CSC2515 Fall 2011
Boosting with Perceptrons and Decision Stumps

Richard Abrich
University of Toronto
richard.abrich@utoronto.ca

Paul Bovbel
University of Toronto
paul.bovbel@utoronto.ca

Abstract

In this paper, we analyze the performance of four different boosting algorithms and two families of weak classifiers on three datasets. Using this data, we try to evaluate the parameters for the weak classifiers, as well as the advantages of the different types of boosting algorithms.

1 Introduction

The data classification problem in machine learning can be stated as follows: given a set of examples composed of one or more features and a corresponding classification, predict the classification for future examples. We refer to the classified data as training data, and the data for which we wish to predict the classification as testing data.

A classifier is an algorithm whose input is a set of feature vectors, and whose output is the corresponding classification. Its error rate is defined as the proportion of incorrect classifications on training or testing data.

Much study has been devoted to the development of strong classifiers. These are computationally complex algorithms which are able to model the training data to an arbitrary degree of accuracy. By contrast, weak classifiers are computationally simple algorithms which are able to output the correct classification only slightly more than half of the time. Although not particularly useful on their own, it can been shown that a combination of many weak learners can also predict the correct classification to an arbitrary degree of accuracy [1]. One such method of combining many weak classifiers is known as boosting, which has the added advantage of avoiding the problem of overfitting [2].

Boosting is a general method that can be used to improve the performance of any learning algorithm. It is based on the principle of 'Divide and Conquer', in which a single complex task is solved by dividing it into many simpler tasks, and then combining the solutions. In particular, a boosting algorithm models the data by iteratively building a linear combination of weak classifiers. Each successive weak classifier is trained on a data set whose distribution is weighted in favour of examples which were misclassified by the previous one. This can be intuitively understood as allowing each weak classifier to concentrate on learning a particular aspect of the data. In this paper, we will consider four different boosting algorithms (AdaBoost, ArcGv, ArcGvMax, and Smooth Margin), and two families of weak classifiers (perceptrons and decision trees).

The rest of this paper is organized as follows: in section 2, we will describe the methodology of our experiments; in section 3 we will analyze the results; and in section 4 we will summarize our findings.

2 Methodology

2.1 Data Selection and Preprocessing

In order to analyze the relative performance of the different boosting and classification algorithms, we tested them on three different datasets, each exhibiting different properties. Although boosting can be used for multi-class classification as well as regression problems, our datasets involved binary classification only.

Toy Dataset This dataset was artificially generated for the purposes of testing algorithms quickly. It is well separated along three continuous and three discrete features with some random noise. Additionally, two uncorrelated features (one discrete and one continuous) are used in the dataset to test effectiveness of the base classifiers.

Molecular Biology Dataset Taken from the UCI machine learning repository [3], this dataset contains sequences of DNA to predict whether each sequence is an Intron or an Exon. Each element in the sequence represents a different feature.

Adult Census Data Dataset Also from the UCI repository [3], this dataset contains a mix of discrete and continuous census data from 1994 to predict whether an individual's salary is below or above $50,000 per year.

Since we test several types of classifiers on each dataset, certain modifications to the original data were required to accommodate their structural requirements.

Perceptrons Categorical features needed to be reinitialized as sets of binary features, since a perceptron trains on categories as continuous data. Some correlation was necessarily lost, as the classification algorithm is unable take into account which binary features were intrinsically related as members of a category.

Decision Trees The information gain algorithm requires modification to be able to process continuous data. However since our datasets contain a mix of both continuous and categorical features, we found it simplest to separate the continuous data into categories representing discrete intervals, with the value of zero being treated as a separate category.

2.2 Boosting Algorithms

The general boosting algorithm is defined as follows [2]:

> For N training examples, let $d_t(i)$ be the weight on training example i on the t^{th} round of boosting, and initialize $d_1(i) = 1/N$ for all $i \in 1 \dots N$.
>
> Let h_t be the hypothesis on the t^{th} round, and let α_t represent the weight of the t^{th} hypothesis. For $t \in 1 \dots T$ rounds of boosting:

1. Calculate h_t with respect to d_t according to the weak classifier(s) being used
2. Calculate α_t according to the rules of the boosting algorithm being used
3. Update d_{t+1}, where Z_t is a normalizer such that $\sum_i^N d_{t+1} = 1$:

$$d_{t+1}(i) = \frac{d_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Once training is complete, the final classification H for example x is as follows:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

The margin of a round of boosting describes the correctness of the algorithm's classification, as well as the algorithm's confidence in its output, and is defined as follows:

$$\text{margin}_f(x, y) = y\frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t}$$

where $y \in \{-1, 1\}$ is the class label, and α_t and h_t are the weight and output of the t^{th} weak classifier, respectively.

The absolute value of the margin corresponds to the algorithm's confidence in its classification: the larger the absolute value, the more confident the algorithm was in its decision. The sign of the margin indicates whether or not the decision was correct: a positive value corresponds to a correct classification, and a negative value corresponds to an incorrect one. The cumulative distribution of margins is a useful tool for analyzing the performance of a boosted classifier [Reyzin].

2.2.1 AdaBoost

In AdaBoost [2] (from 'Adaptive Boosting'), the parameter α_t is set as follows:

$$\alpha_t = \frac{1}{2}\log\frac{1 - \epsilon_t}{\epsilon_t}$$

where the edge, ϵ_t is the sum of all the weights of misclassified examples on the t^{th} weak classifier, defined as

$$\epsilon_t = \sum_{n=1}^{N} d_n^{(t)} I(y_n \neq h_t(x_n))$$

2.2.2 ArcGv

In their paper, Freund and Schapire [4] derive a relationship between the margin and the test error. Their claim was that higher margins were responsible for better classification. In order to challenge this claim, Breiman developed the ArcGv ('Arcing-Game Value') algorithm, and showed that its minimum margin will always be greater than that of AdaBoost, while not necessarily achieving lower error rates [5].

In ArcGv, the parameter α_t is set as follows:

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} - \frac{1}{2}\log\frac{1+\varrho}{1-\varrho}$$

where ϱ is the minimum margin of the hypothesis, defined as

$$\varrho = \min_{n=1,\ldots,N} y_n f_{t-1}(x_n)$$

and where $f_i(x)$ is the normalized sum of the weighted hypotheses generated so far, defined as

$$f_i(x) = \frac{\sum_{t=1}^{i} \alpha_t h_t(x)}{\sum_{r=1}^{i} \alpha_r}$$

2.2.3 ArcGvMax

ArcGvMax is a reformulation of ArcGv developed by Meir [6], in which the minimum margin is set as follows:

$$\varrho = \max(\varrho_{t-1}, \min_{n=1,\ldots,N} y_n f_{t-1}(x_n))$$

2.2.4 Smooth Margin

A fourth method of boosting, developed by Rudin et. al. [7], defines a Smooth Margin function G as follows:

$$G = \frac{\sum_{i=1}^{N} \exp(-y_i \sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x_i))}{\sum_{j=1}^{t} \alpha_j}$$

Their method updates α_t as follows:

$$\alpha_t = \begin{cases} \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} - \frac{1}{2}\log\frac{1+G}{1-G} & \text{if } G > 0 \end{cases}$$

where the update rule is the same as in AdaBoost while G is not positive [1].

2.3 Classification Algorithms

We have implemented two different types of weak classifiers to use with our boosting implementation: perceptrons and decision trees.

2.3.1 Perceptrons

A linear perceptron generally maps a vector of real values x to an output $h_t(x)$ using parameters $\theta = \{w, b\}$ as follows [8, p. 192]:

$$f(x) = w \cdot x + b \tag{1}$$

where w is a vector of weights applied to each input value and b is the bias term. We adjusted the parameters iteratively using steepest descent. There are two approaches to the training algorithm: updating the parameters based on the error of each individual training case (online), or updating the parameters based on the error of the entire training set (offline). In general, while offline training is faster, online training converges more quickly. To test both within the boosting framework, it was important to incorporate the data distribution D into each training method.

Offline Since each training case within a batch contributes some small δ to the overall change in weights dW, we weighed the contribution $\delta(n)$ by the probability distribution of the data $D(n)$ in order to focus the training on the 'weak' cases.

Online Since training cases are processed iteratively during online training, a much more natural way to bias the training process toward the 'weak' cases was to sample cases from the training set using the weighted distribution D.

It is important to note that in order to make the comparison 'fair', when in practice the offline algorithm requires i iterations until convergence, we provide the online algorithm $i \cdot N$ samples for training, where N is the size of the dataset.

Additionally, we found that the boosted perceptron had a tendency to overfit training data, so we introduced a weight-decay regularizing term to the optimization.

2.3.2 Decision Trees

A decision tree maps a vector of features to a class label. Each internal node in the tree operates on a unique feature, and has children for every possible unique value of that feature. External nodes (leaves) in the tree correspond to the output of the classifier.

The tree is built during the training phase according to the decision tree algorithm in use. We chose to implement the the ID3 algorithm, which selects each successive feature to split on as that which maximizes the information gain [8, p. 663]:

$$IG(X|Y) = H(X) - H(X|Y)$$

where X is the feature with possible values $\{x_1, \ldots, x_n\}$, Y is the class, and H is the Shannon entropy defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

The standard ID3 information gain formula was modified to compute the entropy as a function of the data distribution d instead of the data frequency p:

$$H(X) = -\sum_{i=1}^{n} d(x_i) \log_2 d(x_i)$$

Classification is performed during the testing phase by following the branches on the tree according to the value of the corresponding feature of the example being classified, until a leaf node is reached.

In practice, we found that training variable-depth decision trees was too costly in terms of execution time. Instead, we chose to implement decision stumps, or decision trees with only one level. Although individually less accurate than full trees, an ensemble of stumps could perform equally well if not better. Furthermore, decision stumps would not be prone to the overfitting problem present in decision trees.

While experimenting with this configuration using the toy example, we found that the information gain formula tended to favour categorical features with large sets of possible values. Even an uncorrelated categorical feature could be selected to maximize the information gain if its set of possible values was large enough. Therefore, we introduced a normalizing divisor of $\log(2n+1)$ to be applied to each feature's information gain, where n is the number of unique values in the categorical feature.

As well, we found that on certain datasets successively rounds of boosted stumps would repeatedly split on only one feature in the dataset. We use the term 'feature blocking' to refer to the prevention of n successive stumps from deciding based on the same feature. As discussed in section 3, this modification reduces the error significantly for some datasets, while having little to no effect on others.

3 Experimental Results

In order to analyze how each algorithm performs under different conditions and with different datasets, we performed three different experiments. The first two compare the relative performance of weak classifiers and boosting algorithms, respectively. The third experiment tests the performance of boosting with a combination of two different types of classifiers.

3.1 Comparing Weak Classifiers

In order to isolate the performance of the various classification algorithms, all experiments in this section were performed using AdaBoost with 50 weak classifiers. (See Appendix A for error plots.)

Table 1: Final training and testing error by classifier and dataset

Weak Classifier	Dataset	Final Training Error (%)	Final Testing Error (%)	Test − Train (%)
Offline Perceptrons	Toy	9.1	9	-0.1
	DNA	20.6	23	2.4
	Adult	24.2	23.7	-0.5
	Mean	**18.0**	**18.6**	**0.6**
	Std. Dev.	**6.4**	**6.8**	**0.3**
Online Perceptrons	Toy	6.2	6.8	0.6
	DNA	1.7	7.9	6.2
	Adult	18.6	18.8	0.2
	Mean	**8.8**	**11.2**	**2.3**
	Std. Dev.	**7.1**	**5.4**	**-1.7**
Decision Stumps	Toy	10.2	9.9	-0.3
	DNA	5.3	11	5.7
	Adult	24.3	23.8	-0.5
	Mean	**13.3**	**14.9**	**1.6**
	Std. Dev.	**8.1**	**6.3**	**-1.7**
Decision Stumps with Feature Blocking	Toy	6.3	6.3	0
	DNA	5.4	10.8	5.4
	Adult	17	17.2	0.2
	Mean	**9.6**	**11.4**	**1.9**
	Std. Dev.	**5.3**	**4.5**	**-0.8**

Table 1 indicates the final testing and training errors of each classifier for each dataset, the mean and standard deviation of these values across all three datasets, and the difference between these values for training and testing. In all cases, lower values indicate better performance.

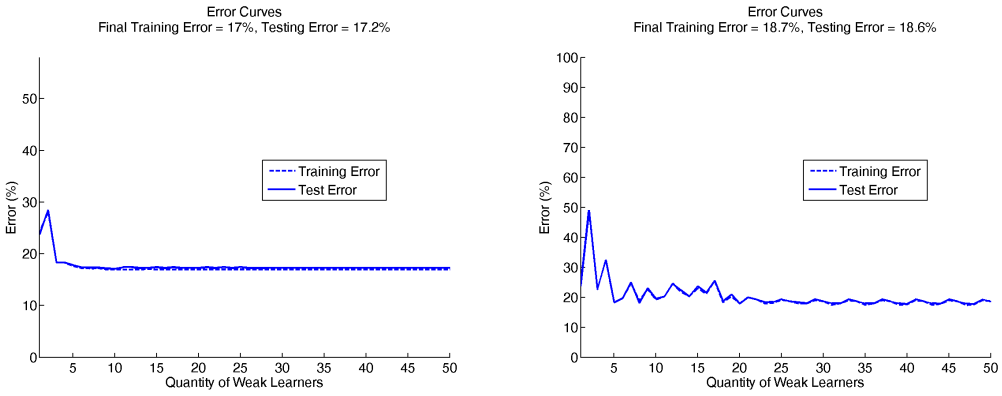
Offline perceptrons performed very poorly compared to the online variety, both in terms of convergence speed and final classification error. This may be partially due to the the inefficacy of the method applied to incorporate the data distribution D into the offline training method, however we believe that the online perceptron is generally more adaptive due to its iterative nature. Disregarding the slower training speed, the online algorithm seems better suited for boosting applications. It's important to note that both perceptrons required weight-decay regularization to be incorporated, to prevent significant overfitting on the training data.

Decision stumps performed comparably well, however the 'feature-blocking' modifier vastly improved performance on the Adult and Toy datasets, while having little effect on the DNA dataset. This is essentially a rudimentary implementation of using a 'memory' to influence the training of subsequent weak classifiers, and we are sure there is a more elegant solution, however it would be outside the scope of this project.

3.2 Comparing Boosting Algorithms

3.2.1 AdaBoost and ArcGv

In order to analyze the performance of the different boosting algorithms on real-world data, we ran a boosting cycle of 50 weak learners on the Adult and DNA datasets mentioned previously. In general, the lowest classification errors were achieved by AdaBoost.

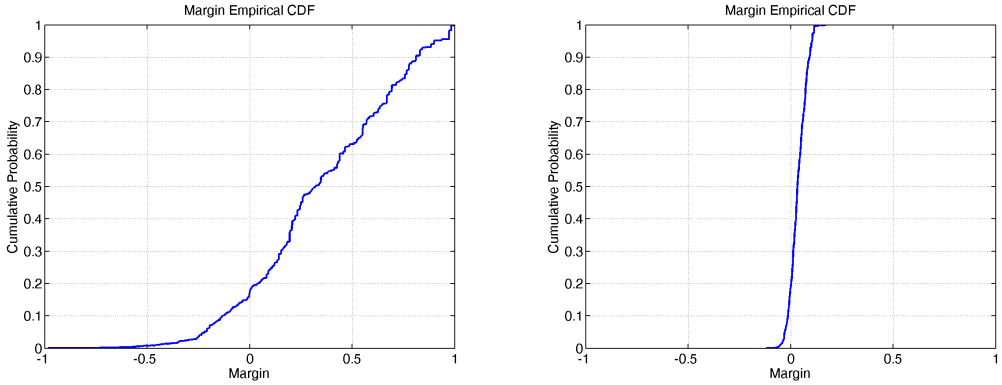


(a) Error curve for AdaBoost

(b) Error curve for ArcGv

Figure 1: Error curves for the Adult dataset, using Decision Stumps

For example, on the Adult dataset, using decision stumps as the weak learner, AdaBoost converges very quickly at 17.2% testing error. ArcGv on the other hand takes longer to converge, and never performs as well as AdaBoost. The real effect of ArcGv is noticeable in the final margin distibutions of the boosted classifiers:



(a) Margin CDF for AdaBoost

(b) Margin CDF for ArcGv

Figure 2: Margin CDF for the Adult dataset, using Decision Stumps

We can see that as intended by Breiman [5], ArcGv drives up the minimum margin of the classifier with each iteration, leading to ideally fewer negative margins. However the margins from ArcGv are smaller overall, meaning that the classifier fails to assign confidence to even a correct classification. Similar effect on the margin distribution was evident on the other datasets and weak classifiers (See Appendix B).

3.2.2 ArcGvMax and Smooth Margin

When running the ArcGvMax algorithm, we found that the modified *max* update rule for the minimum margin was rarely invoked, leading to indistinguishable performance from vanilla ArcGv. The smooth margin update rule was similarly never invoked, as the value of the smooth margin G never rose above 0.

3.3 Combination Method

As in the first experiment, in this section we use AdaBoost with 50 weak classifiers as to isolate the performance of the various classification algorithms. The layout of Table 2 is identical to that of Table 1, but in this case, we are comparing the best two classifiers from the first experiment to a combination method. Here, the online perceptron and the decision stump algorithms are alternated as weak learners to train one boosted classifier.

Table 2: Final training and testing error by classifier and dataset

Weak Classifier	**Dataset**	**Final Training Error (%)**	**Final Testing Error (%)**	**Test − Train (%)**
Online Perceptrons	Toy	6.2	6.8	0.6
	DNA	1.7	7.9	6.2
	Adult	18.6	18.8	0.2
	Mean	**8.8**	**11.2**	**2.3**
	Std. Dev.	**7.1**	**5.4**	**-1.7**
Decision Stumps with Feature Blocking	Toy	6.3	6.3	0
	DNA	5.4	10.8	5.4
	Adult	17	17.2	0.2
	Mean	**9.6**	**11.4**	**1.9**
	Std. Dev.	**5.3**	**4.5**	**-0.8**
Combination	Toy	6.3	6.3	0
	DNA	2	7	5
	Adult	18.4	18.6	0.2
	Mean	**8.9**	**10.6**	**1.7**
	Std. Dev.	**6.9**	**5.6**	**-1.3**

The results of this approach are mixed (see also Appendix C). The Toy dataset shows little improvement overall, while the Adult dataset combination shows errors somewhere in-between its constituent classifiers. The only real improvement is shown on the DNA dataset, where the combination method shows a generalization error below either the pure perceptron or decision stump algorithms.

We were hoping for more dramatic results, wherein a variety of classifier algorithms would outperform one unique algorithm in a boosting framework. Ideally, we considered that an inherently different type of classification could be better at compensating for the 'weak' spots of each previous learner during the boosting iterations.

4 Conclusions

In this paper, we considered four different boosting algorithms: AdaBoost, ArcGv, ArcGvMax, and Smooth margin. We implemented these algorithms using two families of weak classifiers, Perceptrons (Offline and Online) and Decision Stumps (with and without Feature Blocking), and analyzed their performance on three different datasets.

We found that in general, Online Perceptrons performed better than Offline Perceptrons, and Decision Stumps performed better with Feature Blocking than without. We also found that alternating between Online Perceptrons and Feature Blocking Decision Stumps had mixed results, depending on the dataset. The generalization performance of Decision Stumps was similar to that of Perceptrons; however we found that Stumps were much quicker to train, and less prone to overfit the data.

We confirmed Breiman's intended behaviour of ArcGv over AdaBoost, with AdaBoost resulting in lower testing errors in general, even while ArcGv maximized its margins more aggressively. Meanwhile, the performance of the ArcGvMax and Smooth Margin algorithms rarely deviated that of ArcGv and AdaBoost, respectively.

Our proposed techniques of alternating between classifiers, as well as blocking successive features for decision stumps, both seem promising in our preliminary results, but require further study.

References

[1] L. Reyzin, "Analyzing margins in boosting," 2005.

[2] R. E. Schapire, "The Boosting Approach to Machine Learning: An Overview." Nonlinear Estimation and Classification, Berkeley, CA, USA, 2001.

[3] A. Frank and A. Asuncion, "UCI machine learning repository," 2010.

[4] Y. Freund and R. E. Schapire, "A short introduction to boosting," in *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1401–1406, Morgan Kaufmann, 1999.

[5] L. Breiman, "Arcing Classifiers," *The Annals of Statistics*, vol. 26, no. 3, pp. 801–824, 1998.

[6] R. Meir and G. Rätsch, *An introduction to boosting and leveraging*, pp. 118–183. New York, NY, USA: Springer-Verlag New York, Inc., 2003.

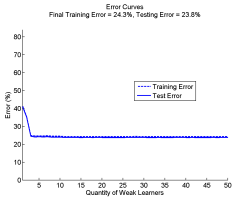
[7] C. Rudin, R. E. Schapire, and I. Daubechies, "Boosting based on a smooth margin," in *Learning Theory, 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004, Proceedings* (J. Shawe-Taylor and Y. Singer, eds.), vol. 3120 of *Lecture Notes in Computer Science*, pp. 502–517, Springer, 2004.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

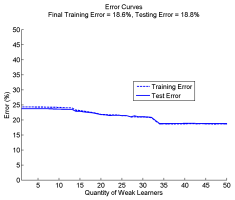
Appendices

Appendix A

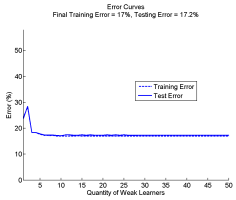
Error plots from test 1



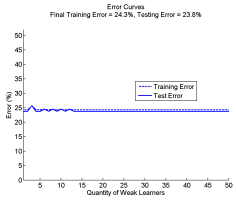
(a) Error curve for AdaBoost with Offline Perceptrons



(b) Error curve for AdaBoost with Online Perceptrons

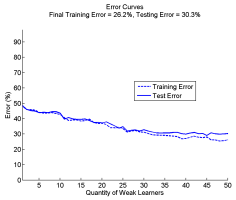


(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

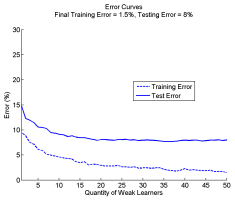


(d) Error curve for AdaBoost with Decision Stumps

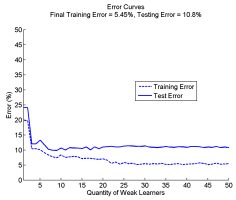
Figure 3: Error and Margin plots for AdaBoost on Adult dataset



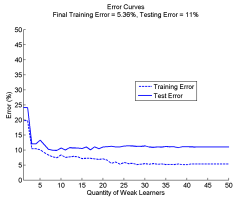
(a) Error curve for AdaBoost with Offline Perceptrons



(b) Error curve for AdaBoost with Online Perceptrons

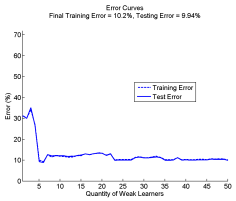


(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

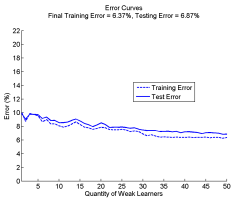


(d) Error curve for AdaBoost with Decision Stumps

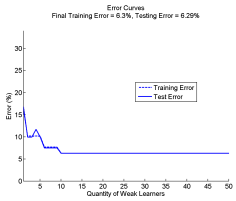
Figure 4: Error and Margin plots for AdaBoost on DNA dataset



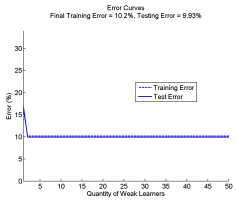
(a) Error curve for AdaBoost with Offline Perceptrons



(b) Error curve for AdaBoost with Online Perceptrons



(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

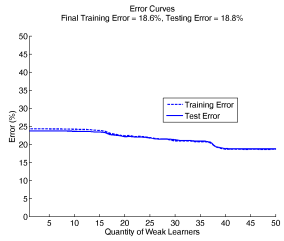


(d) Error curve for AdaBoost with Decision Stumps

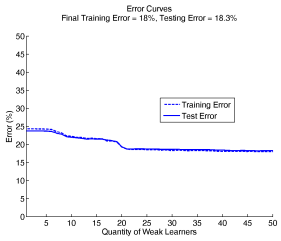
Figure 5: Error and Margin plots for AdaBoost on Toy dataset

Appendix B

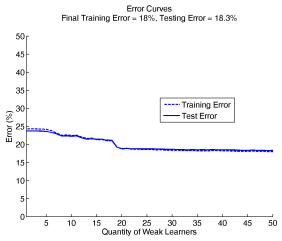
A selection of error plots and final margin plots from test 2



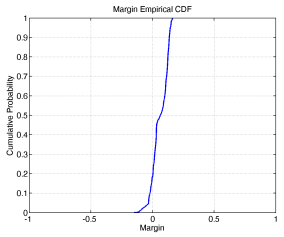
(a) Error curve for AdaBoost



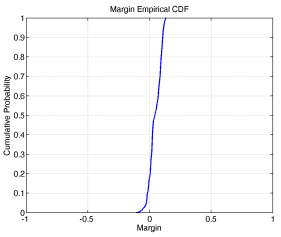
(b) Error curve for ArcGv



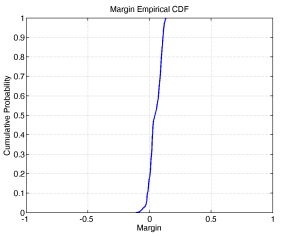
(c) Error curve for ArcGvMax



(d) Margins for AdaBoost

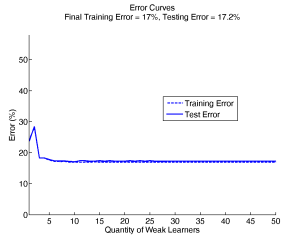


(e) Margins for ArcGv

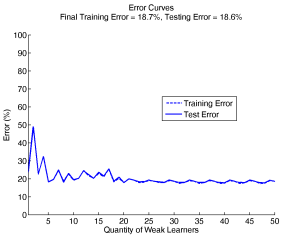


(f) Margins for ArcGvMax

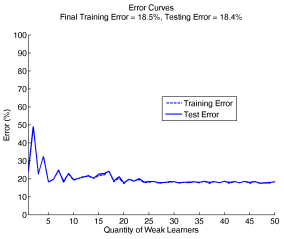
Figure 6: Error and Margin plots for Online Perceptrons on Adult dataset



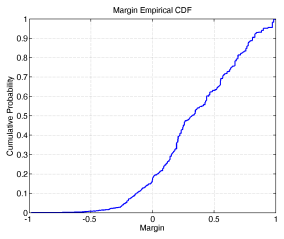
(a) Error curve for AdaBoost



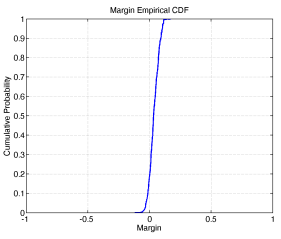
(b) Error curve for ArcGv



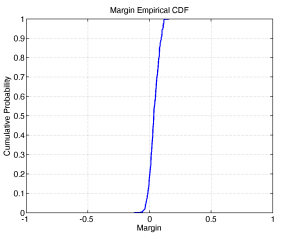
(c) Error curve for ArcGvMax



(d) Margins for AdaBoost



(e) Margins for ArcGv

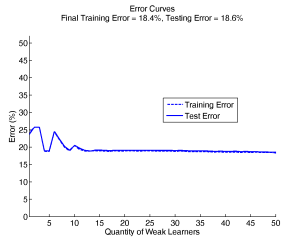


(f) Margins for ArcGvMax

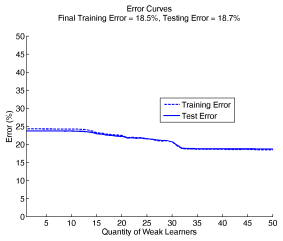
Figure 7: Error and Margin plots for Feature Blocking Decision Stumps on Adult dataset

Appendix C

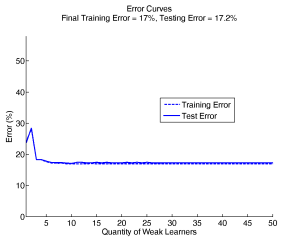
Error plots for test 3



(a) Error curve for AdaBoost with Mixture of Classifiers

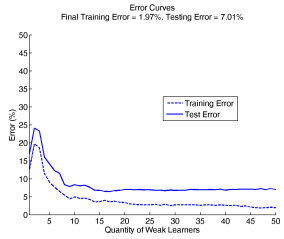


(b) Error curve for AdaBoost with Online Perceptrons

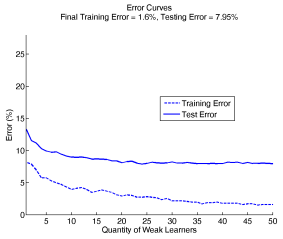


(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

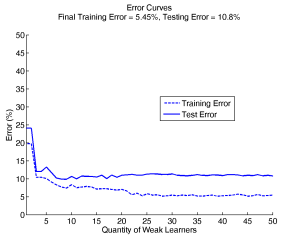
Figure 8: Error and Margin plots for AdaBoost on Adult dataset



(a) Error curve for AdaBoost with Mixture of Classifiers

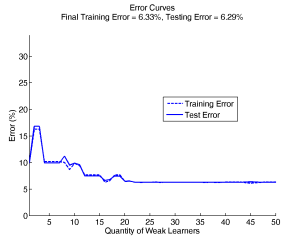


(b) Error curve for AdaBoost with Online Perceptrons

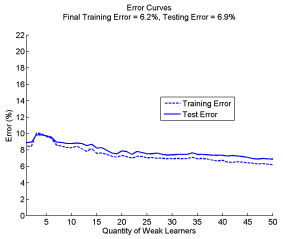


(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

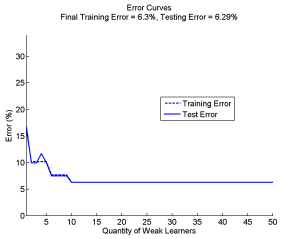
Figure 9: Error and Margin plots for AdaBoost on DNA dataset



(a) Error curve for AdaBoost with Mixture of Classifiers



(b) Error curve for AdaBoost with Online Perceptrons



(c) Error curve for AdaBoost with Feature Blocking Decision Stumps

Figure 10: Error and Margin plots for AdaBoost on Toy dataset