An Empirical Comparison of Three Boosting Algorithms on Real Data Sets with Artificial Class Noise

Ross A. McDonald[1], David J. Hand[1], and Idris A. Eckley[2]

[1] Imperial College London
[2] Shell Research Ltd.

Abstract. Boosting algorithms are a means of building a strong ensemble classifier by aggregating a sequence of weak hypotheses. In this paper we consider three of the best-known boosting algorithms: Adaboost [8], Logitboost [10] and Brownboost [7]. These algorithms are adaptive, and work by maintaining a set of example and class weights which focus the attention of a base learner on the examples that are hardest to classify. We conduct an empirical study to compare the performance of these algorithms, measured in terms of overall test error rate, on five real data sets. The tests consist of a series of cross-validatory samples. At each validation, we set aside one third of the data chosen at random as a test set, and fit the boosting algorithm to the remaining two thirds, using binary stumps as a base learner. At each stage we record the final training and test error rates, and report the average errors within a 95% confidence interval. We then add artificial class noise to our data sets by randomly reassigning 20% of class labels, and repeat out experiment. We find that Brownboost proves the least likely to overfit in this circumstance, because the algorithm incorporates an extra parameter which allows it to compensate for noisy examples.

1 Introduction

Boosting algorithms have their origins in the analysis of the theory surrounding the PAC (Probably Approximately Correct) learning model first introduced by Valiant in 1984 [22].

In the PAC framework, a data set is said to be strongly (PAC) learnable if there exists a classifier that can achieve an arbitrarily low error rate for all instances in the set. A weak learnable set requires only that the algorithm marginally outperform random guessing in terms of overall error rate. Kearns and Valiant [13] later proposed that these two definitions of learnability might be equivalent, and that this might be proven if it were shown to be possible to transform a weak learner into an arbitrarily strong one.

Schapire published the first hypothesis boosting algorithm in 1990 [19]. The more robust *Boost-by-Majority* (BBM) algorithm [6] was introduced by Freund at around the same time. The essential property of any boosting algorithm is that it is possible to derive an upper bound on the final training error rate. Both these precursor algorithms defined this upper bound in terms of γ, which is an amount by which the weak learner is guaranteed to outperform random guessing (so that in the two-class case, the weak learner would have to be guaranteed to yield an error rate below $\frac{1}{2} - \gamma$).

In practice it is usually unreasonable to assume that a base learning algorithm can always outperform a fixed error rate, and indeed the definition of a weak learner only requires that it should outperform random guessing by an arbitrarily small amount.

In 1995 Freund and Schapire published an adaptive algorithm known as Adaboost [8], which makes no prior assumptions about the base learner and has been the focus of much subsequent research. Adaboost is short for **Ada**ptive **Boost**ing, and the algorithm is characterised by the adaptive way that it generates and combines weak hypotheses. A monotonically decreasing upper bound on the training error can be derived, based on the performance of the individual component hypotheses. Thus if the base learner can consistently outperform random guessing, and we iterate long enough, we can eventually achieve any arbitrarily small error rate. It is also possible to derive an approximate upper bound for the the error rate of the fitted aggregate hypothesis when presented with new data.

It was subsequently observed [10] that Adaboost is in effect approximating a stagewise additive logistic regression model by optimising an exponential criterion. This leads us to new variants of Adaboost that fit additive models directly. One such variant is Logitboost, which uses the Newton-like steps to optimise the loss criterion.

In general terms, it has been observed that boosting algorithms do not tend to overfit within the number of iterations for which they are likely to be run. They are, however, particularly susceptible to class noise (where we take this to mean that a proportion of class labels have been redefined at random - but note that many authors use an alternative definition). Since the examples hardest to classify are very likely to be these noisy data, weak hypotheses induced at later iterations when such examples dominate will tend to be given undue influence in the final combined hypothesis, leading to a poor generalisation performance. In his empirical comparison of methods for constructing ensembles of decision trees [5], Dietterich concluded that 'the performance of Adaboost can be destroyed by classification noise'.

Brownboost [7], introduced by Freund and based on his Boost-by-Majority algorithm, may help to address this problem. It is derived by considering what happens to the BBM algorithm if the example reweighting is assumed to happen in continuous time. This leads us to an adaptive algorithm that resembles Adaboost, but which incorporates an extra parameter (the time parameter) that roughly corresponds to the proportion of noise in the training data. Because the algorithm knows in advance for how long it is to be run, it will not attempt to fit examples that are unlikely to be learnable in the remaining time. These are likely to be the noisy examples, so given a good estimate of the time parameter Brownboost is capable of avoiding overfitting. It can be shown [7] that Adaboost is a special case of Brownboost in the limit as the time parameter is allowed to tend to infinity.

In this paper, we conduct a series of empirical tests on four real data sets, using implementations of the Adaboost, Logitboost and Brownboost algorithms. We report our results in terms of overall test error rate. We then randomly reassign one in five class labels in each of the datasets and rerun the tests.

In Sections 2, 3 and 4 of this paper, we briefly describe each of the three boosting algorithms in turn. In setting out the Adaboost and Brownboost algorithms, we adopt notation that is consistent with the work of Freund and Schapire. The multi-class Logitboost algorithm is quoted from Friedman [10]. In Section 5, we describe our empirical study in detail, and report our findings. Finally, in Section 6 we summarise our conclusions.

2 Adaboost

Adaboost was the first adaptive boosting algorithm, and has received a good deal of attention since being introduced by Freund and Schapire in [8].

Our multi-class version of the algorithm uses the Hamming decoding and Error-Correcting Output Codes (ECOC) method of Allwein et al. (see [2] for a full description of this). The algorithm that we use is equivalent to the Adaboost.MH algorithm described in [21], and is an analogue of our own multi-class extension to the Brownboost algorithm [15].

Adaboost works by fitting a base learner to the training data using a vector or matrix of weights. These are then updated by increasing the relative weight assigned to examples that are misclassified at the current round. This forces the learner to focus on the examples that it finds harder to classify. After T iterations the output hypotheses are combined using a series of probabilistic estimates based on their training accuracy.

The Adaboost algorithm may be characterised by the way in which the hypothesis weights α are selected, and by the example weight update step. At iteration i, if γ_i is the *gain* of the weak learner over random guessing, then the hypothesis weight is chosen so that

$$\alpha_i = \frac{1}{2}\ln(\frac{1+\gamma_i}{1-\gamma_i}).$$

The weight update at step i multiplies the weights by an exponential function of the confidence of the prediction times the true label value, scaled by $-\alpha_i$.

In our multi-class adaptation of the algorithm, we maintain a separate weight for every example and class label. When calling our base learner, we take account of the possibility that this will either fit a binary (two-class) model, or a model that returns separate independent predictions for each of the k class labels. In the latter case, we assume that our coding matrix is the $k \times k$ matrix with 1 in all diagonal entries, and -1 everywhere else. We assume that hypotheses generated by base learners output confidence-rated predictions that are real values in the range $[-1, 1]$.

Adaboost Algorithm (Multi-Class Version)

Inputs:
ECOC Matrix: The $k \times \ell$ coding matrix $\mathbf{M}$.
Training Set: A set of m labelled examples: $T = (x_n, y_n), n = 1, ..., m$ where $x_n \in \mathbb{R}^d$ and $y_n \in \{y_1, y_2, ..., y_k\}$. Each y_n is associated via the matrix $\mathbf{M}$ with a set of ℓ binary labels $\{\lambda_1^n, ..., \lambda_\ell^n\}$, where $\lambda_j^n \in \{-1, 1\}$, $\quad j = 1, ..., \ell$.
Weights: An $m \times \ell$ vector of initial weights, say, $W_{1,j}(x_n, y_n) = \frac{1}{m\ell}$, $\quad n = 1, ..., m, \quad j = 1, ..., \ell$
WeakLearn – A weak learning algorithm.

Do for $i = 1, 2, ..., T$

1. **Binary base learner:** Call **Weaklearn** ℓ times $j = 1, ..., \ell$, each time passing it the weight distribution defined by normalizing $W_{i,j}(x_n, y_n)$ for fixed j, and the training data set alongside the binary labels defined by column j of the matrix $\mathbf{M}$.
   Multi-class base learner: Call **Weaklearn**, passing it the training data and the full set of weights.

   Receive from **Weaklearn** a set of ℓ hypotheses, $h_{i,j}(x)$, which have some advantage over random guessing

   $$\frac{\sum_{n=1}^m \sum_{j=1}^\ell W_{i,j}(x_n, y_n)(h_{i,j}(x_n)\lambda_j^n)}{\sum_{n=1}^m \sum_{j=1}^\ell W_{i,j}(x_n, y_n)} = \gamma_i > 0, \quad n = 1, ..., m, \quad j = 1, ..., \ell.$$

2. Select $\alpha_i = \frac{1}{2}\ln\left(\frac{1+\gamma_i}{1-\gamma_i}\right)$.
3. Weight update: $W_{i+1,j}(x_n, y_n) = \frac{W_{i,j}(x_n, y_n)\exp(-\alpha_i \ell_j^n h_{i,j}(x_n))}{\sum_{n=1}^m \sum_{j=1}^\ell W_{i,j}(x_n, y_n)}$.

Output Final hypotheses: $p_j(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i h_{i,j}(x)\right), \quad j = 1, ..., \ell.$

Figure 1: A Multi-class Adaboost Algorithm

3 Logitboost

The Logitboost algorithm [10] is based on the observation that Adaboost is in essence fitting an additive logistic regression model to the training data. An additive model is an approximation to a function $F(x)$ of the form

$$F(x) = \sum_{m=1}^{M} c_m f_m(x)$$

where the c_m are constants to be determined and the f_m are basis functions.

If we assume that $F(x)$ is the mapping that we seek to fit as our strong aggregate hypothesis, and the $f(x)$ are our weak hypotheses, then it can be shown that the two-class Adaboost algorithm is fitting such a model by minimising the criterion

$$J(F) = E(e^{-yF(x)})$$

where y is the true class label in $\{-1, 1\}$. Logitboost minimises this criterion by using Newton-like steps to fit an additive logistic regression model to directly optimise the binomial log-likelihood

$$-\log(1 + e^{-2yF(x)}).$$

This multi-class version of the algorithm is quoted from [10].

```
Logitboost Algorithm (Multi-Class Version)

1.  Start with weights $w_{i,j} = 1/N$,  $i = 1, ..., N$,  $j = 1, ..., J$,  $F_j(x) = 0$ and
    $p_j(x) = 1/J \quad \forall j$.

2.  Repeat for $m = 1, 2, ..., M$ :

        (a) Repeat for $j = 1, ..., J$:
            i. Compute working responses and weights for the $j$th class

                $z_{i,j} = \frac{y^*_{i,j} - p_j(x_i)}{p_j(x_i)(1-p_j(x_i))}$

                $w_{i,j} = p_j(x_i)(1 - p_j(x_i))$

            ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression of $z_{ij}$
            to $x_i$ with weights $w_{ij}$.

        (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^J f_{mk}(x))$, and $F_j(x) \leftarrow F_j(x) + f_{mj}(x)$.

        (c) Set $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^J e^{F_k(x)}}$, enforcing the condition $\sum_{k=1}^J F_k(x) = 0$.

3. Output the classifier $\text{argmax}_j F_j(x)$.
```

Figure 2: The multi-class Logitboost Algorithm, quoted from [10].

4 Brownboost

Brownboost is a continuous-time adaptive version of the Boost-by-Majority algorithm. Here we quote our own multi-class extension to this algorithm [15].

The derivation of the algorithm is beyond the scope of this paper, but we briefly summarise some of its key points.

The 'total time' parameter, c, sets the total amount of time for which the algorithm is set to run. At each iteration a quantity t is subtracted from this, and the algorithm terminates when the remaining time s reaches 0.

For every example (x_n, y_n) and class j, the algorithm maintains a margin. These are all initially set to 0, and at iteration i they are updated to:

$$r_{i+1,j}(x_n, y_n) = r_{i,j}(x_n, y_n) + \alpha_i h_{i,j}(x_n)\lambda_j^n$$

where λ_j^n is the label related to the example for class j by the ECOC matrix.

The hypothesis weights α_i are derived by solving a differential equation, and the weight updates are a function of these and the margin.

We can relate the parameter c to the final training error ϵ of the strong hypothesis via

$$\epsilon = 1 - \text{erf}(\sqrt{c}) \tag{4.1}$$

where 'erf' is the error function. Thus we can select c to guarantee any desired final error.

Brownboost Algorithm (Multi-Class Version)

Inputs:
ECOC Matrix: The $k \times \ell$ coding matrix $\mathbf{M}$.
Training Set: A set of m labelled examples: $T = (x_n, y_n), n = 1, ..., m$ where $x_n \in \mathbb{R}^d$ and $y_n \in \{y_1, y_2, ..., y_k\}$. Each y_n is associated via the matrix $\mathbf{M}$ with a set of ℓ binary labels $\{\lambda_1^n, ..., \lambda_\ell^n\}$, where $\lambda_j^n \in \{-1, 1\}, j = 1, ..., \ell$.
WeakLearn – A weak learning algorithm.
c – a positive real valued parameter.
$\nu > 0$ – a small constant used to avoid degenerate cases.

Data Structures:
prediction value: With each example we associate a set of real valued margins.
The margin of example (x_n, y_n) for label λ_j^n on iteration i is denoted $r_{i,j}(x_n, y_n)$. The initial value of all margins is zero: $r_{1,j}(x_n, y_n) = 0, n = 1, ..., m, j = 1, ..., \ell$.

Initialize 'remaining time' $s_1 = c$.
Do for $i = 1, 2, ...$

1. Associate with each example and label a positive weight

$$W_{i,j}(x_n, y_n) = e^{-(r_{i,j}(x_n,y_n)+s_i)^2/c}, n = 1, ..., m, j = 1, ..., \ell,$$

2. **Binary base learner:** Call **Weaklearn** ℓ times $j = 1, ..., \ell$, each time passing it the weight distribution defined by normalizing $W_{i,j}(x_n, y_n)$ for fixed j, and the training data set alongside the binary labels defined by column j of the matrix $\mathbf{M}$.
Multi-class base learner: Call **Weaklearn**, passing it the training data and the full set of weights.

   Receive from **Weaklearn** a set of ℓ hypotheses $h_{i,j}(x)$ which have some advantage over random guessing

$$\frac{\sum_{n=1}^m \sum_{j=1}^\ell W_{i,j}(x_n,y_n)(h_{i,j}(x_n)\lambda_j^n)}{\sum_{n=1}^m \sum_{j=1}^\ell W_{i,j}(x_n,y_n)} = \gamma_i > 0.$$

3. Let γ, α and $\mathbf{t}$ be real valued variables that obey the following differential equation:

$$\frac{d\mathbf{t}}{d\alpha} = \gamma = \frac{\sum_{n=1}^m \sum_{j=1}^\ell \exp(-\frac{1}{c}(r_{i,j}(x_n,y_n)+\alpha h_{i,j}(x_n)\lambda_j^n+s_i-\mathbf{t})^2)h_{i,j}(x_n)\lambda_j^n}{\sum_{n=1}^n \sum_{j=1}^\ell \exp(-\frac{1}{c}(r_{i,j}(x_n,y_n)+\alpha h_{i,j}(x_n)\lambda_j^n+s_i-\mathbf{t})^2)}$$

   where $r_{i,j}(x_n, y_n)$, $h_{i,j}(x_n)$ and s_i are constants in this context.
   Given the boundary conditions $\mathbf{t} = 0, \alpha = 0$ solve the set of equations to find $t_i = \mathbf{t}^* > 0$ and $\alpha_i = \alpha^*$ such that either $\gamma^* \leq \nu$ or $\mathbf{t}^* = s_i$.

4. Margin update: $r_{i+1,j}(x_n, y_n) = r_{i,j}(x_n, y_n) + \alpha_i h_{i,j}(x_n)\lambda_j^n$.

5. Update 'remaining time' $s_{i+1} = s_i - t_i$.

Until $s_{i+1} \leq 0$
Output Final hypotheses: $p_j(x) = \text{erf}\left(\frac{\sum_{i=1}^N \alpha_i h_{i,j}(x)}{\sqrt{c}}\right), j = 1, ..., \ell$.

Figure 3: A multi-class Brownboost algorithm based on [7].

5 The Experiments

We conducted a series of tests using the four data sets summarised in Table 1. All of these data sets, with the exception of *Credit* were taken from the *UCI Machine Learning Repository* [1].

The *Credit* data set is a credit scoring data set of the type commonly found in commercial banking. Details of these data have been omitted from Table 1 for commercial reasons.

Wisconsin is the well-known diagnostic data set for breast cancer compiled by Dr William H. Wolberg, University of Wisconsin Hospitals [14].

The *Wine* data, based on a chemical analysis of Italian wines, and *Balance* data, which records the results of a psychological experiment, are three-class data sets which have been included to test the multi-class versions of the algorithms.

Data Set	Entries	Attributes	Classes	Class Distribution
Wisconsin	699	9	2	241,458
Credit	500	–	2	–
Wine	178	13	3	59,71,48
Balance	625	4	3	288,49,288

Table 1: Summary table for the data sets used in experiments.

In order to ensure algorithmic convergence in the time available, the *Credit* data set was curtailed to 500 examples. Indicator variables were substituted for categorical variables where these occurred (see [11], Section 9.7 for more details).

We constructed a noisy version of each data set by assigning a randomly chosen, incorrect class label to 20% of training examples.

We implemented Adaboost and Brownboost in Matlab, using a purpose written binary stump algorithm as our base learner. Logitboost was implemented in S-Plus using the internal *tree* function to generate binary stumps.

Each experiment consisted of 100 trials (25 for Logitboost). At each trial, one third of the data examples were selected at random and set aside as a test set. The remaining two thirds of examples were used to train the algorithm. We recorded the final error rates of the output hypothesis on both the training and test data.

We trained Adaboost and Logitboost on the original data to give us a benchmark for our comparison (recall that Adaboost is equivalent to Brownboost when the final training error is set to 0).

We then trained all three algorithms on the noisy data.

In all trials Adaboost was allowed to run until its training loss matched the expected training loss rate of that data set, or for a maximum of 100 iterations. We used the *one-against-all* approach, so our coding matrix was the $k \times k$ matrix whose diagonal entries are all 1, with all other entries -1.

Logitboost was allowed to run until its training loss matched the expected loss rate, or up to a maximum of 20 iterations. We avoided numerical instabilities in this algorithm using the prescriptions in [10].

For Brownboost, we calculated the appropriate values for c using equation 4.1.

The training and test loss and error rates for each trial are recorded for a 95% confidence interval in Tables 2 and 3.

0% Artificial Class Noise				
	Adaboost		Logitboost	
Data Set	Training Error	Test Error	Training Error	Test Error
Wisconsin	0.034 ± 0.002	0.048 ± 0.003	0.012 ± 0.002	0.037 ± 0.005
Credit	0.078 ± 0.002	0.105 ± 0.005	0.030 ± 0.003	0.102 ± 0.006
Wine	0.000 ± 0.000	0.041 ± 0.005	0.000 ± 0.000	0.046 ± 0.013
Balance	0.076 ± 0.002	0.185 ± 0.005	0.030 ± 0.005	0.124 ± 0.008

Table 2: Error rates for Adaboost and Logitboost on the unmodified data sets, 95% confidence intervals, 0% artificial class noise.

20% Artificial Class Noise						
	Adaboost		Logitboost		Brownboost	
Data Set	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
Wisconsin	0.216 ± 0.003	0.238 ± 0.005	0.190 ± 0.004	0.238 ± 0.009	0.188 ± 0.001	0.230 ± 0.004
Credit	0.239 ± 0.003	0.316 ± 0.010	0.190 ± 0.005	0.281 ± 0.009	0.177 ± 0.002	0.289 ± 0.005
Wine	0.088 ± 0.003	0.287 ± 0.008	0.171 ± 0.012	0.256 ± 0.019	0.165 ± 0.003	0.255 ± 0.011
Balance	0.214 ± 0.003	0.337 ± 0.006	0.185 ± 0.005	0.247 ± 0.012	0.158 ± 0.002	0.280 ± 0.006

Table 3: Error rates for Adaboost, Logitboost and Brownboost on the data sets with 20% artificial class noise, 95% confidence intervals.

6 Discussion of Results and Conclusions

It is interesting that Logitboost proves better able to fit the unmodified versions of our four data sets, and that it appears to yield a much better generalisation error than Adaboost. The exception is the wine data set, which is the only one that both algorithms were able to learn fully in all trials (presumably because it consists of relatively few examples). But in this case the confidence interval for Logitboost is too wide to draw any firm conclusions.

Broadly speaking, our results bear out claims that Adaboost is especially susceptible to class noise, while providing strong evidence that Brownboost is particularly robust in such situations. We were surprised that Logitboost compares so favourably with Brownboost, which suggests that some property of the algorithm (possibly the fact that it implicitly avoids making large weight updates) makes it especially robust to overfitting.

It is immediately evident from the test error rates in Tables 2 and 3 that the introduction of class noise to real data seriously impairs the generalisation performance of Adaboost. This would appear to tally with the observations made by Dietterich in [5].

When implementing Brownboost, we were able to calculate the value of c directly given our prior knowledge. Of course, in a real situation we would be very unlikely to know the level of class noise in advance. It remains to be seen how difficult it would prove to estimate c in practice.

Acknowledgements

RAM was supported in this work by Shell Research Ltd. and research grant number 0130322X from the Engineering and Physical Sciences Research Council.

References

1. *UCI Machine Learning Repository.* `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
2. E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113 – 141, 2000.
3. E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36:105 – 142, 1999.
4. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, U.S., 1984.
5. T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *AI Magazine*, 18:97 – 136, 1997.
6. Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121, 1995.
7. Y. Freund. An adaptive version of the boost by majority algorithm. *Machine Learning 43*, 3:293 – 318, 2001.
8. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Second European Conference on Computational Learning Theory*, 1995.
9. Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14:771 – 780, 1999.

10. J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28:337 – 374, 2000.
11. D. J. Hand. *Construction and Assessment of Classification Rules.* John Wiley & Sons, Chichester, 1997.
12. W. Jiang. Some results on weakly accurate base learners for boosting regression and classification. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 87 – 96, 2000.
13. M. Kearns and L. G. Valiant. Learning boolean formulae or finite automata is as hard as factoring. *Technical Report TR-14-88, Harvard University Aiken Computation Laboratory*, 1988.
14. O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1 – 18, 1990.
15. R. A. McDonald, I. A. Eckley, and D. J. Hand. A multi-class extension to the brownboost algorithm. *In Submission.*
16. J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, San Mateo, CA, 1986. Morgan Kauffmann.
17. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
18. J. R. Quinlan. Bagging, boosting and c4.5. *AAAI/IAAI*, 1:725 – 730, 1996.
19. R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197 – 227, 1990.
20. 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*, 26:1651 – 1686, 1998.
21. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297 – 336, 1999.
22. L. G. Valiant. A theory of the learnable. *Artificial Intelligence and Language Processing*, 27:1134 – 1142, 1984.