Minimum Majority Classification and Boosting

Philip M. Long
Genome Institute of Singapore
1 Science Park Road
The Capricorn, #05-01
Singapore 117528, Republic of Singapore
gislongp@nus.edu.sg

Abstract

Motivated by a theoretical analysis of the generalization of boosting, we examine learning algorithms that work by trying to fit data using a simple majority vote over a small number of a collection of hypotheses. We provide experimental evidence that an algorithm based on this principle outputs hypotheses that often generalize nearly as well as those output by boosting, and sometimes better. We also provide experimental evidence for an additional reason that boosting algorithms generalize well, that they take advantage of cases in which there are many simple hypotheses with independent errors.

Introduction

Boosting algorithms (Schapire 1990; Freund 1995; Freund & Schapire 1997) work by repeatedly applying a subalgorithm to a dataset. Before each application, the examples are reweighted. The hypotheses returned are then combined by voting: for example, in the two-class case, each hypothesis is assigned a weight, and an item is classified as 1 if the total weight of the hypotheses classifying it as 1 is more than that of hypotheses classifying it as 0. Boosting has been successfully applied in a variety of domains (Drucker, Schapire, & Simard 1993; Freund & Schapire 1996a; Drucker & Cortes 1996; Abney, Schapire, & Singer 1999; Cohen, Schapire, & Singer 1999; Freund *et al.* 1998; Bauer & Kohavi 1999; Iyer *et al.* 2000; Schapire, Singer, & Singhal 1998).

The generalization observed by boosting algorithms appeared to run counter to the Occam's Razor principle which has been the bedrock of much of both theoretical and applied machine learning research. Loosely, Occam's Razor says that simpler hypotheses are to be preferred, because simple hypotheses are less apt to perform well on the training data by chance. But the hypotheses output by boosting algorithms are more complex than those output by the subalgorithm, and generalization is seen to improve as the number of rounds of boosting increases, even after all of the training data is classified correctly (Drucker & Cortes 1996; Quinlan 1996; Breiman 1998).

An influential theoretical analysis (Schapire *et al.* 1998) bounded the generalization error of the hypothesis output by the boosting algorithm in terms of the *margin*.[1] This is the minimum, over all examples, of how much greater is the total weight of the hypotheses classifying the example correctly than the total weight of the incorrect hypotheses. (Grove and Schuurmans (1998) evaluated an algorithm that used linear programming to train weights to maximize the margin, and found that the resulting performance was often somewhat worse than obtained by Adaboost.)

The proof of the generalization bound in terms of the margin can be crudely paraphrased as follows: hypotheses that classify data correctly with a large margin generalize well because they can be accurately approximated by simple majority votes over few of the constituent hypotheses, which is a "simple" class of hypotheses. Continuing boosting for more rounds improves generalization even after the voting hypothesis has zero training error because, the resulting explanation goes, while the hypotheses generated are themselves more complex, they can be accurately approximated by simple hypotheses to an improved degree.

This observation motivates the following question. What if a learning algorithm worked by *directly* trying to find small collections of simple hypotheses for which a majority vote correctly classified much of the data? This is the subject of this paper. Our goals are both to better understand the reason that boosting algorithms generalize well, and to provide an approximation to boosting that outputs simpler hypotheses. (This problem was posed by Quinlan (1999) .)

The design of an algorithm applying this minimum majority principle gives rise to a nontrivial optimization problem. The author (Long 2001) previously proposed a polynomial-time algorithm for trying to minimize the size of a collection of hypotheses for which a simple vote yields correct classifications on all members of a dataset. The algorithm has a theoretical approximation bound, but it needed to be modified substantially to work well in practice. It makes use of a nonstandard variant (Pach & Agarwal 1995; Williamson 1999) of *randomized rounding* (Raghavan & Thompson 1987), a technique for solving integer programming problems in which, roughly

- the requirement that the variables are integers is removed,

[1] In fact, their analysis was more general, but the spirit of the analysis is captured in this simple case, and the below discussion also applies to their more general analysis.

- the resulting solution is used to generate a probability distribution over integer solutions, and
- an integer solution is sampled from this distribution.

Using this algorithm, we carried out a suite of experiments comparing its generalization with that obtained through boosting. As was done in some of the experiments in (Freund & Schapire 1996a), both algorithms used decision stumps (Iba & Langley 1992) as the hypotheses of the subalgorithm. (Decision stumps test the value of a single attribute, comparing it with a threshold if it is numerical, and base their classification on the result. Examples include "classify as 1 exactly when $x_3 \geq 2.2$" and "classify as 0 exactly when $x_7 =$ RED".) We compared them on a variety of datasets previously used for evaluating boosting (Freund & Schapire 1996a). In almost all cases, the minimum majority algorithm accurately approximated the generalization of boosting; sometimes it performed slightly better. However, in a few cases, it performed significantly worse.

Why? One possible explanation is as follows. The reweighting of the examples done by boosting tends to force the subalgorithm to return hypotheses whose errors are approximately independent (Ali & Pazzani 1996; Niyogi, Pierrot, & Siohan 2001), that is for example

$$\begin{aligned}
&\mathbf{Pr}(\text{first hypothesis right} \\
&\quad \text{AND second hypothesis wrong} \\
&\quad \text{AND ...} \\
&\quad \text{AND } k\text{th hypothesis right}) \\
&\approx \mathbf{Pr}(\text{first hypothesis right}) \\
&\quad \times \mathbf{Pr}(\text{second hypothesis wrong}) \\
&\quad \times \ldots \\
&\quad \times \mathbf{Pr}(k\text{th hypothesis right}).
\end{aligned}$$

If it is successful in finding a large number of different hypotheses that are fairly accurate and whose errors are mutually independent, then it can hurt generalization to only make use of a small number of them. This is because if there are many (approximately) independent hypotheses participating in a vote, the variance of the fraction of them that are correct on a random instance will be smaller, and thus the fraction will be less likely to dip below $1/2$.

This phenomenon is illustrated dramatically with a very simple artificial dataset. There are 100 attributes that are conditionally independent given the class label, agreeing with it with probability 2/3, and 100 examples. On this data, boosting decision stumps results in hypotheses using nearly all the attributes, which achieve 97.8% accuracy on independent test data, whereas the minimum majority algorithm chooses small subsets (averaging 6.6) of the attributes, and only gets 77.2% correct.

Another artificial dataset provides an example of where the minimum majority algorithm has the advantage. In this dataset, there are 111 boolean attributes. The first 100 are independent flips of a fair coin, and a majority vote over the first 11 of these determines the class label. The final 11 attributes are conditionally independent given the resulting class label, agreeing with it with probability $0.5 + 1/\sqrt{11} \approx 0.8$ (which a simple calculation shows is greater than the probability that a determiner agrees with the class label). When there are 6667 examples, the conditionally independent attributes that correlate with the class label feature prominently in the boosted hypothesis, whereas in ten runs with data sets of this type, the minimum majority algorithm always consisted of a majority vote over precisely the 11 attributes that determine the class label. (This is encouraging evidence of the effectiveness of the optimization routine. Note that any greedy algorithm would include the conditionally independent attributes that correlate better with the class label.) The generalization of minimum majority is of course 100%, where boosting yields 99.1%.

The above two artificial datasets demonstrate a clear separation between boosting and following the minimum majority principle, and suggest the following extended explanation of the generalization ability of boosting. It appears that in addition to finding hypotheses that can be approximated with sparse, simple majority votes, boosting also makes maximal use of large collections of hypotheses whose errors are independent when such opportunities arise.

The experiments on the UC Irvine data (see Table 1) suggest that the minimum majority principle leads to a learning algorithm that for many applications generalizes nearly as well as boosting, but outputs simpler hypotheses based on fewer attributes, and is based on basic, easily understood principles.

Algorithm

In this paper, we will concentrate on the two-class case. The two-class case is sufficient to bring up the points we wish to explore, and, besides, much of the work on multiclass prediction using boosting proceeds by reducing to the two-class case (Freund & Schapire 1995; Schapire & Singer 2000).

The algorithm is perhaps best described in stages. First, we will describe the basic algorithm, then describe a series of modifications.

Generating a list of decision stumps

For each numerical variable, we create a list of thresholds by sorting the values that the variable takes on the dataset, and then putting a threshold halfway between each consecutive pair of values in the resulting list. For example, if a variable took the values 0, 3, 4, and 6, then the thresholds created would be 1.5, 3.5 and 5. For each threshold, two decision stumps are created, one that predicts 1 if the variable is at least the threshold, and one that predicts 1 if the variable is at most the threshold. Boolean variables are treated as numerical variables taking the values 0 and 1.

For each nominal variable i, for each value y that it takes, two decision stumps are created: one which predicts 1 exactly when variable i takes the value y, and one which prediction 1 exactly when variable i doesn't take the value y.

Finally, decision stumps that always predict 1 and always predict 0 are added.

The basic algorithm

The basic algorithm addresses the problem of finding the smallest multiset of decision stumps for which a majority

vote correctly classifies all the examples. This can be formulated as an integer program. Suppose there are n decision stumps and m examples, and let A be the $m \times n$ matrix in which $A_{i,j}$ is

- 1 if the ith decision stump is correct on the jth example,
- -1 if the ith decision stump is incorrect on the jth example,
- 0 if the variable queried has a missing value.

Then the problem can be phrased as that of setting nonnegative-integer-valued variables $x_1, ..., x_n$ to minimize $\sum_{i=1}^{n} x_i$ subject to the constraints that $Ax \geq (1, 1, ..., 1)^T$ (where $x = (x_1, ..., x_n)^T$). The ith variable is the number of times that ith decision stump gets to vote, and each constraint requires that the number of correct votes on a certain example is at least 1 more than the number of incorrect votes.

The basic algorithm

- solves the linear program obtained by removing the requirement that the variables are integers and gets the solution $(u_1, ..., u_n)$,
- sets

$$(p_1, ..., p_n) = (u_1, ..., u_n)/(u_1 + ... + u_n)$$

  to be the probability distribution over decision stumps in which the probability of decision stump i is proportional to the value of u_i,
- repeatedly randomly picks a variable using the distribution $(p_1, ..., p_n)$ keeping track for each variable i of the number of times x_i that i has been picked, until
- $Ax \geq (1, 1, ..., 1)^T$, at which time it quits and outputs x.

This algorithm has been shown to approximate the optimal solution in polynomial time (Long 2001). Two facts help one imagine why. First, if a solution is feasible, then the solution obtained by scaling all components by a constant factor is also feasible; e.g., if a certain solution calls for a certain collection of decision stumps to each vote once, then if instead they all always vote twice, all the votes come out the same way. Second, if $Z = \sum_{i=1}^{n} u_i$ and $u = (u_1, ..., u_n)$, then, after the algorithm has randomly chosen ℓ variables, the expectation of the current solution x is $(\ell/Z)u$.

The system used in our experiments solves the linear program using PCx (Czyzyk *et al.* 1999), which implements an interior-point algorithm.[2]

Resolving

Note that after the algorithm has committed to its first choice of a decision stump, it is left with a subproblem of a similar form. It wants to add as few decision stumps as possible in subsequent iterations, subject to updated constraints. For examples on which the first decision stump is correct, only half of the remaining choices need to be correct. For examples on which the first decision stump is incorrect, the number of subsequent decision stumps that are correct must be at least two greater than the number that are incorrect, in order to overcome the error made by the first decision stump. These constraints lead to different priorities than were in effect at the beginning, so presumably solving the linear programming relaxation of this new integer program should lead to a more appropriate probability distribution to use for choosing the second decision stump. This continues for future iterations. Our algorithm does this. This had a major effect on its performance in practice: prior to this modification, it was effectively useless.

Committing to the integer parts

The above process can be viewed as making successive commitments that the values of various variables are at least certain integer values, and incrementing one of these values at each iteration. This can be seen by expressing the integer program formulated at each iteration in terms of the original variables. To see how, let us focus again on the second iteration. Suppose that the decision stump chosen in the first iteration is number i_1. Suppose we formulate the integer program faced at the beginning of the second iteration using the same variables as the original integer program, where each variable is the number of times one decision stump will be chosen overall. Then the new integer program can be obtained from the old simply by adding the constraint $x_{i_1} \geq 1$. In general, if after a certain number of iterations the decision stumps numbered 1 through n have been chosen a_1 through a_n times respectively, then the integer program to solve in the next iteration is

$$\text{minimize} \sum_{i=1}^{n} x_i, \text{ s.t.}$$
$$\sum_{i=1}^{n} A_{ij} x_i \geq 1, \text{ for all examples } j$$
$$x_i \geq a_i, \text{ for all decision stumps } i.$$

Sometimes, the solution $(u_1, ..., u_n)$ to the linear programming relaxation of an integer program of the above form has many variables for which $u_i \geq a_i + 1$. Rather than wait for those values to be committed to in future iterations, we went ahead and committed to the integer parts of all variables before making the next random selection. For example, suppose there were three decision stumps, and the linear program returned a solution of $(2.1, 3.2, 5.4)$. Then the algorithm would commit to making the first variable at least 2, the second variable at least 3 and the third variable at least 5, adding these constraints for use in the next and subsequent iterations. If $(2, 3, 5)$ was not a feasible solution, then it would sample from the distribution obtained by normalizing the fractional parts, (1/7,2/7,4/7), to decide which variable whose commitment to increase, update the

[2] PCx generally worked impressively well on the linear programs generated by this application. However, with the default parameter settings, PCx infrequently either incorrectly reported infeasiblity, or crashed – when either happened, the system used in our experiments would simply start another attempt to improve the best solution found. Increasing `opttol`, `prifeastol`, and `dualfeastol` to 0.00001 and setting `refinement` to `no` appeared to help.

constraints to respond to this choice, and continue. This also improved performance markedly. To see why, consider what happens if the linear programming relaxation has an integer solution, which happens fairly often on the benchmark data we have experimented with. With the change of this subsection, this is recognized immediately and handled appropriately. The random sampling process that would otherwise be used would often be unlikely to get this solution.

Feature selection

When the data has a large number of numerical attributes, the number of decision stumps can be prohibitively large. Since the decision stumps output by the minimum majority algorithm often were contained in those output in the first 100 rounds of Adaboost, we performed a feature selection step in which Adaboost was run for 100 rounds and the resulting decision stumps were gathered and passed to the minimum majority algorithm described above. This change does not affect the motivation derived from the analysis of Adaboost (Schapire *et al.* 1998): their argument showed that the hypothesis could be approximated using a simple majority vote of a few hypotheses, even if those hypotheses were restricted to be chosen from among those participating in the weighted voting hypotheses output by the boosting algorithm.

Feature selection is only performed when the number of decision stumps generated by the original process was more than 100. (In our experiments, it was not performed on the `house` dataset, but on all others.)

Ensuring feasibility

We ensure feasibility using a method analogous to the "soft margin" approach from Support Vector Machines (Cortes & Vapnik 1995; Klasner & Simon 1995; Rätsch, Onoda, & Müller 2001). For each example, we add a "slack variable" to the integer program. The variable for an example has a coefficient of 1 in the constraint corresponding to that example, and does not appear in any other constraints. By increasing the value of that variable enough, one can therefore satisfy the one constraint in which it appears. These extra variables are assigned a cost of $1/4$ in the objective function. Of course this cost could be made a parameter, to be set using cross-validation on the training set. (The same value of $1/4$ was used in all of the experiments reported in this paper. This was not carefully optimized – it was chosen based on a little tinkering with some simple artificial datasets and the `ionosphere` data set.)

Restarting

An obvious modification is to run the algorithm several times, and output the best solution found. Of course, later runs can halt when the number of decision stumps in the solution becomes as large as the best solution found so far. In fact, later runs can halt whenever a fractional solution has a value that, when rounded up, is at least the best solution found so far, because the value of the integer solution found will be at least the value of the fractional solution, and will be an integer.

Dataset	Minmaj test set accuracy	Adaboost 100 rounds test set accuracy	Adaboost 10 rounds test set accuracy	Minmaj hyp. size
promoters	0.883	0.900	0.883	4.4
hepatitis	0.799	0.833	0.838	1.2
ionosphere	0.851	0.904	0.867	8.8
house	0.952	0.965	0.961	1.0
breast-cancer	0.941	0.950	0.949	4.8
pima	0.734	0.751	0.744	1.0
hypothyroid	0.992	0.991	0.990	5.0
sick-euthyroid	0.972	0.966	0.965	5.8
kr-vs-kp	0.965	0.958	0.937	13.0
mushroom	0.9996	1.0000	0.971	10.6
vote-condind	1.000	0.991	0.975	11.0
condind	0.772	0.978	0.838	6.6

Table 1: Our experimental results. Datasets above the double line were used in Freund and Schapire's experimental evaluation of Adaboost. The entry "Minmaj hyp. size" gives the average number of decision stumps participating in the final vote output by the minimum majority algorithm on that dataset – the other entries should be self-explanatory.

Giving up

In our experiments, the system halted when either 1000 consecutive attempts failed to improve on the best solution found so far, or when a total of two hours of wallclock time (on a loaded, common use, Sun-Fire) had been expended trying.

Initial solution

In early versions of the system, sometimes the first run of the algorithm took a long time before finding a feasible solution. To combat this, the present system first finds the best solution possible by combining a single decision stump with the slack variables described above. Each run of the randomized rounding algorithm halts when it fails to improve on this.

Experiments

We did experiments using a collection of data sets used in Freund and Schapire's experimental evaluation of Adaboost (Freund & Schapire 1996a). The datasets we used were the two-class datasets from the UC Irvine Machine Learning repository that were easiest to put into a common form. We also experimented with two artificial datasets that we created ourselves. For each dataset, we repeated the following experiment 10 times: hold out a random $1/3$ of the dataset, train on the remaining $2/3$ and evaluate the hypothesis on the held out $1/3$. We then added up the results from the 10 runs and tabulated them in Table 1.

The implementation of Adaboost in our experiments used the same decision stumps as the minimum majority algorithm, and ran for 100 iterations. We also ran Adaboost for only 10 iterations, in order to obtain hypotheses with roughly the complexity of those output by the minimum majority algorithm (note that Adaboost's hypotheses are *weighted* majority votes, however).

Dataset	#examples	#attr
promoters	106	57
hepatitis	155	19
ionosphere	351	34
house	435	16
breast-cancer	699	9
pima	768	8
hypothyroid	3163	25
sick-euthyroid	3163	25
kr-vs-kp	3196	36
mushroom	8124	22
vote-condind	10000	111
condind	150	100

Table 2: Some characteristics of the data used.

One trend that is visible is that the comparative performance of the minimum majority algorithm improves as the number of examples increases. This is consistent with the view that the advantage of Adaboost is due to exploitation of groups of hypotheses with independent errors. If the errors of the individuals are not too small, as the number of examples gets large, all members of such a group must be included in a majority vote for it to fit the data well. Note that for three of the datasets, the minimum majority algorithm either always or almost always outputs a single decision stump.

The artificial datasets are generated as follows. The dataset `vote-condind` is the dataset described in the introduction, in which 11 attributes determine the class label by majority vote, 11 are conditionally independent given the class label, agreeing with it roughly with probability 0.8, and 89 attributes are irrelevant. The dataset described in the introduction with 100 conditionally independent attributes agreeing with the label with probability $2/3$ is called `condind`.

Previous work

Perhaps the most closely related work is that of Grove and Schuurmans (1998) , who compared the performance of Adaboost with an algorithm that explicitly tried to maximize the margin using linear programming. The linear program solved in our algorithm for use to determine the probability distribution for the choice of the first decision stump is the same as linear program used to train the hypothesis weights in their algorithm. Mason, et al (2000) designed an algorithm, called DOOM, that chose the weights of a weighted voting classifier to optimize a cost function inspired by the general upper bound on generalization error from (Schapire *et al.* 1998) (the more general bound was in terms of the margin obtained when a certain fraction of the training data was excluded, together with that fraction). They showed that DOOM often improved on the generalization of Adaboost.

A number of papers have given a variety of interpretations of boosting. Friedman, et al (1998) showed that an algorithm closely related to boosting is obtained from additive logistic regression. An interpretation of boosting as gradient descent was given by Mason, et al (2000) . Kivinen and Warmuth (1999) showed that reweighting of the examples used by boosting was the solution of an optimization problem involving a tradeoff between keeping the relative entropy between the new and old weightings small and choosing a weighting orthogonal to the vector indicating whether the previous hypothesis made mistakes on the various examples (thereby driving the new hypothesis to make errors approximately independent of the old). Niyogi, et al (2001) also described a way in which boosting could be viewed as trying to find hypotheses with independent errors, and described an alternative algorithm pursuing this goal in conjunction with having small errors. Ali and Pazzani (1996) had established an empirical relationship between the tendency of a base classifier to return hypotheses with independent errors and the ability of an ensemble method to improve it. Freund and Schapire (1996) described a connection between boosting and game theory. Kutin and Niyogi (2001) performed an analysis of the generalization of Adaboost using the notion of algorithmic stability (Bousquet & Elisseeff 2001).

The analysis of Winnow (Littlestone 1988) in (Littlestone 1989) suggests that the inductive bias of Winnow might be similar to that of the minimum majority algorithm proposed here.

Conclusion

We have provided experimental evidence that an algorithm based on the minimum majority principle can generalize like boosting, while returning simpler hypotheses. This appears especially to be the case when the dataset is large. Our experiments also suggest an extension to the explanation of why boosted hypotheses generalize well: in addition to generating hypotheses that can be approximated by hypotheses of a simple form, they take full advantage of occasions in which many simple hypotheses have independent errors.

Source code associated with this work is available at `http://giscompute.gis.nus.edu.sg/~plong/minmaj`.

Acknowledgements

I'd like to thank Edison Liu, Sayan Mukherjee, and Vinsensius Vega for valuable conversations about this work and related topics, and Shirish Shevade, Vinsensius Vega and anonymous referees for their comments on drafts of this paper.

References

Abney, S.; Schapire, R.; and Singer, Y. 1999. Boosting applied to tagging and pp attachment. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Ali, K. M., and Pazzani, M. J. 1996. Error reduction through learning multiple descriptions. *Machine Learning* 24(3):173–202.

Bauer, E., and Kohavi, R. 1999. An empirical comparison of voting classification algorithm: Bagging, boosting and variants. *Machine Learning* 105–142.

Bousquet, O., and Elisseeff, A. 2001. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems 13*.

Breiman, L. 1998. Arcing classifiers. *The Annals of Statistics*.

Cohen, W.; Schapire, R.; and Singer, Y. 1999. Learning to order things. In *Advances in Neural Information Processing Systems 11: Proc. of NIPS'98*. MIT Press.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

Czyzyk, J.; Mehrotra, S.; Wagner, M.; and Wright, S. J. 1999. PCx: An interior-point code for linear programming. *Optimization Methods and Software* 11:397–430.

Drucker, H., and Cortes, C. 1996. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, 479–485.

Drucker, H.; Schapire, R.; and Simard, P. 1993. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence* 7:705 – 719.

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

Freund, Y., and Schapire, R. 1996a. Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*.

Freund, Y., and Schapire, R. 1996b. Game theory, on-line prediction and boosting. In *Proc. 9th Annu. Conf. on Comput. Learning Theory*, 325–332. ACM Press, New York, NY.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1):119–139.

Freund, Y.; Iyer, R.; Schapire, R.; and Singer, Y. 1998. An efficient boosting algorithm for combining preferences. In *Proc. 15th International Conference on Machine Learning*.

Freund, Y. 1995. Boosting a weak learning algorithm by majority. *Information and Computation* 121(2):256–285.

Friedman, J.; Hastie, T.; and Tibshirani, R. 1998. Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Sequoia Hall, Stanford Univerity.

Grove, A., and Schuurmans, D. 1998. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artifical Intelligence*.

Iba, W., and Langley, P. 1992. Induction of one-level decision trees. *Proc. of the 9th International Workshop on Machine Learning*.

Iyer, R.; Lewis, D.; Schapire, R.; Singer, Y.; and Singhal, A. 2000. Boosting for document routing. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*.

Kivinen, J., and Warmuth, M. 1999. Boosting as entropy projection. In *Proc. COLT'99*.

Klasner, N., and Simon, H.-U. 1995. From noise-free to noise-tolerant and from on-line to batch learning. *Proceedings of the 1995 Conference on Computational Learning Theory* 250–257.

Kutin, S., and Niyogi, P. 2001. The interaction of stability and weakness in adaboost. Technical Report TR–2001–30, The University of Chicago.

Littlestone, N. 1988. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning* 2:285–318.

Littlestone, N. 1989. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. Ph.D. Dissertation, UC Santa Cruz.

Long, P. M. 2001. Using the pseudo-dimension to analyze approximation algorithms for integer programming. *Proceedings of the Seventh International Workshop on Algorithms and Data Structures*.

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

Mason, L.; Baxter, J.; Bartlett, P. L.; and Frean, M. 2000. Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems 12*, 512–518. MIT Press.

Niyogi, P.; Pierrot, J.-B.; and Siohan, O. 2001. On decorrelating classifiers and combining them. Manuscript.

Pach, J., and Agarwal, P. 1995. *Combinatorial Geometry*. John Wiley and Sons.

Quinlan, J. 1996. Bagging, boosting and c4.5. In *Proceedings of the Thirteenth National Conference on Artifiicial Intelligence*, 725–730. AAAI/MIT Press.

Quinlan, J. R. 1999. Some elements of machine learning. *Proceedings of the Sixteenth International Conference on Machine Learning* 523–524.

Raghavan, P., and Thompson, C. 1987. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica* 7(4):365–374.

Rätsch, G.; Onoda, T.; and Müller, K.-R. 2001. Soft margins for AdaBoost. *Machine Learning* 42(3):287–320. also NeuroCOLT Technical Report NC-TR-1998-021.

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

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

Schapire, R.; Singer, Y.; and Singhal, A. 1998. Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval*.

Schapire, R. E. 1990. The strength of weak learnability. *Machine Learning* 5(2):197–226.

Williamson, D. P. 1999. Lecture notes on approximation algorithms. Technical Report RC–21409, IBM.