Analysis of the Performance of AdaBoost.M2 for the Simulated Digit-Recognition-Example

Günther Eibl and Karl Peter Pfeiffer

Institute of Biostatistics, Innsbruck, Austria
guenther.eibl@uibk.ac.at

Abstract. In simulation studies boosting algorithms seem to be susceptible to noise. This article applies Ada.Boost.M2 used with decision stumps to the digit recognition example, a simulated data set with attribute noise. Although the final model is both simple and complex enough, boosting fails to reach the Bayes error. A detailed analysis shows some characteristics of the boosting trials which influence the lack of fit.

1 Introduction

Boosting algorithms were originally designed to turn weak classifiers into strong ones. The performance of such boosted classifiers turned out to be so good that they became an own discipline in the field of pattern recognition. However boosting algorithms seem to have disadvantages at noisy data sets ([1, 4, 6, 7, 10]). The aim of this paper is to apply a boosting algorithm to a simulated data set with attribute noise, analyze its behaviour and find some characteristics that affect the performance of boosting. The digit example is a well known artificial example, so it is possible to compare the result with the true structure of the problem. The chosen boosting classifier is AdaBoost.M2 with decision stumps as base classifiers.

2 AdaBoost.M2 with decision stumps for the digit dataset

2.1 The digit dataset

The artificial digit example is a simulation of a faulty digit display ([2]). The representation of a digit on a display consists of 7 lights, which can be lighted or not (Fig.1). Each representation x of a digit is an element of the set $\{0,1\}^7$, where $x_j = 1$ means, that light number j is lighted and $x_j = 0$ means, that light number j is not lighted. The proper representation of digits $0, \ldots, 9$ is given in figure 1. For example the proper representation of digit 1 is $x = (0, 0, 1, 0, 0, 1, 0)$. One can also say, that $x = (0, 0, 1, 0, 0, 1, 0)$ is the *prototype* of group 1.

The display is simulated to be faulty in displaying digits. All seven lights are independently properly lighted with probability 0.9 and faultily lighted with probability 0.1. So the probability, that a digit is properly represented is 0.9^7,

Fig. 1. Representation of a digit and prototypes of the digits

which is about 0.48. Now we define the *similarity* of a digit x to a group g as the number of identical lights of x and the prototype of group g. The Bayes classifier assigns an unknown digit x to the most similar group. If this most similar group is not uniquely defined, the group gets chosen by chance among the most similar groups. The error rate of the Bayes classifier is 0.26. The most important characteristics of the data set are given in Table 1

Table 1. Properties of the digit example

♯ groups	♯ input variables	♯ training set	Bayes error	SNR
10	7	1000	0.26	0.9

2.2 Decision stumps

Since we use decision stumps as base classifiers we introduce some notation here. A decision stump h is a classification tree with 2 leaves. For the digit example this means that the decision stump chooses a variable j and divides the learning set $\{(x_1, g_1), \ldots, (x_N, g_N); x_i \in X, g_i \in G\}$ into the two leaves $l_0 := \{(x_i, g_i); x_{ij} = 0\}$ and $l_1 := \{(x_i, g_i); x_{ij} = 1\}$. Now we call N_k the number of elements of leave l_k and $N_k(g)$ the number of elements of group g in leave l_k. Then the decision stump estimates the probability that a new, unknown digit x belongs to group g as the corresponding group proportion $\pi_0(g) := N_0(g) \backslash N_0$ if $x_j = 0$ and $\pi_1(g) := N_1(g) \backslash N_1$ if $x_j = 1$.
The result is a map

$$h : \{0,1\}^7 \times \{0,1\}^{10} \to [0,1], \quad h(x,g) := \pi_0(g)[[x_j = 0]] + \pi_1(g)[[x_j = 1]], \quad (1)$$

where $[[\cdot]]$ is a function of boolean values with $[[\text{true}]] = 1$ and $[[\text{false}]] = 0$.

2.3 AdaBoost.M2

Boosting algorithms are often designed for classification of two groups. There are several extensions for problems with more than two classes ([3, 5, 8, 10]).

With X designating the input space and G designating the set of groups the straightforward multiclass extension, AdaBoost.M1, combines base classifiers of the form $h : X \to G$ by majority vote like in the twoclass case. A big drawback of this approach is that the error rate of each base classifier should be less than 50% which can be difficult to reach especially for the multiclass case. For the digit example the error rate of a single decision stump which assigns a case to the group with highest probability is about 80%. So the use of the straightforward multiclass extension with decision stumps has a poor performance.

A more sophisticated multiclass extension is AdaBoost.M2 (Fig.2) published by Freund and Schapire ([5]). AdaBoost.M2 changes the goal of the weak classifiers to predict a set of plausible groups and evaluates the weak classifiers using the pseudo-loss ϵ_t (step 3) which penalizes the weak classifiers for failing to include the correct group in the predicted plausible group set and for each incorrect label in the predicted plausible group set. The exact form of the pseudo-loss is under control of the algorithm so that the weak classifier can focus also on the groups which are hardest to distinguish from the correct group by changing the matrix q. The following theorem guarantees the decrease of the training error as long as the pseudo-loss is less than 1/2 which is much more easier to achieve than training error less than 50%:

Theorem 1. *Let $\epsilon_t \leq \frac{1}{2} - \gamma$ for $\gamma \in (0, 1)$. Then the training error of AdaBoost.M2 is upper bounded by*

$$(|G| - 1)\ 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)} \leq \exp(-2\gamma^2 T).$$

In every boosting round t the algorithm takes a bootstrap sample, where each case (x_i, g_i) is weighted with $D_t(i)$, from the learning set. This bootstrap sample is used for the construction of the weak classifier (step 2). Then the pseudo-loss of this classifier is calculated (step 3). The pseudo-loss is used for three purposes: First it is used for updating the sampling weights $D(i)$ and the matrix q for the next boosting round (steps 4,5 and 1). The algorithm focuses more on cases that were difficult to classify properly at the previous boosting round. Second, according to the theorem above, the algorithm stops if the pseudo-loss reaches 1/2 (step 3). The third time the pseudo-loss is used for the weighted majority-vote of the T base classifiers (step 4 and output).

3 Application of AdaBoost.M2 with decision stumps on the digit-example

To study the algorithm systematically we applied it to 50 independently generated data sets of size 1000. Other algorithms were also applied to these 50 data sets using the number of boosting rounds of the original algorithm. Since the example is artificial the expected error could be calculated directly, so there was no need for a test set. All figures come from trial 31, for which both minimal and final training error are near the corresponding mean errors over all datasets. In figure 3 and table 2 scores are multiplied with 100 for sake of readability.

Algorithm AdaBoost.M2

Input: learning set $\{(x_1, g_1), \ldots, (x_N, g_N); x_i \in X, g_i \in G\}$, $G = \{1, \ldots, |G|\}$
weak classifier of the form $h : X \times G \to [0, 1]$
T: maximum number of boosting rounds

Initialization: $D_1(i) = 1/N$, weight vector $w^1_{i,g} = D_1(i)/(|G| - 1)$

For $t = 1, \ldots, T$:

1. Set $W^t_i = \sum_{g \neq g_i} w^t_{i,g}$

   for $g \neq g_i$: $q^t_{i,g} = \frac{w^t_{i,g}}{W^t_i}$

   and

   $D_t(i) = W^t_i / \sum_{i=1}^{N} W^t_i$
2. Call the weak classifier h_t with data sampled from the learning set with distribution D_t
3. Calculate the pseudo-loss ϵ_t of h_t:

   $\epsilon_t = \frac{1}{2} \sum_{i=1}^{N} D_t(i) \left(1 - h_t(x_i, g_i) + \frac{1}{|G|-1} \sum_{g \neq g_i} q^t_{i,g} h_t(x_i, g) \right)$

   if $\epsilon_t \geq \frac{1}{2}$: $T := t - 1$, **goto** output step
4. Set $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
5. For $i = 1 \ldots, N$ and $g \in G \backslash \{g_i\}$ set the new weight vectors to be

   $w^{t+1}_{i,g} = w^t_{i,g} \mathrm{e}^{-\alpha_t (1 - h_t(x_i, g_i) + h_t(x_i, g))}$

Output: final classifier $h_f(x)$: Normalize $(\alpha_t)_{t=1,\ldots,T}$ and set

$h_f(x) = \arg\max_{g \in G} \sum_{t=1}^{T} \alpha_t h_t(x, g)$

Fig. 2. AdaBoost.M2

3.1 The final model

As we have seen in the previous section the final classifier has the following form:

$$h_f(x) \quad = \quad \arg\max_{g \in G} \sum_{t=1}^{T} \alpha_t h_t(x, g) \ . \tag{2}$$

We can interpret the sum in (2) as a score for group g. Digit x is classified as the digit g with the highest score for instance x. For decision stumps each base hypothesis h_t depends only on one variable j which we will call $j(t)$. Inserting the definition of the decision stumps (1) in (2) we get after some rearrangements of terms

$$\begin{aligned} h_f(x) \quad &= \quad \arg\max_{g \in G} \sum_{t=1}^{T} \alpha_t \left(\pi^t_0(g)[[x_{j(t)} = 0]] + \pi^t_1(g)[[x_{j(t)} = 1]] \right) \\ &= \quad \arg\max_{g \in G} \sum_{t=1}^{T} \alpha_t \left(\pi^t_0(g) + (\pi^t_1(g) - \pi^t_0(g)) x_{j(t)} \right) \end{aligned}$$

$$= \quad \arg\max_{g\in G}\left(\sum_{t=1}^{T}\alpha_t\pi_0^t(g)+\sum_{t=1}^{T}\sum_{j=1}^{7}\alpha_t[[j=j(t)]](\pi_1^t(g)-\pi_0^t(g))x_j\right) \quad .$$

Setting

$$a_0(g) := \sum_{t=1}^{T}\alpha_t\pi_0^t(g)$$

and

$$a_j(g) := \sum_{t=1}^{T}\alpha_t[[j=j(t)]](\pi_1^t(g)-\pi_0^t(g))$$

we finally get

$$h_f(x) \;=\; \arg\max_{g\in G}\mathrm{score}(g)(x) \tag{3}$$

$$\mathrm{score}(g)(x) = a_0(g)+\sum_{j=1}^{7}a_j(g)x_j \quad . \tag{4}$$

The resulting classifier can be shown graphically (Fig.3).

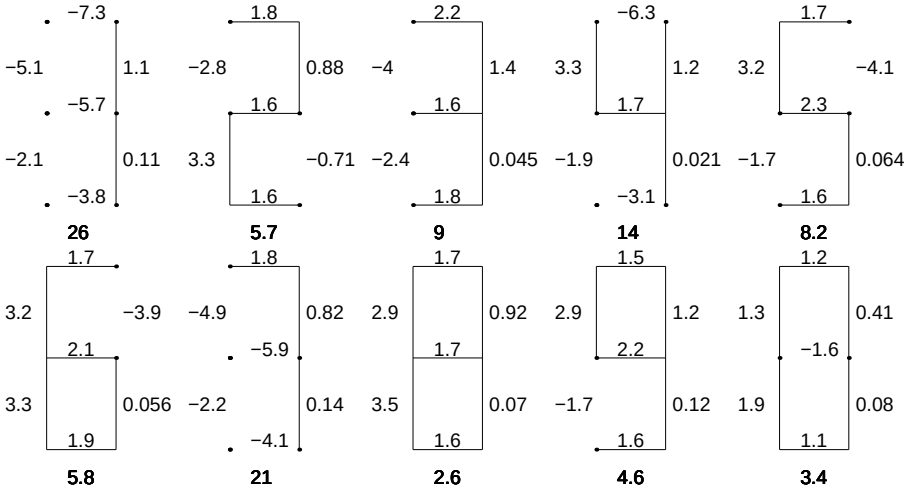

Fig. 3. Graphical representation of the resulting classifier

The figure consists of the prototypes of the 10 groups. To calculate for case x the score for group g look at the corresponding prototype. The number below is $a_0(g)$. Then for each of the 7 lights look at the number beside and above respectively and add it if light j of x is lighted ($x_j = 1$), otherwise add nothing.

As a result the final classifier can be represented by voting of the 10 scores score(1),. . .,score(10) where each score is a linear function of the 7 values x_j. The

final classifier h_f is determined by fixing the 80 constants $a_j(g),\ j = 0, \ldots, 7,\ g = 0, \ldots, 10$. This is a hint that we can get a quite complex classifier even from such simple base classifiers as decision stumps. Using classification trees with more leaves leads to interactions of higher orders with the disadvantage of loosing an easy interpretation like above.

3.2 Performance of the algorithm

The boosting routine quickly diminishes the training error, showing some jumps of about 5 percent. The training error rate reaches a minimum and then slightly increases to a final training error (Fig.4). The jumps of the curve come from the fact, that at certain rounds t a prototype of a digit, which contains about 5% of all data, is assigned correctly and incorrectly respectively. Figure 4 shows that the pseudo loss increases with t, so the weights of the base classifiers in the combination step decrease. Therefore the changes of the final classifier get smaller which leads to the smoothness of the error curve at higher boosting rounds.

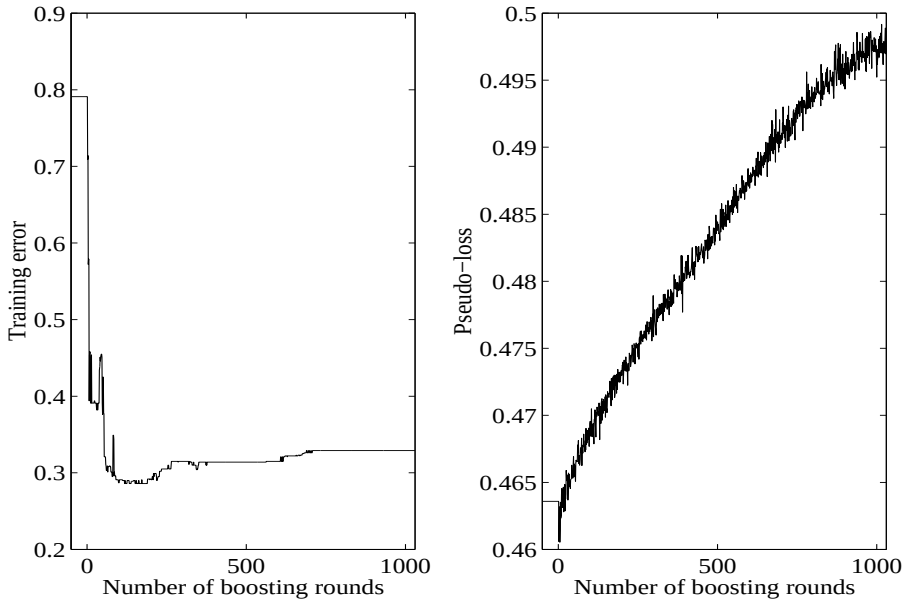

Fig. 4. Error (left) and pseudo loss (right) dependent on boosting round

This error curve is a typical one as can be seen by comparison with Table 3. The mean expected error of 34.3% is good in comparison with the error of the best single decision stump of 80%, but a comparison of the minimum test error

with the Bayes error of 26% shows that there is much room for improvement. Despite the additive structure of the problem and the simplicity of the final model the algorithm didn't manage to fit it well.

To be sure that the final model is complex enough we built an additive model whose scores are just the number of lights identical to the lights of the prototype of the corresponding group (Fig.5) which is just the Bayes classifier. Therefore we solved 10 systems of 128 linear equations. The ranks of all 10 corresponding matrices were full, so the representation of the desired score as an additive model is unique.

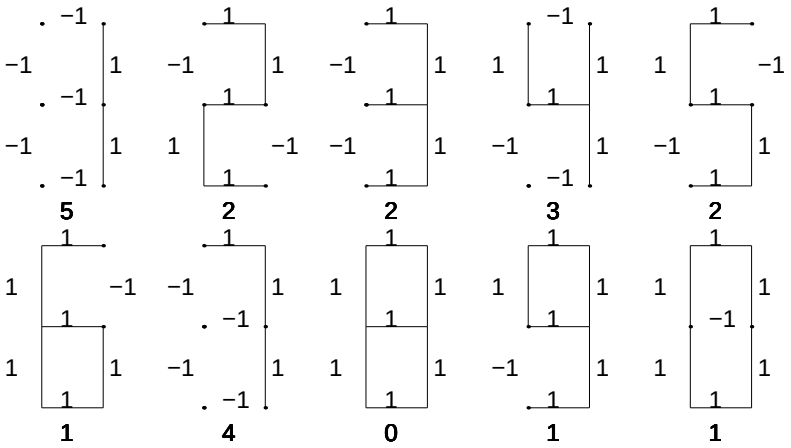

Fig. 5. Optimal additive model

An analysis of the results shows, that many digits are assigned wrongly to groups 1 and 7, because the scores for these two groups can get higher than the scores for the other eight groups (Table 2). An analysis of the algorithm shows that these two groups were chosen with higher probability the longer the boosting algorithm calculated (see Table 2 and Fig.6).

Table 2. Analysis of algorithm: $p_f(g)$: mean relative frequency of cases in the training set assigned to group g; score for group g: mean score(g) for cases in the training set of group g; Δp: mean difference between the group proportions at the last and the first boosting round in percent

group g	1	2	3	4	5	6	7	8	9	0
$p_f(g)$	**14.3**	8.9	6.5	10.0	10.1	12.0	**15.2**	9.1	6.1	7.8
score for group g	**23.0**	15.2	13.0	16.7	14.7	16.1	**19.6**	13.9	12.3	14.5
Δp	**9.9**	-3.8	-2.6	0.5	-2.9	-2.6	**10.7**	-2.6	-3.3	-3.4

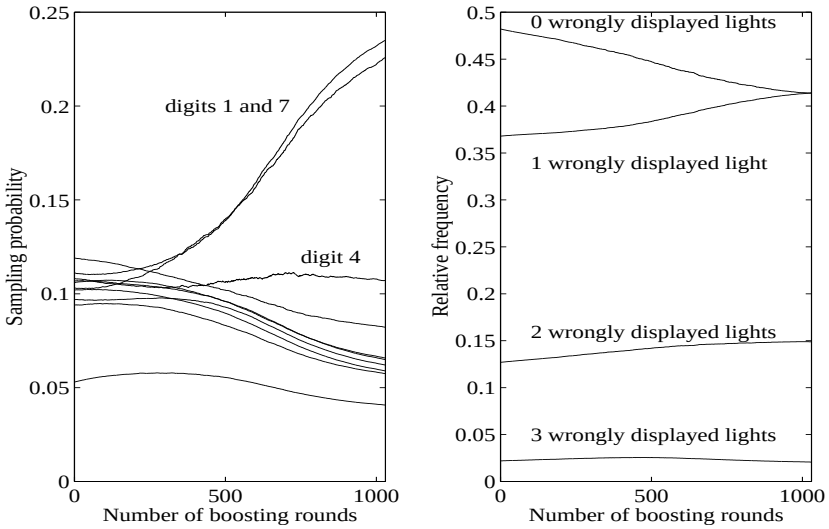

Fig. 6. Proportion of groups $\delta_t(g)$ (left) and wrongly lighted lights (right) in weighted bootstrap sample dependent on boosting round

The idea of the boosting algorithm is, that it concentrates on hard-to-classify training data. Since there is no obvious reason, why digits 1 and 7 should be harder to classify than the others we made an ad-hoc regularization of the algorithm by holding the group proportions constant. Therefore we inserted an additional normalization between step 1 and step 2 of AdaBoost.M2, we name it

$$\text{step 1.5 :} \qquad D_t(i) = D_t(i)\pi(g)/\delta_t(g), \quad \text{with } \delta_t(g) := \sum_{i;g_i=g} D_t(i) \ . \tag{5}$$

A natural choice for the constant group proportion $\pi(g)$ is the group prior estimated directly from the data

$$\pi(g) = \hat{\pi}(g) := \frac{N(g)}{N} \ .$$

We denote the algorithm using $\hat{\pi}(g)$ as AdaBoost$\hat{\pi}$. Since we know the real group proportion $\pi(g) = 0.1$ from theory we also used this choice and denote the corresponding algorithm as AdaBoostπ. Table 3 shows that AdaBoost$\hat{\pi}$ worked worse than the original algorithm, but AdaBoostπ brought a substantial improvement, so the exact knowledge about the true group proportions appears to be a crucial point for the modification. For a comparison we also applied bagging with confidence-rated votes to the data sets and got much worse classifiers (Table 3).

It's also interesting to see how the different error rates are made up of minimal training error, *overfit* and *generalization error*, where the overfit is defined as the difference between the final and minimal reached training error and the generalization error is the difference between the expected and training error

of the classifier (Table 3). The two modified algorithms turned out to be more robust to overfitting, and as expected AdaBoostπ generalizes best. The greatest differences between the algorithms were seen in the minimal training error, where the 4 algorithms had the following ranking: AdaBoostπ, original AdaBoost.M2, AdaBoost$\hat{\pi}$, bagging. AdaBoost$\hat{\pi}$ comes closer to original AdaBoost.M2 because of its lower overfit, but this ranking also holds for the expected error of the final classifier.

Table 3. Comparison of algorithms

		AdaBoost.M2	AdaBoost$\hat{\pi}$	AdaBoostπ	Bagging
Final expected error	MW	34,31	36,65	27,79	48,69
	STD	5,45	5,80	0,76	9,08
	MIN	26,74	27,0	26,58	30,55
	MAX	48,69	53,77	30,20	74,80
Minimal training error	MW	28,67	32,54	26,06	40,06
	STD	3,15	4,82	1,47	7,41
	MIN	23,30	25,00	22,60	24,5
	MAX	37,20	47,20	29,70	61,5
Final training error	MW	32,60	34,26	27,33	45,32
	STD	4,74	5,36	1,66	8,36
	MIN	25,40	25,90	23,20	27,20
	MAX	45,40	48,80	30,80	71,10
Overfit	MW	3,93	1,72	1,27	5,26
	STD	2,98	2,03	0,79	5,51
	MIN	0,20	0,00	0,10	0,00
	MAX	12,4	7,30	3,50	31,00
Generalization error	MW	1,71	2,39	0,47	3,37
	STD	1,61	1,54	1,54	1,52
	MIN	-2,66	-1,18	-2,93	-1,21
	MAX	4,92	5,86	3,44	6,14

As we have seen above AdaBoost.M2 has some overfit which leads to the question how to choose the final boosting round T. We compared the original final classifier to the one which is the result of stopping at the minimal training error and the one which stops at the maximal sum of margins ([9], Table 4). Stopping at the minimal training error is clearly superior and leads to an improvement in expected error of $3.99 \pm 3.82\%$. The maximal margin approach was supposed to have less generalization error, but the generalization error was the same for all three stopping criteria.

Table 4. Description of errors at the last round, the round with minimal training error (minerror) and the round with maximal margin (maxmargin)

method	T	training error(T)	expected error(T)
last round	1215±282	32.60±4.74	34.31±5.45
minerror	447±226	28.67±3.15	30.04±3.81
maxmargin	330±88	30.99±4.29	32.53±5.06

Another interesting question is how noisy digits behave during the algorithm. To investigate this we saved at each boosting round the sampling weights of right represented digits and digits with 1, 2 and 3 wrongly lighted lights respectively. As expected the proportion of digits with one and two wrongly lighted lights increased during boosting (Fig.6). Looking at the correlation of the proportion of lights with one and two false lights at the last step and the final test error we get -0.3516 and 0.391. Our interpretation is that most digits with one wrong light can get classified well whereas most digits with two wrong lights can't get classified well and can therefore be considered as noise.

4 Validation of results with two other databases

To validate the results of the previous section we applied our modifications to two additional datasets: the iris and the waveform dataset. We chose the iris dataset as a hard dataset, where boosting didn't work well in previous studies, the waveform dataset was chosen as an example, where boosting improved the base classifier with some space for improvement remaining by comparison with the Bayes classifier ([4, 6]). We estimated the error by 10-fold crossvalidation for the iris dataset and by the use of a test set for the waveform dataset with 1000 cases in the learning and 4000 cases in the test set.

For both datasets AdaBoost.M2 again concentrated on some groups more than on others (Table 5).

Table 5. Analysis of algorithm: Difference Δp between the group proportions at the last and the first boosting round in percent

	group 1	group 2	group 3
Δp for iris	-23.7	12.4	11.3
Δp for waveform	7.3	1.0	-8.3

With a similar calculation as in 3.1 the final classifier can be interpreted as an additive model

$$\text{score}(g)(x) = \sum_j f(x_j, g)$$

with step functions $f(x_j, g)$. An additive model seems to be well suited for both datasets, because the algorithm showed a good performance for both datasets and is more than competitive to AdaBoost.M1 using classification trees (Table 6).

Table 6. Performance of AdaBoost.M2

	iris	waveform
Final error	4,0	16,4
Minimal training error	2,0	13,3
Final training error	2,3	13,6
Overfit	0,3	0,3
Generalization error	1,7	2,8

In the previous chapter we examined two modified boosting algorithms that held the group proportions constant. For the digit dataset AdaBoostπ was clearly superior to AdaBoost.M2. A look on the results for the iris and waveform datasets show that both modifications seem to worsen the algorithm for both datasets, but there is one thing that makes the comparison difficult. In ([1]) the problem of rounding errors at extreme differences between weights of cases was addressed, so for every boosting round we saved the ratio r between the maximal and the minimal weight for all boosting rounds. For the modified algorithms r was extremely high (about 10^{12}) for two of the iris trials which lead to much greater error for these two trials, r was also higher for the waveform trial (1695 and 5767 for AdaBoostπ and AdaBoost$\hat{\pi}$ compared to 135 for AdaBoost.M2) but had similar values for the digit dataset (28, 27, 38). We didn't expect this behaviour of the algorithm, in contrast we even saw the modification as a regularization step. As a result the validation confirmed that the weight distribution should be at least watched during boosting and that further research in this direction is needed.

5 Conclusion and future work

The final classifier of AdaBoost.M2 using decision stumps as base classifiers can be shown and interpreted very well as an additive model without interactions for the digit example. The performance of the algorithm for the digit example suffers from attribute noise leading to error rates far from the Bayes error, although the final model is complex enough to handle the problem. The additional

error (8.29%) comes from the lack of fit for the training data (2.65%), overfitting (3.93%) and generalization (1.71%). During boosting the algorithm concentrates on the two groups 1 and 7. Holding group proportions constant on the can improve the algorithm considerably, but exact knowledge of the group proportions is crucial for the success of this method, which reduces the minimal training error and also errors resulting from overfitting and generalization.

In experiments with the iris and the waveform datasets the algorithm also concentrates on certain groups. The resulting classifiers are at least as good as classifiers resulting from AdaBoost.M1 with classification trees as base classifiers. As an additional advantage the resulting model is additive and therefore easier to interpret. Experiments to validate the good performance of the modified algorithm AdaBoostπ show no clear results but addressed the need to at least check the weight distribution during boosting. Additional work must be done in this area.

Future work will explore methods to take noise into account on one hand by changes within the algorithm and on the other hand by an additional analysis of the output of the algorithm. It will then be necessary to generalize the results to other algorithms, base classifiers and data sets.

References

1. E. Bauer, R. Kohavi, 1999. An empirical comparison of voting classification algorithms: bagging, boosting and variants. Machine Learning 36, 105-139.
2. L.Breiman, J.Friedman, R.Olshen, C.Stone, 1984. Classification and regression trees. Belmont, CA: Wadsworth.
3. T.G.Dietterrich, G.Bakiri, 1995. Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research 2, 263-286.
4. T.G.Dietterrich, 2000. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. Machine Learning 40,139-157.
5. Y.Freund, R.E.Schapire, 1997. A decision-theoretic generalization of online-learning and an application to boosting. Journal of Computer and System Sciences 55 (1), 119-139.
6. J.R.Quinlan, 1996. Bagging, boosting, and C4.5. Proceedings of the Thirteenth National Conference on Artificial Intelligence, 725-730.
7. G.Rätsch, T.Onoda, K.-R.Müller, 2000. Soft margins for AdaBoost. Machine Learning 42(3), 287-320.
8. R.E.Schapire, 1997. Using output codes to boost multiclass learning problems. Machine Learning: Proceedings of the Fourteenth International Conference, 313-321.
9. R.E.Schapire, Y.Freund, P.Bartlett, S.L.Wee, 1998. Boosting the margin: a new explanation for the effectiveness of voting methods. Annals of Statistics 26 (5), 1651-1686.
10. R.E. Schapire, Y.Singer, 1999. Improved boosting algorithms using confidence-rated predictions. Machine Learning 37, 297-336.