and Y. Baram. Smoothed bootstrap and statistical data cloning for classifier evaluation. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 521–528, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

M.M. Taylor and C.D. Creelman. Pest: Efficient estimates on probability functions. *The Journal of the Acoustical Society of America*, 41(4A):782–787, April 1967.

V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2000.

A.B. Watson. Probability summation over time. *Vision Research*, 19:515–522, 1971.

F.A. Wichmann and N.J. Hill. The psychometric function i: Fitting, sampling, and goodness-of-fit. *Perception and Psychophysics*, 63:1293–1313, 2001.

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