CGBoost: Conjugate Gradient in Function Space

Ling Li **Yaser S. Abu-Mostafa** **Amrit Pratap**
Learning Systems Group, California Institute of Technology,
Pasadena, CA 91125, USA
`{ling,yaser,amrit}@caltech.edu`

Abstract

The superior out-of-sample performance of AdaBoost has been attributed to the fact that it minimizes a cost function based on margin, in that it can be viewed as a special case of AnyBoost, an abstract gradient descent algorithm. In this paper, we provide a more sophisticated abstract boosting algorithm, CGBoost, based on conjugate gradient in function space. When the AdaBoost exponential cost function is optimized, CGBoost generally yields much lower cost and training error but higher test error, which implies that the exponential cost is vulnerable to overfitting. With the optimization power of CGBoost, we can adopt more "regularized" cost functions that have better out-of-sample performance but are difficult to optimize. Our experiments demonstrate that CGBoost generally outperforms AnyBoost in cost reduction. With suitable cost functions, CGBoost can have better out-of-sample performance.

1 Introduction

AdaBoost [4] is probably the most popular algorithm among the boosting family which generates a linear combination of weak hypotheses. Given a weak learner $\mathcal{L}$, AdaBoost iteratively adds hypotheses generated by $\mathcal{L}$ to the linear combination. It emphasizes difficult examples by giving them higher sample weights and favors hypotheses with lower training errors by giving them larger coefficients. AdaBoost can be viewed as a special case of AnyBoost [7], a general gradient descent in function space.

It has been observed experimentally that AdaBoost keeps improving the out-of-sample error even after the training error of the linear combination has reached zero [2]. One explanation to this is that AdaBoost improves the margins of the training examples even after all the examples have positive margins, and larger margins imply better out-of-sample performance [9]. However, this explanation was challenged in [5] where the algorithms achieve larger minimum margins than AdaBoost, but do not have better out-of-sample performance than AdaBoost, mostly worse. Another related explanation is that AdaBoost optimizes a cost function based on example margins [7]. Although there is a theoretical bound on the out-of-sample error based on cost, it is still unclear whether minimizing the cost is helpful in practice.

We take a closer look at this question, examining how the cost function, in and of itself,

Caltech Computer Science Technical Report CaltechCSTR:2003.007, Aug. 2003
`http://resolver.caltech.edu/CaltechCSTR:2003.007`

affects the out-of-sample performance. To do so, we apply more sophisticated optimization techniques directly to the cost function. We obtain three sets of results:

1. The introduction of a new abstract boosting algorithm, CGBoost, based on conjugate gradient in function space which has better cost optimization performance than AnyBoost.
2. The conclusion that AdaBoost cost function is much more vulnerable to overfitting when it is directly minimized instead of being minimized within the confines of the AdaBoost algorithm.
3. The identification of more "regularized" cost functions whose direct minimization results in a better out-of-sample performance than that of the AdaBoost cost function.

The paper is organized as follows. The CGBoost algorithm and its implementation with the margin cost functions are introduced in Section 2. In Section 3, we compare CGBoost and AnyBoost with two different cost functions. One cost function is observed to have better out-of-sample performance but is more difficult to optimize. CGBoost has superior performance than AnyBoost with that cost function. We then give results on some UCI data sets in Section 4.

2 CGBoost

We assume the examples $(\mathbf{x}, y)$ are randomly generated according to some unknown probability distribution on $X \times Y$ where X is the input space and Y is the output space. Since this paper focuses on voted combinations of binary classifiers, $Y = \{-1, 1\}$.

The voted combination is $\operatorname{sign}(F(\mathbf{x}))$ where

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

with weak hypotheses $f_t\colon X \to Y$ from a base learning model $\mathcal{G}$, and hypothesis coefficients $\alpha_t \in \mathbb{R}^+$. Let $\operatorname{lin}(\mathcal{G})$ denote the set of all linear combinations (with nonnegative coefficients) of functions in $\mathcal{G}$, and $C\colon \operatorname{lin}(\mathcal{G}) \to \mathbb{R}^+$ be a cost function. We want to construct a combination $F \in \operatorname{lin}(\mathcal{G})$ to minimize $C(F)$.

2.1 AnyBoost: Gradient Descent

AnyBoost [7] is an abstract boosting algorithm that provides a general framework for minimizing C iteratively via gradient descent in function space.

Suppose we have a function $F \in \operatorname{lin}(\mathcal{G})$ and we wish to find a "direction" $f \in \mathcal{G}$ so that the cost $C(F + \epsilon f)$ decreases for some small positive ϵ. The desired direction such that the cost decreases most rapidly (for small ϵ) is the negative functional gradient $-\nabla C(F)$, where

$$\nabla C(F)(\mathbf{x}) = \left. \frac{\partial C(F + \tau 1_{\mathbf{x}})}{\partial \tau} \right|_{\tau=0},$$

where $1_{\mathbf{x}}$ is the indicator function of $\mathbf{x}$. In general, it may not be possible to choose $f = -\nabla C(F)$ since f has to be one of the hypotheses in $\mathcal{G}$. So, instead, AnyBoost searches for f that maximizes $\langle -\nabla C(F), f \rangle$.[1] After f is fixed, a line search can be used to determine

[1]The inner product $\langle \cdot, \cdot \rangle$ is define on $\operatorname{lin}(\mathcal{G})$. Generally, we want $f \in \mathcal{G}$ to maximize the *normalized* inner product $\langle -\nabla C(F), f \rangle / \sqrt{\langle f, f \rangle}$. For the inner product definition (4) used in this paper and some other papers [7], $\langle f, f \rangle$ is a constant for binary classifiers. So there is no need for normalization.

the coefficient of f in the new combination of hypotheses.

2.2 CGBoost: Conjugate Gradient

If we replace gradient descent in AnyBoost with the more efficient conjugate gradient technique [8, §12.4], we obtain a new and more powerful abstract boosting algorithm: CGBoost (Algorithm 1). The main difference between conjugate gradient and gradient descent is that the former also utilizes the second-order information of the cost to adjust search directions so that the cost could be decreased faster.

Let d_t denote the search direction at iteration t, and $f_t \in \mathcal{G}$ denote the weak hypothesis approximating the negative functional gradient $-\nabla C(F)$. Instead of letting $d_t = f_t$ directly, we choose the search direction to be

$$d_t = f_t + \beta_t d_{t-1}, \tag{1}$$

where $\beta_t \in \mathbb{R}$ and d_{t-1} is the direction from last iteration. With this change, the search direction d_t is no longer limited to a single hypothesis in $\mathcal{G}$. Instead, it is some linear combination of the current and previous f_t's and thus $d_t \in \operatorname{lin}(\mathcal{G})$.

The β_t in equation (1) determines how much the previous search direction d_{t-1} affects the current direction d_t. If $\beta_t = 0$, d_t is solely determined by the current gradient f_t, which usually helps conjugate gradient recover from some bad situations [8, pp. 408]. In Algorithm 1, β_1 is effectively forced to be 0 since d_0 is initialized to 0. For reasons that will be explained in Section 3, β_t is also clipped to 0 for the first several iterations. For other cases, we can use the Polak-Ribiére formula [8, pp. 399]

$$\beta_t = \frac{\langle f_t, f_t - f_{t-1}\rangle}{\langle f_{t-1}, f_{t-1}\rangle} \tag{2}$$

which automates the "restart" mechanism.

Although the search direction of CGBoost is more complicated than that of AnyBoost, the combination F_T is still a linear combination of (at most) T weak hypotheses in $\mathcal{G}$, since all d_t are in the space spanned by $\{f_1, \ldots, f_T\}$. For $i \le t$, define

$$\beta_{i,t} = \begin{cases} \prod_{j=i+1}^{t} \beta_j, & \text{if } i < t; \\ 1, & \text{if } i = t. \end{cases}$$

Algorithm 1 CGBoost: Conjugate gradient in function space.

Require:

- A base learning model $\mathcal{G}$ and an inner product defined on $\operatorname{lin}(\mathcal{G})$.
- A differentiable cost function $C\colon \operatorname{lin}(\mathcal{G}) \to \mathbb{R}^+$.
- A weak learner $\mathcal{L}(F)$ that accepts $F \in \operatorname{lin}(\mathcal{G})$ and returns $f \in \mathcal{G}$ with a large value of $\langle -\nabla C(F), f\rangle$.

```
 1: $F_0 \leftarrow 0$, $d_0 \leftarrow 0$
 2: for $t = 1$ to $T$ do
 3:     $f_t \leftarrow \mathcal{L}(F_{t-1})$
 4:     $d_t \leftarrow f_t + \beta_t d_{t-1}$ for some $\beta_t \in \mathbb{R}$
 5:     if $\langle -\nabla C(F_{t-1}), d_t\rangle \le 0$ then
 6:         return $F_{t-1}$
 7:     end if
 8:     $F_t \leftarrow F_{t-1} + \alpha_t d_t$ for some $\alpha_t > 0$
 9: end for
10: return $F_T$
```

We then have $d_t = \sum_{i=1}^{t} \beta_{i,t} f_i$, and

$$F_T = \sum_{t=1}^{T} \alpha_t d_t = \sum_{i=1}^{T} \left(\sum_{t=i}^{T} \alpha_t \beta_{i,t} \right) f_i. \tag{3}$$

If the base learning model $\mathcal{G}$ is negation closed, which is trivially true for almost all reasonable binary classification models, it is obvious that $F_T \in \operatorname{lin}(\mathcal{G})$. In the following subsection, we will see a definition for the inner product that also guarantees the coefficients in (3) are nonnegative. If it is the size of linear combinations that generally decides the complexity of the combination, these features imply that using conjugate gradient will not increase the complexity.

The search step α_t can be determined by some line search technique. If α_{t-1} ensures that $\nabla C(F_{t-1}) \perp d_{t-1}$ (which could be achieved by an *exact* line search), we have

$$\langle -\nabla C(F_{t-1}), d_t \rangle = \langle -\nabla C(F_{t-1}), f_t \rangle .$$

That is, the adjusted direction d_t is just as close to $-\nabla C(F_{t-1})$ as f_t is, while f_t is guaranteed by the weak learner $\mathcal{L}$ to be a good approximation of the negative gradient.

2.3 CGBoost with Margin Cost Functions

Commonly used cost functions are usually defined on example margins. Given a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ of N examples, the margin cost of F has the form

$$C(F) = \frac{1}{N} \sum_{i=1}^{N} c(y_i F(\mathbf{x}_i)),$$

where $y_i F(\mathbf{x}_i)$ is the margin of example $(\mathbf{x}_i, y_i)$ and $c\colon \mathbb{R} \to \mathbb{R}^+$ is a (decreasing) function of the margin. We may thus use $c(\cdot)$ to refer the whole cost function C.

The inner product between hypotheses f and g is defined as

$$\langle f, g \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) g(\mathbf{x}_i). \tag{4}$$

In this case,

$$\langle -\nabla C(F), f \rangle = \frac{1}{N^2} \sum_{i=1}^{N} y_i f(\mathbf{x}_i) \cdot \left[-c'(y_i F(\mathbf{x}_i)) \right] .$$

Maximizing $\langle -\nabla C(F), f \rangle$ is thus equivalent to minimizing the training error with sample weight $D(i) \propto -c'(y_i F(\mathbf{x}_i))$. This explains why a weak learner $\mathcal{L}$ is used in Algorithm 1 to return f that has a large value of $\langle -\nabla C(F), f \rangle$.

For a binary classifier f, the definition in (4) gives $\langle f, f \rangle \equiv 1$. Then equation (2) becomes

$$\beta_t = 1 - \langle f_t, f_{t-1} \rangle$$

which is always nonnegative.

Algorithm 2 summarizes the implementation of CGBoost with margin cost functions.

3 Cost Functions

The frameworks of CGBoost and AnyBoost leave the choice of cost functions open to users. However, not all the cost functions are suitable for learning purposes. We will discuss two cost functions in this section as well as the performance of CGBoost and AnyBoost on these cost functions.

Algorithm 2 CGBoost with margin cost functions

```
Require:
    • A base learning model $\mathcal{G}$ containing hypotheses $f\colon X \to \{-1,1\}$.
    • A differentiable cost function $c\colon \mathbb{R} \to \mathbb{R}^+$.
    • A weak learner $\mathcal{L}(S,D)$ that accepts a training set $S$ and a sample distribution
      $D$ and returns $f \in \mathcal{G}$ with small weighted error $\sum_i D(i)\,[f(\mathbf{x}_i) \neq y_i]$.
 1: $F_0 \leftarrow 0$, $d_0 \leftarrow 0$
 2: for $t = 1$ to $T$ do
 3:   $D_t(i) \leftarrow c'(y_i F_{t-1}(\mathbf{x}_i)) / \sum_{j=1}^N c'(y_j F_{t-1}(\mathbf{x}_j))$ for $i = 1,\ldots,N$
 4:   $f_t \leftarrow \mathcal{L}(S, D_t)$
 5:   $d_t \leftarrow f_t + \beta_t d_{t-1}$ for $\beta_t = 1 - \langle f_{t-1}, f_t \rangle$
 6:   if $\sum_{i=1}^N D_t(i) y_i d_t(\mathbf{x}_i) \leq 0$ then
 7:     return $F_{t-1}$
 8:   end if
 9:   $F_t \leftarrow F_{t-1} + \alpha_t d_t$ for some $\alpha_t > 0$
10: end for
11: return $F_T$
```

3.1 AdaBoost Exponential Cost

When the margin cost function $c(\rho) = e^{-\rho}$ is used, AnyBoost is equivalent to AdaBoost [7]. To have a taste of the performance of CGBoost, we compare CGBoost using this cost function with AdaBoost. Since the same cost function is used, this is a comparison between two optimization methods.

The data set used in this comparison was generated by the Caltech Data Engine[2]. The input space is $X = \mathbb{R}^8$ and output space is $Y = \{-1,1\}$. We use 400 examples for training and 3000 examples for testing. Our learning model $\mathcal{G}$ contains decision stumps and the weak learner $\mathcal{L}$ returns the decision stump with best weighted training error.

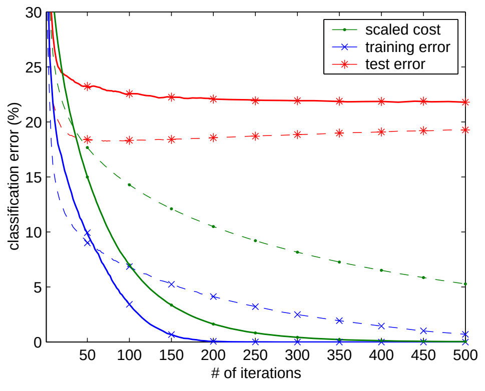
The results averaged over 132 independent trials are shown in Figure 1(a). We can see that, though the cost from AdaBoost was lower on average during the first 20–30 iterations, CGBoost overall decreased the cost faster and achieved a significantly lower cost. The training error, with similar trend as the cost, was also decreased much faster by CGBoost.

However, the out-of-sample behavior was the opposite. Noticing that AdaBoost got overfitting after roughly 50 iterations, the deteriorating out-of-sample performance of CGBoost implies that the exponential cost function is more vulnerable to overfitting when optimized directly and more aggressively.

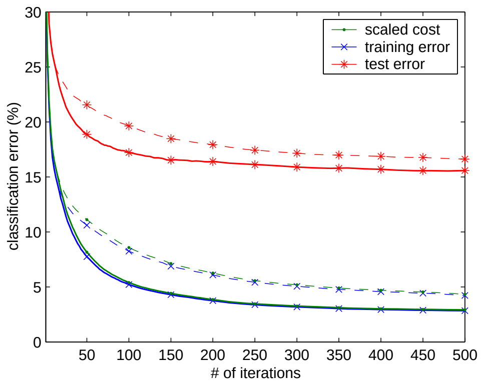
This result is of no surprise since the exponential cost function has a very steep curve for negative margins (see Figure 2) and thus emphasizes “difficult” examples too much [3, 5, 6]. While better optimization techniques can help decreasing the cost faster, the out-of-sample performance would be mainly determined by the cost function itself.

Based on the observation of this comparison, we set $\beta_t = 0$ for the first several iterations in the following experiments to take advantage of the initial efficiency of gradient descent.

[2]The Caltech Data Engine [1] 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) parameters 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.



(a) Exponential cost



(b) Bisigmoid cost

Figure 1: Performance of CGBoost (solid curves) and AnyBoost (dashed curves) with two different cost functions. (a) AnyBoost with exponential cost is equivalent to AdaBoost; (b) Bisigmoid cost with $\kappa_+ = 1$ and $\kappa_- = 1.05$ in (5). Since $[\rho < 0] \approx \frac{1}{2}c(\rho)$ when $|\rho|$ is large, the training error and cost (scaled by 50) coincide quite well.

3.2 Bisigmoid Cost

Because of the power of conjugate gradient as an optimization technique, one can afford to use more "regularized" cost functions that are harder to optimize but have better out-of-sample performance.

The sigmoid margin cost function $c(\rho) = 1 - \tanh(\rho)$ was suggested in [7]. Since it has a flatter part for negative margins compared to the exponential cost (see Figure 2), it does not punish outliers too much. However, this cost function causes difficulties to AnyBoost and CGBoost. If the weak learner $\mathcal{L}$ finds the optimal hypothesis for the uniform sample distribution, AnyBoost and CGBoost will terminate at the second iteration due to $c'(-\rho) = c'(\rho)$ (Proof is similar to [7, Lemma 12.9]).

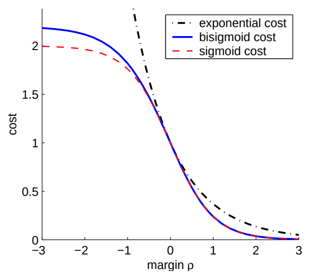


Figure 2: Three margin cost functions.

A simple technique to avoid $c'(-\rho) = c'(\rho)$ is to concatenate sigmoid functions with different slopes for negative margins and positive margins. Below is what we call the *bisigmoid* function:

$$c(\rho) = \begin{cases} \kappa_+ - \kappa_+ \tanh(\rho/\kappa_+), & \text{for } \rho > 0; \\ \kappa_+ - \kappa_- \tanh(\rho/\kappa_-), & \text{otherwise.} \end{cases} \tag{5}$$

where κ_+ and κ_- are positive numbers controlling the slopes for positive and negative margins, respectively. We usually set $\kappa_- > \kappa_+$ so that the negative margins could also be emphasized a bit more. The closer κ_- is to κ_+, the more similar the bisigmoid curve is to a scaled sigmoid.

We applied CGBoost and AnyBoost with the bisigmoid function ($\kappa_+ = 1$ and $\kappa_- = 1.05$) to the problem in Subsection 3.1. Again we observed in Figure 1(b) that CGBoost optimized the cost and training error faster, though this time with the bisigmoid function, the cost could not be reduced to near zero and the training error was also above zero. However, we observed much better out-of-sample performance of both CGBoost and AnyBoost, compared to test errors in Figure 1(a). On average, *CGBoost achieved the lowest test error*.

This result reinforces the idea that the cost functions have great impact on the out-of-sample performance, while the optimization techniques only help to a lesser extent.

4 Experimental Results

We compared CGBoost and AnyBoost on six UCI data sets[3] with the exponential cost function and the bisigmoid cost function. κ_+ is fixed to 1 in all these experiments. Since the value of κ_- decides how flat the bisigmoid is and thus how difficult the optimization is, we tried four values of κ_-, namely, 1.05, 1.1, 1.15, and 1.2. We observed that the smaller κ_- is, the more difficult the optimization is.

Each data set was randomly partitioned so that 80%, 10%, and 10% of the examples were used for training, validation, and testing. CGBoost and AnyBoost with different cost functions were allowed to run 300 iterations. Results were averaged over more than 60 trials.

Table 1 gives the geometric mean of the cost ratios between two optimization algorithms, CGBoost and AnyBoost, at the final iteration. As we expected, the cost of CGBoost is generally much lower than that of AnyBoost.

Cost Function	**pima**	**sonar**	**clevel**	**vote84**	**cancer**	**iono**
exponential	0.5716	0.0000	0.0675	0.1006	0.0656	0.0000
$\kappa_- = 1.05$	0.8067	0.2674	0.6882	0.4997	0.8896	0.9949
$\kappa_- = 1.2$	0.7615	0.0000	0.6374	0.8058	0.9011	0.0000

Table 1: The average final cost ratio of CGBoost to AnyBoost. Numbers less than 0.00005 are shown as 0.0000. To save space, the ratios with $\kappa_- = 1.1$ and $\kappa_- = 1.15$ are omitted.

We also compared the out-of-sample performance of these two algorithms. During one trial, the linear combination with the best validation error was picked. That is, for the exponential cost, validation chose the size of boosting; for the bisigmoid cost, validation also chose the "optimal" κ_- value.

The average test errors are listed in Table 2. Though it seems that CGBoost did not yield better test errors, the results from these algorithms are similar, and the relatively high error bars prevent us from drawing statistically significant conclusions for these limited data sets.

Cost	**Method**	**pima**	**sonar**	**clevel**	**vote84**	**cancer**	**iono**
exp.	AnyBoost	**25.10%**	**19.74%**	16.56%	**4.38%**	5.38%	**11.42%**
exp.	CGBoost	25.72%	22.02%	17.83%	4.46%	4.99%	12.62%
bisigmoid	AnyBoost	25.83%	23.97%	**16.45%**	4.59%	**4.14%**	11.49%
bisigmoid	CGBoost	26.25%	24.07%	17.77%	4.67%	4.78%	11.77%
roughly error bar		4.85%	9.22%	7.45%	3.04%	2.60%	5.59%

Table 2: Average test errors of CGBoost and AnyBoost on six UCI data sets. The lowest error in each column is in bold font. The error bars are much higher than the differences.

[3]The six UCI data sets are pima-indians-diabetes, sonar, cleveland-heart-disease, voting-records, breast-cancer-wisconsin, and ionosphere. Examples with missing features are removed from the original data sets.

5 Conclusions

AdaBoost can be viewed as gradient descent of the exponential cost function in function space. In this paper, we introduced a new boosting algorithm, CGBoost, based on conjugate gradient in function space. We demonstrated with Caltech Data Engine data and UCI data that CGBoost generally optimized the cost faster and achieved much lower training error.

We also observed that the exponential cost of AdaBoost was much more vulnerable to overfitting when it was minimized by the more aggressive CGBoost. The bisigmoid cost function which has a flatter part for negative margins was introduced to alleviate the overfitting problem. It also avoids the optimization difficulties that normal sigmoid function might have. Our experiments showed that, though it is harder to optimize, it generally leads to better out-of-sample performance. CGBoost with the bisigmoid cost yielded the lowest test error with Data Engine data.

However, the impact of cost functions on the out-of-sample performance still remains unclear, partly due to the statistically insignificant results on the UCI data sets.

Acknowledgments

This work was supported by the Caltech Center for Neuromorphic Systems Engineering under NSF Cooperative Agreement EEC-9402726.

References

[1] Data engine for machine learning research. Technical report, California Institute of Technology. In preparation. `http://www.work.caltech.edu/dengin/`.

[2] L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Department of Statistics, University of California at Berkeley, Apr. 1996.

[3] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, Aug. 2000.

[4] 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.

[5] A. J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699. AAAI Press / MIT Press, 1998.

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

[7] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, chapter 12, pages 221–246. MIT Press, Cambridge, MA, Oct. 2000.

[8] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill series in industrial engineering and management science. The McGraw-Hill Companies, Inc, New York, 1996.

[9] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, Oct. 1998.