Regularizing AdaBoost

to prevent overfitting on label noise

Dirk W. J. Meijer

TUDelft

Regularizing AdaBoost

to prevent overfitting on label noise

by

Dirk W. J. Meijer

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 29 2016, at 14:00.

Student number:	1369326	
Project duration:	September 21, 2015 – June 30, 2016	
Thesis committee:	Prof. M. J. T. Reinders,	TU Delft
	Dr. D. M. J. Tax,	TU Delft, supervisor
	Dr. M. Loog,	TU Delft
	Dr. M. T. J. Spaan,	TU Delft

Acknowledgements

This project has been a source of joy and occasionally frustration, but I will let the result speak for itself. I owe a lot to David, for his advise, encouragement and questions, but especially for his contagious enthousiasm. I could not have done any of this without Elja, who colors my life and inspires me in more ways than she knows. Lastly, I thank anyone who provided me with coffee or listened to my ramblings, which should cover everyone else.

Contents

Introduction

In modern pattern recognition we often do not look for a single best classifier for a specific problem, but an ensemble that performs best. Combining multiple classifiers often leads to better results than a single classifier could. We might train several weak classifiers, train classifiers on a subset of the data or a subset of the dimensionality, initialize complex classifiers such as SVMs with different parameters and combine the resulting classifiers. Many more cases exist where combining is beneficial over a single classifier. Methods that combine multiple classifiers are also called *ensemble methods*.

Boosting relies on many weak classifiers and combining these into a single high-accuracy classifier [29]. Building a high-accuracy classifier is often difficult, while it is easy to come up with many weak, "slightly better than random" classifiers. The AdaBoost algorithm by Freund and Schapire [11] is an ensemble method that combines a large amount of weak learners to "arbitrary accuracy". AdaBoost struggles however, when the training set contains mislabeled objects, because it tries to accommodate these objects too much.

In this work, we present several intuitive adjustments to the AdaBoost algorithm and inspect the influence of one of these adjustments on classification performance. Chapter 1 summarizes the current results from literature. Chapter 2 introduces AdaBoost and its limitations. Chapter 3 introduces the proposed changes to the AdaBoost algorithm. Chapter 4 sets the stage for our experiments and presents the results of said experiments. Chapter 5 considers an abstraction of the ideas introduced in chapter 3 and looks at other domains where we might implement similar regularizations. Finally, chapter 6 will summarize this work, discuss the results and their implications and discuss possible continuations of our research.

Portions of this work have been incorporated in a research paper, which is set to be published in the proceedings of the International Conference on Pattern Recognition (ICPR) 2016. This paper can be found in the appendix.

1 Related Work

1.1. Classifier Combining

At the top level, we can divide classifier combining into two cases, classifier selection and classifier fusion [35]. In classifier selection the base classifiers are complementary, each classifier is an expert for a small domain, and the combiner selects which expert is most likely to be correct for each test object. Classifier fusion on the other hand, has all base classifiers trained on the full domain, making them competitive classifiers. The classifier output is now aggregated into a final output for the classification system.

To combine classifier outputs, we should know what these outputs look like. We expect one of four possible outputs [15]:

1. A c-element vector, where c is the number of classes. The elements in this vector may be posterior probabilities, continuous outputs or crisp outputs. They might add up to one, and they might not. In general, we call this the *supports* for each class.
2. Ranked class labels, in order of most to least likely.
3. A single class label, of the class that is deemed most likely.
4. Oracle output, whether an object was correctly classified or not. This is slightly artificial, but often used in literature [15].

A vector of scores is the best case, this gives us by far the most information. The vector of scores can be converted to ranked class labels, typically trough sorting the list by membership in each class. Label outputs are already more restrictive, and are suitable for voting rules at best (typically Borda count or various other methods for a ranked list, and majority vote for a single label output). The oracle output is a special case, of course we only have this output for labeled datasets. Nevertheless, it can be useful for classifier selection, imagine we know that objects similar to a new object were correctly classified by a certain classifier, then this might be a good choice for classifier selection. The type of output we get is dependent on both the classifier and the implementation.

For each object x we can now create a decision profile $DP(x)$ [17]. This decision profile is basically a matrix of size $L \times c$, with c the number of classes and L the number of base classifiers. $DP_{ij}(x)$ contains the support that classifier i gives to class j for object x. The final classification system is an *aggregation rule* on the decision profile. This aggregation rule can be untrained (examples include majority vote, maximum or minimum rule, mean or product of supports) or trained (examples include Naive Bayes [37] and traditional classifiers). The aggregation rule is either class-conscious, meaning that it uses only the scores for a certain class to calculate the final scores for that class, or class-indifferent [17].

When the aggregation rule is a classifier, Kuncheva et al. [17] argue that traditional classifiers might not be the best choice, since the distribution of objects in the new *intermediate feature space* is not very well-behaved. Duin et al. [8] show that the Nearest Neighbor classifier performs well and robust as a combiner, while it often shows a large bias when used as a base classifier. This method is prone to overfitting [7], and the base classifiers should either be weak classifiers, or the combiner should be trained on different training data than the classifiers themselves. Dietrich et al. [5] experiment with different amounts of overlap between the training sets for the base classifiers and the combiner, and show improved results with a 50% overlap over completely disjoint training sets. This might be attributed to the increase in size for the training sets, because these results do not show in their synthetic experiment.

1.2. Ensemble Diversity

In classifier design, the goal is often to minimize a combination of bias and variance. Ensemble methods should do exactly this, high-bias classifiers might be combined into a single low-bias classifier, or high-variance classifiers can be combined into a single low-variance classifier. Ensemble diversity is an important aspect of ensemble methods [4, 16, 18], and can be seen as the variance between base classifiers. We want a large amount of diversity, because base classifiers that are too similar will not improve one another when combined.

Cunningham and Carney [4] use entropy as a diversity measure for ensemble creation, and show that increased entropy of the base classifiers correlates well with the conditional entropy loss function and the accuracy on an unpublished medical dataset. Kuncheva et al. show in [16] however that ensemble diversity is *not* a good predictor of ensemble accuracy. They find “no clear relationship between diversity and [...] accuracy” but maintain that diversity still plays a role in classifier ensembles.

Of special interest is the AdaBoost algorithm by Freund and Schapire [11], this ensemble method trains classifiers sequentially. After a classifier has been trained, the algorithm reweights or resamples the data in such a way that the subsequent classifier is more likely to correctly classify objects that were incorrectly classified before. This *enforces* diversity among the base classifiers and makes the training error converge quickly.

Brown and Kuncheva [2] remark that ‘diversity’ is not well-defined and suggest that the choice of diversity measure should depend on the used error function and aggregation rule in the ensemble classifier design. A practically infinite number of diversity/similarity measures are available, and there is no consensus on when to use which one. [16] shows that the most commonly used ones are highly intercorrelated.

1.3. Dissimilarity Representation

An approach to pattern recognition that fundamentally differs from the feature-space approach is the dissimilarity representation. We create a dissimilarity matrix D of size $n \times n$ for n objects and determine a dissimilarity measure which tells us how *different* two objects are. Cell D_{ij} contains the dissimilarity between object i and j. The dissimilarity matrix is always symmetrical and has zeroes on all the diagonals. The dissimilarity measure used is often non-Euclidean or non-metric. If it were not, we could express the dissimilarity as a distance and use an ordinary feature-space.

Several methods of classification are possible [25], such as calculating the dissimilarities for a new object and selecting the nearest neighbor, using the dissimilarities of chosen prototype objects as features for a traditional feature-based classifier, or decision rules based on the relations that arise from the dissimilarity matrix.

Pękalska et al. introduce Classifier Projection Space in [24], later called *ensemble maps* in [15]. The novel idea here is to treat the pairwise diversity of classifiers in an ensemble as dissimilarities. Since these diversities are based on classifier *outputs*, we can also compare aggregation rules of ensembles and even the true labels of the objects. These dissimilarities are mapped onto a two-dimensional space by a Sammon mapping [26] for visual inspection. The images show how the ensemble relates (in terms of *similarity*) to its base classifiers and to the true labels. A qualitative connection might be drawn between distance to the true labels and classification error, though we should be very careful with this.

1.4. AdaBoost

1.4.1. Loss Function and Outliers

AdaBoost [11] in its design, minimizes exponential loss. As a result, if a training object is misclassified by a lot of base classifiers, its contribution to the error will be very large, and its weight will become very large as well. [29] mentions that this is a way to identify outliers in the dataset but acknowledges its detrimental effect on the performance of AdaBoost. Friedman, Hastie and Tibshirani [12] introduce “Gentle AdaBoost”, which is less prone to outliers and label noise because the instance weights increase more gently.

Wyner [36] argues that exponential loss is an odd choice, and we would expect the algorithm to overtrain, increasing in error when run for too many iterations. A possible reason for AdaBoost’s relative resilience to overtraining might be the use of weak base classifiers: as each base classifier is too weak to overfit on outliers and difficult objects, this may be reflected in the final classifier.

Arguably one of the most important theoretical aspects of AdaBoost is that the training error of the final classifier is bounded by a function of the training error of the base classifiers. This property stems from the use of the multiplication rule to update the weights and the use of a convex loss-function. This is true for both AdaBoost’s original design with the exponential loss function and LogitBoost’s use of the logistic loss function.

Long and Servedio [20] show that boosting algorithms based on convex loss functions do not tolerate even a low level of label noise, and neither regularization nor early termination will help. Mason et al. [22] show that boosting can be solved by performing a gradient descent in function space and provide a framework they call AnyBoost, in which a loss function of choice can be substituted. After proving that any convex function has a bounded training error, they introduce DOOM II, their own algorithm with a non-convex, sigmoidal shaped loss function and show it performs favorably compared to AdaBoost, because it is willing to ‘give up’ on instances that are too difficult, i.e. instances that would receive extremely large weights a with convex loss function.

Masnadi-Shirazi and Vasconcelos [21] argue that a better alternative is to specify the conditional risk first and later assess what loss function this minimizes. They show that their algorithm SavageBoost, which uses a non-convex loss function similar to a sigmoidal function, performs well in comparison to other variants on AdaBoost, especially when a large number of training objects are outliers. The use of a non-convex loss function also means SavageBoost no longer has the same error bounds that other variants of AdaBoost have. Performance is very similar to Real AdaBoost [30], taking the edge slightly in cases with high amounts of label noise.

In the end Wyner [36] concludes that while different error functions might seem like a superior choice, in practice the different error functions proposed and inspected in their work perform similar. Their general conclusion is that loss minimization cannot be the only secret to AdaBoost’s success.

The PAC (probably approximately correct) learning framework was introduced by Leslie Valiant in [33] to formalize the field of machine learning for mathematical analysis. In [6] Duffy & Helmbold summarize the definition of boosting in the PAC learning framework as follows:

Definition: A PAC boosting algorithm is a generic algorithm which can leverage any weak PAC learner to meet the strong PAC learning criteria.

They then go on to show that AdaBoost and LogitBoost are in fact boosting algorithms according to the PAC learning definition, whereas DOOM II, which uses a non-convex potential function, is not. This does not mean that DOOM II does a worse job at classification than AdaBoost, it simply means that we cannot theoretically prove that it will improve performance to ‘arbitrary accuracy’.

1.4.2. Error Estimation and Overtraining

Friedman et al. [12] and Grove & Schuurmans [13] argue that AdaBoost is prone to overtrain when it is run for too many iterations . This stems from the fact that AdaBoost directly minimizes the *training error*. It can be argued that in case of overtraining, the base classifiers are not *weak* enough for the data. This relates back to [7], where overfitting for classifier combiners is considered: either the base classifiers should be weak classifiers, or different data should be used by the combiner.

In the case of AdaBoost, this would mean that we would need a different training set (unseen by the base classifiers) to determine the classifier weights and (indirectly) the object weights for the objects in the first training set. Another approach could be to only interfere with AdaBoost after training is complete, by either recalculating the weights for the weighted majority vote on a different dataset, using a different classifier combiner altogether, or even using a second layer classifier that takes the base classifiers generated by AdaBoost as input. Using such an approach to avoid overtraining, we can even look into using strong classifiers as base classifiers, since the overfitting of the base classifiers can be corrected for by the combiner. These methods decouple the relation between instance weights and classifier weights as originally described by Freund and Schapire in [11], so again we lose the error bounds described in their original design.

Assessing the error of a base classifier is not the only step in AdaBoost where we focus on the training error, at each sequential step the weights of the training instances are increased or decreased, based on whether they were correctly or incorrectly classified by the previous base classifier. Reweighting instances in a training set based on performance on a *different* training set is less trivial than simply using the output of AdaBoost in a different way. And again, the weights are used in proving the bounds to the training error, meaning we will lose this theoretical bound and have to rely on empirical results.

Whether AdaBoost actually does overtrain is heavily debated and Mease and Wyner[23] think otherwise. They use artificial datasets to show that supposed situations in which AdaBoost ‘should’ overtrain, it actually does not. They also show that stumps might be more prone to overtraining than deeper decision trees, which is counterintuitive. They suggest that overtraining is likely to occur in low-dimensional cases only. In the end, it is clear that AdaBoost *might* overtrain, while it might not on some other data.

Bylander and Tate [3] divide the training set into two sets, AdaBoost is performed on the first and second set separately, and the error and weights are calculated for the classifiers trained on the *other* set. They average the classifiers weights obtained using the training and validation error, thus they obtain twice as many classifiers, trained on half the data. This method stays very close to AdaBoost, while trying not to violate the principles that keep classifier combiners from overtraining. They show very promising results, especially when using more sophisticated base classifiers. This is expected, because a more complex base classifier is expected to overfit more.

Torres-Sospedra et al. [32] go a step further and add cross-validation to AdaBoost. They take as many folds as classifiers and each classifier is tested on its corresponding validation fold. This limits AdaBoost in the sense that it limits the number of base classifiers, based on the size of the datasets (as we do not want the validation fold to be too small). Interestingly, they obtain better results by simply combining their base classifiers with the average combiner aggregation rule, as opposed to AdaBoost’s standard combiner, weighted majority vote.

2 AdaBoost

AdaBoost is a linear model, at every iteration a new term is added to the ensemble of classifiers, as shown in equation 2.1.

$$H_T(x) = \alpha_T h_T(x) + \sum_{t=1}^{T-1} \alpha_t h_t(x) \tag{2.1}$$

The loss is chosen to be the exponential loss,

$$\begin{align} L &= \sum_{i=1}^{N} \exp(-y_i H_T(x_i)) \tag{2.2}\\ &= \sum_{i=1}^{N} \exp\left(-y_i \left[\alpha_T h_T(x_i) + \sum_{t=1}^{T-1} \alpha_t h_t(x_i)\right]\right) \tag{2.3}\\ &= \sum_{i=1}^{N} \exp\left(-y_i \alpha_T h_T(x_i)\right) \exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t h_t(x_i)\right) \tag{2.4}\end{align}$$

Where N is the number of objects. Note that in the simplest classification case, $Y = \{-1, 1\}$, $h_t(x) \in Y$ and $y_i \in Y$ will be in this domain, but $H_T(x) \in \mathbb{R}$. The magnitude of $H_T(x)$ can be interpreted as a confidence of assigning an object to class $\operatorname{sign}(H_T(x)) \in Y$.

To optimize every α_t and $h_t(x)$ at the same time would be problematic, so instead, we take every α_t for $t = 1 \ldots T - 1$ to be fixed and we optimize α_T iteratively. Since the second exponent in 2.4 is fixed and known for every value of T, we call this w_i. A distinction is now made between the object correctly and incorrectly classified by $h_T(x)$. The loss becomes

$$L = \sum_{i=1}^{N} w_i \exp\left(-y_i \alpha_T h_T(x_i)\right) \tag{2.5}$$

$$= \sum_{\{i: h_T(x_i)=y_i\}} w_i \exp(-\alpha_T) + \sum_{\{i: h_T(x_i)\neq y_i\}} w_i \exp(\alpha_T) \tag{2.6}$$

$$= \sum_{i=1}^{N} w_i \exp(-\alpha_T) - \sum_{\{i: h_T(x_i)\neq y_i\}} w_i \exp(-\alpha_T) + \sum_{\{i: h_T(x_i)\neq y_i\}} w_i \exp(\alpha_T) \tag{2.7}$$

$$= \sum_{i=1}^{N} w_i \exp(-\alpha_T) + \sum_{\{i: h_T(x_i)\neq y_i\}} w_i (\exp(\alpha_T) - \exp(-\alpha_T)) \tag{2.8}$$

The base classifier $h_T(x)$ that minimizes this loss should minimize the weighted error function

$$\epsilon_T = \sum_{\{i: h_T(x_i)\neq y_i\}} w_i \tag{2.9}$$

$$= \sum_{\{i: h_T(x_i)\neq y_i\}} \exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t h_t(x_i)\right) \tag{2.10}$$

$$w_i = \prod_{t=1}^{T-1} \exp(-\alpha_t y_i h_t(x_i)) \tag{2.11}$$

In other words, the object weights are multiplied by $\exp(\alpha_t)$ every time the objects are misclassified by classifier $h_t(x)$ and multiplied by $\exp(-\alpha_t)$ every time they are correctly classified. All that's left is to minimize with regard to α_T by setting the partial derivative to zero.

$$\frac{\partial L}{\partial \alpha_T} = \sum_{i=1}^{N} -w_i \exp(-\alpha_T) + \sum_{\{i: h_T(x_i)\neq y_i\}} w_i (\exp(\alpha_T) + \exp(-\alpha_T)) = 0 \tag{2.12}$$

$$\sum_{i=1}^{N} w_i = (\exp(2\alpha_T) + 1) \sum_{\{i: h_T(x_i)\neq y_i\}} w_i \tag{2.13}$$

$$\alpha_T = \frac{1}{2} \log\left(\frac{\sum_{i=1}^{N} w_i}{\sum_{\{i: h_T(x_i)\neq y_i\}} w_i} - 1\right) \tag{2.14}$$

$$= \frac{1}{2} \log\left(\frac{\sum_{i=1}^{N} w_i - \sum_{\{i: h_T(x_i)\neq y_i\}} w_i}{\sum_{\{i: h_T(x_i)\neq y_i\}} w_i}\right) \tag{2.15}$$

$$= \frac{1}{2} \log\left(\frac{\sum_{\{i: h_T(x_i)= y_i\}} w_i}{\sum_{\{i: h_T(x_i)\neq y_i\}} w_i}\right) \tag{2.16}$$

If we call ϵ_T the normalized weighted error for classifier $h_T(x)$, defined as

$$\epsilon_T = \frac{\sum_{\{i: h_T(x_i)\neq y_i\}} w_i}{\sum_{i=1}^{N} w_i} \tag{2.17}$$

Then we can rewrite α as

$$\alpha_T = \frac{1}{2} \log\left(\frac{1-\epsilon_T}{\epsilon_T}\right) \tag{2.18}$$

Algorithm 1: AdaBoost

```
Input: Dataset $\mathbf{x}$, consisting of $N$ objects $\langle x_1, \ldots, x_N \rangle \in X$ with labels $\langle y_1, \ldots, y_N \rangle \in Y = \{1, \ldots, c\}$
Input: Weak learning algorithm WeakLearn
Input: Number of iterations $T$
Initialize: Weight vector $w_i^1 = \frac{1}{N}$ for $i = 1, \ldots, N$
for $t = 1$ to $T$ do
    Call WeakLearn, providing it with weights $w_i^t$
    Get back hypothesis $h_t : X \to Y$
    Compute $\epsilon = \sum_{i=1}^{N} w_i^t [h_t(x_i) \neq y_i]$
    if $\epsilon > 1 - \frac{1}{c}$ then
        Set $\alpha_t = 0$
        Set $w_i^{t+1} = w_i^t$
    else
        Compute $\alpha_t = \log((1-\epsilon)/\epsilon) + \log(c-1)$
        Set $w_i^{t+1} = w_i^t \exp(\alpha_t [h_t(x_i) \neq y_i])$ for all $i$
        Normalize $w_i^{t+1}$
    end
end
Output: Hypothesis $H(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t [h_t(x) = y]$
```

To go from this two class derivation to an algorithm that works in a multi-class environment, we have to make some changes. The output domain is no longer $\{-1, 1\}$ but $\{1, 2, \ldots, c\}$, where c is the number of classes. Any time we would multiply $h_t(x)$ and y, we now instead have to check whether $h_t(x)$ predicts the correct class y. Another change is that if a base classifier is worse than random, we should not use its error to calculate α, as a negative weight does not have the same meaning here that it does in a two class case. In a two class case, a negative weight means a positive weight for the same classifier, with the output labels swapped. In the multi-class case, a negative vote in a weighted majority vote is not well defined. As a result, we only use a classifier if its performance is better than random.

Consider the multi-class AdaBoost shown in algorithm 1 with some small modifications. The error threshold has been changed from $\frac{1}{2}$ to $1 - \frac{1}{c}$. Freund and Schapire ([11]) assume a worst case scenario with $\frac{1}{2}$ and give an example of a three class case: suppose one class is easy to distinguish from the other two, but the remaining two classes are very difficult to distinguish. A base classifier will easily reach an error below $\frac{2}{3}$ because the first class is easy to distinguish from the others, but it might not be possible to boost this classifier to arbitrary accuracy, if the other two classes are indistinguishable to it. The problem arises from the fact that we want to use a decision stump classifier on multi-class problems, a decision stump on a 10 class problem (with equal priors) cannot have an error below 0.8, because it can only output 2 classes and will always misclassify the other 8 classes. In practice however, even if the base classifier cannot be boosted to *arbitrary accuracy*, boosting it in this way still reduces the error. The α is changed to scale with this change, as is shown by Zhu et al. in [38]. The α weights no longer have the factor $\frac{1}{2}$, but only the misclassified objects have their weights altered. Since the weights are normalized each iteration, the result is the same.

Note that if a hypothesis h_t is worse than a random guess, we do not halt the algorithm, but set its weight to zero and continue to the next iteration. This means for deterministic weak learners that after one base classifier gets a weight of zero, the subsequent ones will also get a weight of zero. Often however, we will use the weights simply to sample from the training set, meaning that a subsequent sampling could provide us with a weak learner that *will* have a better-than-random performance.

A problem arises when some of the training objects have the wrong label y_i. Firstly, such an object will probably be difficult for the weak base classifiers to correctly classify. Because of this, it will be misclassified by a lot of the base classifiers, causes its weight to increase rapidly. As its weight increases, classifiers will be trained to "correctly" classify this object (in quotes, because the correct class is not actually known to the algorithm). This leads to poor base classifiers, and in the end to a classifier ensemble that is overtrained on these mislabeled objects. The adaptive property of AdaBoost becomes a sensitivity to noise, as the algorithm tries to accommodate the noise.

3 Proposed Changes

In this chapter, three different alterations to the AdaBoost algorithm are presented. Of these three, ValidBoost showed the most promise in our initial experiments, and it is the only one that will be discussed in the rest of this work.

3.1. ValidBoost

To avoid overfitting in AdaBoost, methods like LogitBoost [12] and DOOM II [22] have tried regularizing the weights assigned to objects. Our approach differs, in that we intervene at an earlier step, the computation of the error in equation (2.17). This error calculation is actively minimizing the training error, which we expect leads to overtraining. More specifically, the error is underestimated because the estimator is biased, the generalization error is taken to be the error on which the classifier was trained.

Hypothesis 1: *AdaBoost overtrains (in part) because it uses a biased estimator to calculate the error a base classifier makes.*

To test this hypothesis, we perform the following simple experiment. In a simple, two dimensional, banana-shaped, two class case, we have a training set with label noise (figure 1), a test set without label noise and a very large validation set without label noise. We compare AdaBoost to a modified method, which uses not the training set, but the validation set to estimate α. In this way, we simulate the error estimator to be unbiased. Because the validation set is very large, in a 2D, 2 class domain, we expect it to basically contain the distribution, and the error calculated on it to be close enough to the actual error made on the domain. To stay as close as possible to vanilla AdaBoost, we reweight the objects in the validation set when they are misclassified, and use the weighted error to determine α. Weights for the training set are still calculated and used for training. Figure 2 shows the averaged results of 20 trials with 10% label noise on the training set. A few interesting things should be remarked. Firstly, AdaBoost definitely starts to overfit, at around 30 iterations. Secondly, our new changes to the error estimation prevent AdaBoost from overfitting. Thirdly, unfortunately, this method does not reach an error as low as AdaBoost does, before it starts overfitting. The results are similar and the observations the same for all levels of label noise (though the overfitting is barely noticeable below 5% label noise).

The reason for the unfortunate early convergence of this method has the same reason as the overfitting, the presence of label noise. The noisy objects in the training set will be misclassified in the first few iterations, increasing their weight. After a while, the weights of the noisy objects will be so large that they will be the main focus of the base classifier. This is the point where AdaBoost would usually start to overfit, but in this case, the (weighted) error on the validation set will exceed $1/2$ and every base classifier from this point on will receive a weight of $\alpha = 0$.

This small experiment is the basis of the changes we propose. But in a real world supervised learning situation, we will not have access to a large validation set, unless we use a holdout sample from the

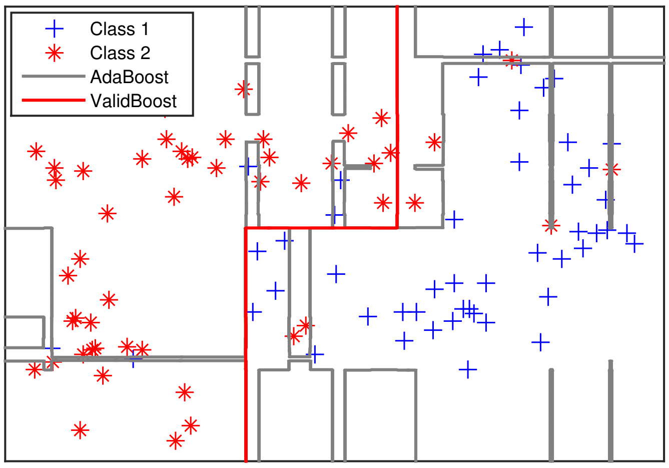

Figure 1: Banana-shaped dataset with 10% added label noise.

trainings set. When using a holdout sample, we cannot expect it to be noise free either. But this allows us to sacrifice some of the accuracy of the classifier (by training it on a smaller set) in exchange for a better estimate of the accuracy of the classifier (by estimation this on the holdout sample). Luckily, because we typically use weak classifiers in boosting, we expect a classifier trained on a slightly smaller set to be very similar to a classifier trained on the full set.

We perform the holdout experiment with the same parameters, 10% noise, 20 trials. The original training set is 200 objects, divided 50/50 into training and holdout set. The results are shown in figure 3. It is very clear that these results are nowhere near as good as when we had a noise-free validation set.

Bylander and Tate [3] took a similar route by dividing the training set into two sets before starting, and performing AdaBoost twice. The final classifier weights are then calculated from the averaged errors on the original classifier weights calculated by AdaBoost, and the errors as calculated on the validation set.

Averaging the errors to calculate the weights is an important step here. Imagine a dataset with outliers, when we take a holdout sample as a validation set, each outlier is either in the training set or in the validation set. If it is in the training set, the classifier will suffer and as a result the assigned weight will be low and the classifier will not contribute much to the ensemble. If it is in the validation set, the classifier will be good, but the error will be overestimated because we want an outlier to be correctly classified by a classifier that was not trained on it. As a result, it too will obtain a low weight and also contribute less to the ensemble than it should. The first case is less troubling, because the use of weak base classifiers ensures its impact is limited. In the second case however, we will overestimate the error. As the error estimated on the training is an underestimation of the true error, averaging should hopefully give us a better estimation of the error than either the training set or the validation set would.

But this is only one of multiple possible options, none of which are inherently better than the others. We could use only the validation error, only the training error, or a weighted combination. Using cross validation to estimate the error made by a learner on a training set gives us a slight overestimation [10], Efron and Tibshirani suggest offsetting this upward bias by averaging with the downward biased training error, as shown in equation 3.1.

$$\epsilon = \tau\epsilon_v + (1-\tau)\epsilon_t \tag{3.1}$$

In bootstrapping it is common to set $\tau = 0.632$, or $1-e^{-1}$, because this is the average fraction of objects selected at least once in a bootstrap sample, this is known as the ".632 rule," also introduced by Efron [9]. It should be noted that we do not suffer the same problem this method was introduced for, the .632 rule is used to unbias the cross validation estimate of the true error *when the entire training set is used.*

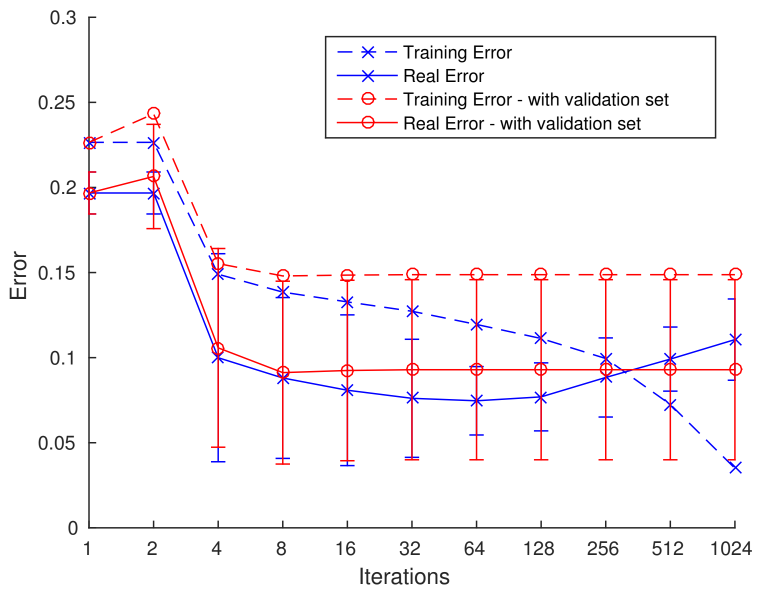

Figure 2: Training and test errors on a banana-shaped dataset with 10% label noise, for AdaBoost and a method using a validation set to estimate the classifier weights.

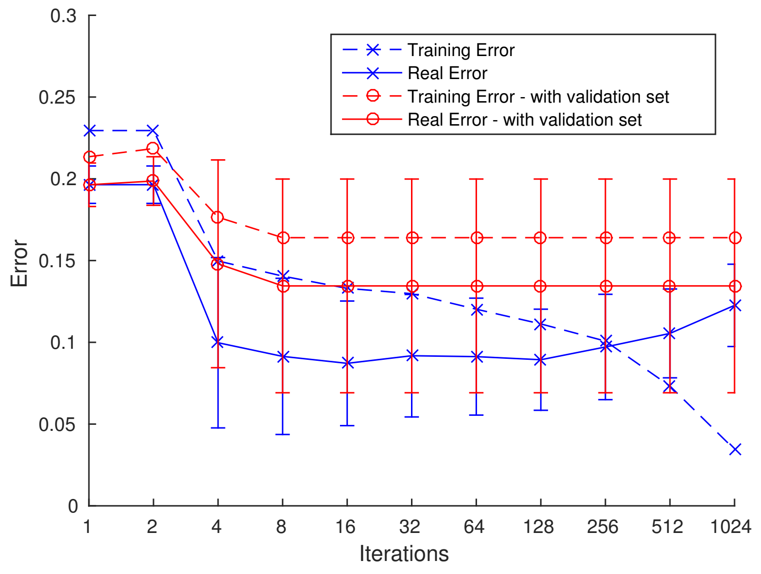

Figure 3: Training and test errors on a banana-shaped dataset with 10% label noise, for AdaBoost and a method using a holdout sample to estimate the classifier weights.

But the train of thought is interesting and perhaps we should use this in another way.

Since AdaBoost starts out very good before it starts overfitting, perhaps we can leverage this to avoid overfitting. We want to vary τ between 0 and 1 over the course of the runtime of the algorithm. We set τ equal to

$$\tau = \frac{\log t}{\log T} \tag{3.2}$$

where t is the current iteration and T is the total number of iterations. In this way we aim to take advantage of AdaBoost's superior accuracy, while increasing τ over time to prevent overfitting. The rationale for choosing a logarithmic increase is as follows: AdaBoost trains a classifier ensemble, adding a new

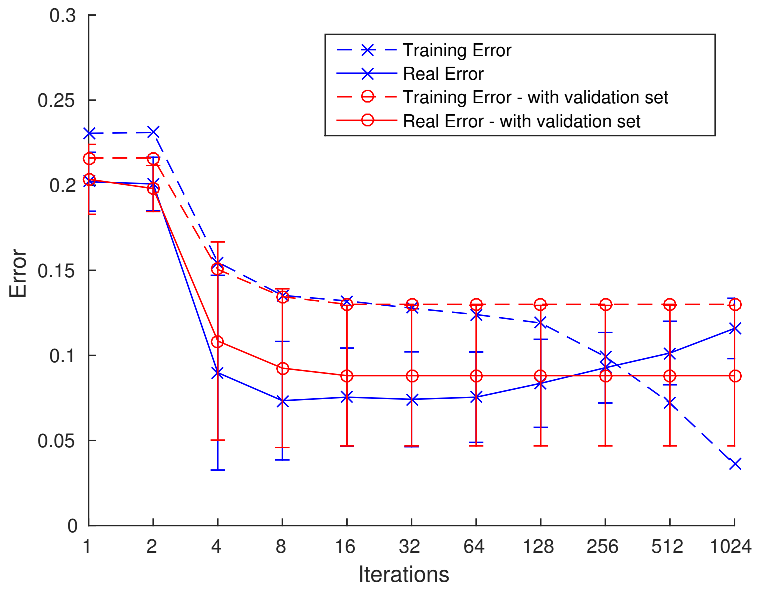

Figure 4: Training and test errors on a banana-shaped dataset with 10% label noise, for AdaBoost and a method using both the training set and a holdout sample in logarithmically varying proportion to estimate the classifier weights.

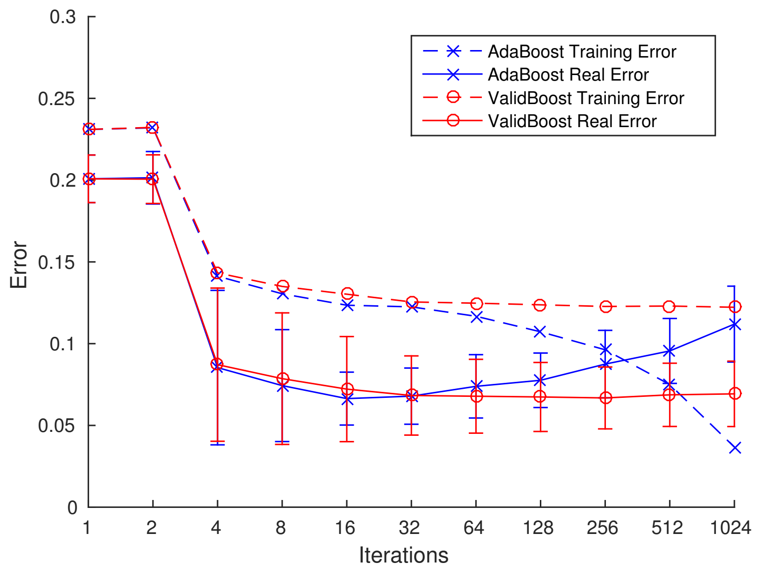

Figure 5: Training and test errors on a banana-shaped dataset with 10% label noise, for AdaBoost and our method using a holdout sample of logarithmically increasing size.

base classifier at every iteration. This means that the “impact” that every new classifier has on the growing ensemble is $1/t$. This explains why AdaBoost initially converges quickly and later iterations have less impact. The initial steps of AdaBoost are by far the most important, with every iteration, one classifier is added to an ensemble of ever increasing size. To reflect this in our choice of τ, we recognize that not the absolute iteration number is a good indicator of what stage of the process we are in, but instead the logarithm of the iteration number tells us more. Figure 4 shows the results of setting τ in this way.

In the initial steps of the algorithm, the holdout sample will have very little influence and using a holdout sample of 50% of the data is something of a waste of perfectly good training data. Following this rationale, we decide the size of the holdout sample to scale linearly with τ, starting at an empty set and

Algorithm 2: ValidBoost

```
Input: Dataset $\mathbf{x}$, consisting of $N$ objects $\langle x_1, \dots, x_N \rangle \in X$ with labels $\langle y_1, \dots, y_N \rangle \in Y = \{1, \dots, c\}$
Input: Weak learning algorithm WeakLearn
Input: Number of iterations $T$
Initialize: Weight vector $w_i^1 = \frac{1}{N}$ for $i = 1, \dots, N$
for $t = 1$ to $T$ do
    Set $\tau = \frac{\log t}{\log T}$
    Split $\mathbf{x}$ into $\mathbf{x}_v$ of size $\tau N/2$ and $\mathbf{x}_t$
    Call WeakLearn on $\mathbf{x}_t$, providing it with weights $w_i^t$ for $\{i : x_i \in \mathbf{x}_t\}$
    Get back hypothesis $h_t : X \to Y$
    Compute $\epsilon_v = \sum_{\{i:x_i \in \mathbf{x}_v\}} w_i^t [h_t(x_i) \neq y_i]$
    Compute $\epsilon_t = \sum_{\{i:x_i \in \mathbf{x}_t\}} w_i^t [h_t(x_i) \neq y_i]$
    Set $\epsilon = \tau\epsilon_v + (1 - \tau)\epsilon_t$
    if $\epsilon > 1 - \frac{1}{c}$ then
        Set $\alpha_t = 0$
        Set $w_i^{t+1} = w_i^t$
    else
        Compute $\alpha_t = \log((1 - \epsilon)/\epsilon) + \log(c - 1)$
        Set $w_i^{t+1} = w_i^t \exp(\alpha_t [h_t(x_i) \neq y_i])$ for all $i$
        Normalize $w_i^{t+1}$
    end
end
Output: Hypothesis $H(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t [h_t(x) = y]$
```

reaching 50% of the entire data when $\tau = 1$ at $t = T$. We also sample a new holdout set from the training set at every iteration, it does not increase incrementally.

Figure 5 shows us very promising results on the same dataset, with all our new changes in place. The final algorithm is presented in Algorithm 2.

3.2. Cross-Validated AdaBoost

Another alteration of AdaBoost we investigated integrated the use of k-fold cross-validation into the algorithm. The difference between our algorithm and AdaBoost is that the body of the loop is essentially executed k times. At the start of every iteration the training set is split into k folds of equal size. We first train our base classifier on all data expect for the validation fold, and assess the error the classifier made by testing on the validation fold. The classifier and object weights are set from this validation error. In this way, we are no longer dependent on the training error, but use the cross-validation error to get a better estimate of the true error that the new base classifier makes.

The error calculation requires a weighting factor, it is divided by the sum of the weights in the validation fold, so that total misclassification receives an error of 1. This ensures that we don't underestimate our cross-validation error, because the weights of a single fold will not add up to 1.

A base classifier trained on $k-1$ folds will still minimize the same loss function, but with less information. This can be seen as a regularization. Likewise, the obtained error ϵ has been calculated with less information and is now an estimation of the true error made by the base classifier. The same principles that underlie AdaBoost are the base of our new algorithm. The difference is that $h_T(x)$, w_i and α_T are

Algorithm 3: Cross-Validated AdaBoost

```
Input: Dataset $\mathbf{x}$, consisting of $N$ objects $\langle x_1, \dots, x_N \rangle \in X$ with labels $\langle y_1, \dots, y_N \rangle \in Y = \{1, \dots, c\}$
Input: Weak learning algorithm WeakLearn
Input: Number of iterations $T$
Input: Number of folds $k$
Initialize: Weight vector $w_i^1 = \frac{1}{N}$ for $i = 1, \dots, N$
for $t = 1$ to $T$ do
    Split $\mathbf{x}$ into $k$ sets of equal size and with equal prior probabilities, $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$
    for $f = 1$ to $k$ do
        Set the validation set $\mathbf{x}_v$ to $\mathbf{x}_f$
        Set the training set $\mathbf{x}_t$ to $\bigcup_{i \neq f} \mathbf{x}_i$
        Call WeakLearn on $\mathbf{x}_t$, providing it with weights $w_i^t$ for $\{i : x_i \in \mathbf{x}_t\}$
        Get back hypothesis $h_{t,f} : X \to Y$
        Compute $\epsilon_v = \sum_{\{i : x_i \in \mathbf{x}_v\}} w_i^t [h_{t,f}(x_i) \neq y_i]$
        Compute $\epsilon_t = \sum_{\{i : x_i \in \mathbf{x}_t\}} w_i^t [h_{t,f}(x_i) \neq y_i]$
        Set $\epsilon = (\epsilon_v + \epsilon_t)/2$
        if $\epsilon > 1 - \frac{1}{c}$ then
            Set $\alpha_{t,f} = 0$
            Set $w_i^{t+1} = w_i^t$ for all $i$
        else
            Compute $\alpha_{t,f} = \log((1-\epsilon)/\epsilon) + \log(c-1)$
            Set $w_i^{t+1} = w_i^t \exp(\alpha_{t,f} [h_{t,f}(x_i) \neq y_i])$ for all $i$
            Normalize $w_i^{t+1}$
        end
    end
end
Output: Hypothesis $H(x) = \arg\max_{y \in Y} \sum_{f=1}^{k} \sum_{t=1}^{T} \alpha_{t,f} [h_{t,f}(x) = y]$
```

no longer optimized directly, but estimated from partial data. In the end, similar to Bylander and Tate's use of validation sets [3], we have trained kT base classifiers, each on part of the training set. The time complexity is slightly less than k times AdaBoost's time complexity. This seemingly small change has some important implications. One of the strengths of the original AdaBoost algorithm is the fact that the training error is guaranteed to converge. This guarantee disappears now that we are no longer basing object weights on the training error. On the other hand, the change in calculation of the classifier weights is expected to be beneficial. Since the cross validation error is a better indicator of the test error than the training error is, the classifier weights should be better suited to minimize the test error. The training error might no longer approach zero and might be slower to decrease, but we expect this will not be detrimental to the test error.

A second difficulty arises from the calculation of the classifier weights. In AdaBoost, if ϵ turns out to be zero, α_t is given a value of positive infinity. This is not a problem, because apparently this classifier cannot be improved upon by boosting, so the weighted majority vote ensemble will be identical to this base classifier. Our cross-validation error however, is allowed to become zero for any given base classifier, but this doesn't mean it should give that base classifier a weight of positive infinity. So we need some kind of regularization to make sure all classifier weights are finite.

An obvious choice is to again average the validation error with the training error, since if these are both zero, a classifier weight of infinite is justified. This does go against our reason to use cross validation in the first place: to better estimate the performance of a base classifier and its importance in the ensemble. In practice, this solution would mean replacing the calculation of the classifier weights with a weighted sum. Like Bylander & Tate, we use take equal weights for both. The final algorithm is shown in Algorithm 3.

3.3. Recombined AdaBoost

Finally, we look at a different use of AdaBoost's output. We split the training set into a new reduced training set, and a holdout sample. The new training set is used by AdaBoost to train T classifiers. After that we use the holdout sample as a training set for a classifier combiner.

The suggested size of the holdout sample depends on the size of the training set and the combiner used, but usually between 0% (for an untrained combiner) and 50% is advised. If we make the holdout sample too small, the classifier combiner may overfit on the holdout sample, if we make the holdout sample too big, AdaBoost may overfit on the reduced training set. The algorithm is shown in Algorithm 4.

Algorithm 4: Recombined AdaBoost

```
Input: $N$ training objects $\langle x_1, ..., x_N \rangle$ with labels $y_i \in Y = \{1, ..., c\}$
Input: Trained classifier combiner ClassfCombiner
Input: Holdout parameter $\sigma \in [0, 1)$
Input: Number of iterations $T$
Randomly split the training set into two sets, $X_t$ and $X_h$, such that $|X_h| = \sigma N$
Call AdaBoost.M1 on $X_t$, for $T$ iterations
Get back $T$ hypotheses $h_t : X \to Y,\ t = 1, ..., T$
Train ClassfCombiner on all $h_t$ and $X_h$
Get back combiner hypothesis $H_c : Y^T \to Y$
Output: Hypothesis $H(x) = f(h_1(x), ..., h_T(x))$
```

Unless we use an exceptionally complex classifier combiner, this algorithm has the same complexity as the original AdaBoost algorithm. The long term results of this algorithm are unclear, we expect a trained combiner to perform better than AdaBoost's stock combiner, but this might be offset by the holdout sample we are not using for training. An interesting experiment could be to not use a holdout sample, but use the same training set both for AdaBoost and for the combiner. This will increase the likelihood of overfitting, but it will allow for more advanced classifier combining without reducing the training set size.

Kuncheva performs a similar experiment in [14], in which she specifically looks at her Decision Templates model, which functions akin to a nearest-mean classifier in the output space, but for different similarities as opposed to the Euclidean distance used by the nearest-mean classifier. Her results show that for the small number of iteration she inspects (up to 15), using a different combiner from AdaBoost is likely beneficial. In most cases AdaBoost will take a lot more than 15 iterations to converge, and this is hardly a fair comparison. We are more interested in the long term behavior of other combining methods. We expect these methods to converge a lot quicker than AdaBoost, because they will use the base classifiers trained by AdaBoost in a more efficient way, but this discrepancy may be made up for by AdaBoost in the long run.

It should be noted that this method may prove very prone to overfitting, as for every iteration of AdaBoost, we are increasing the dimensionality of the output space. The dimensionality in which we train the classifier combiner increases linearly with the number of iterations we use and the curse of dimensionality [34] dictates that this method may not provide accurate generalization at a higher number of iterations because of this.

4 Experiments

In order to assess the effectiveness of ValidBoost, we will compare the performance of AdaBoost, ValidBoost and related algorithms on several standard training sets from the UCI Machine Learning Repository [19]. The datasets are shown in table 1, note that objects with missing values have been removed.

Table 1: Datasets that will be used for performance comparisons.

Dataset	Objects	Dimensions	Classes
breast-cancer-wisconsin	699	9	2
ecoli	336	7	8
glass	214	9	4
heart-disease	297	13	2
ionosphere	351	34	2
iris	150	4	3
liver-disorders	345	6	2
pima-indians-diabetes	768	8	2
satimage	6435	36	6
sonar	208	60	2
soybean-large	266	35	15
soybean-small	136	35	4
wine	178	13	3

We will not only compare our proposed changes with AdaBoost, but also with other methods that have the same goal, preventing overfitting. Table 2 shows the algorithms we will be comparing. We are not only interested in if our method improves AdaBoost, but also in the quantitative improvement compared to other regularizations.

Table 2: Algorithms that will be compared.

Method	Description
AdaBoost	The AdaBoost.M1 algorithm by Freund and Schapire [11].
LogitBoost	The LogitBoost algorithm as described in [12].
Bylander & Tate	The proposed use of validation sets to improve AdaBoost in [3].
ValidBoost	The method described in section 3.1.

Since we are mostly interested in AdaBoost's tendency to overfit [13], the intolerance to label noise [20] and the use of more complex base classifiers, we will inspect performance of the algorithms when they are run for different numbers of iterations, with added label noise to the training sets, with different base classifiers. The performance of each method on each dataset will be assessed with 10-fold cross-validation, repeated 5 times to average out different samplings.

4.1. Results

The classification errors of the algorithms in table 2 on the datasets in table 1 are presented and discussed in this section. Tables 3, 4 and 5 show the results when using decision stumps as base classifiers, for a noise level of 0%, 10% and 20% respectively. Tables 6, 7 and 8 show the results when using decision trees with a (maximum) depth of 8 as base classifiers, for a noise level of 0%, 10% and 20% respectively. The lowest error for each dataset is bolded, as are the errors that are not significantly higher (determined by a paired sample T-test with a p-value threshold of 0.05).

Figures 6, 7 and 8 show the comparison of ValidBoost with the 3 other methods graphically, comparing the avarage error rates per dataset of ValidBoost on the vertical axis to the other relevant algorithm on the horizontal axis. Statistically significant differences are shown in red.

First off, we consider the datasets without added noise. When decision stumps are used, there does not seem to be a clear winner, all four algorithms perform very similar. (Except on the soybean-large dataset, which will be discussed momentarily.) When we switch to a more complex base classifier, the decision tree, ValidBoost outperforms all the other methods by a small margin. We assert that the base classifiers themselves are overfitting, which ValidBoost prevents by not using the entire dataset for the training. While this is certainly a visible and statistically significant improvement over the other methods, the difference is still small.

As expected, when label noise is added to the dataset, AdaBoost begins to struggle. LogitBoost's results are very similar to AdaBoost's, indicating that the change in loss function (logistic loss for LogitBoost instead of exponential loss for AdaBoost) is not enough to stop the boosting algorithm from overfitting. Both Bylander & Tate's method and ValidBoost are clear improvements over AdaBoost and LogitBoost. When we use decision stumps as base classifiers, Bylander & Tate and ValidBoost are very close in error rate, with Bylander & Tate having a small edge in a few cases. Using decision trees as base classifiers however, ValidBoost outperforms Bylander & Tate's method.

Two specific cases are of interest. The first is the behavior on the soybean-large dataset in the noiseless case with decision stumps, as shown in figure 9. We see that all four methods have very similar performance up to ~100 iterations. After this ValidBoost and Bylander & Tate's method continue improve while AdaBoost and LogitBoost seem to stagnate. After ~250 iterations, Bylander & Tate's method starts to overfit and ends up with an error rate similar to AdaBoost, while ValidBoost continues to decrease its error.

A similar effect is observed with the dataset soybean-small, with no added noise and using decision trees as base classifiers, as shown in figure 10. Looking at only AdaBoost, it seems that very little can be gained from boosting decision trees in this scenario, as the classifier ensemble remains unchanged after a few iterations. This indicates that after this point, the next classifier which would be found will have an error larger than the error threshold $(1 - 1/c)$ and will receive a weight of zero. Because the weights of the objects do not change, on the next iteration, the exact same classifier will be found, with the same error, also receiving a weight of zero. We observe that this also happens to LogitBoost and Bylander & Tate's method. ValidBoost however, keeps reducing its error rate. We attribute this behavior to the fact that at each interation, we take a different holdout sample from the trainig set. Because of this, if a base classifier has an error above the threshold, in the next iteration we will likely take a different holdout sample, and train a different classifier. This means that we always have a chance to add a useful classifier to the ensemble, even after the other methods have stagnated.

Table 3: Error - 0% noise, decision stumps

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	**0.04 (0.02)**	**0.04 (0.03)**	**0.04 (0.02)**	**0.05 (0.02)**
ecoli	**0.20 (0.07)**	**0.19 (0.05)**	**0.21 (0.05)**	**0.19 (0.06)**
glass	**0.34 (0.10)**	**0.35 (0.10)**	**0.34 (0.10)**	**0.39 (0.11)**
heart-disease	**0.19 (0.07)**	**0.20 (0.07)**	**0.17 (0.06)**	**0.17 (0.07)**
ionosphere	**0.07 (0.03)**	**0.08 (0.03)**	**0.07 (0.04)**	**0.08 (0.03)**
iris	**0.07 (0.05)**	**0.06 (0.04)**	**0.04 (0.04)**	**0.06 (0.05)**
liver-disorders	**0.29 (0.05)**	0.29 (0.05)	**0.28 (0.05)**	**0.26 (0.07)**
pima-indians-diabetes	**0.24 (0.04)**	**0.24 (0.04)**	**0.25 (0.04)**	**0.24 (0.04)**
satimage	0.23 (0.02)	0.22 (0.02)	**0.20 (0.02)**	0.21 (0.02)
sonar	**0.12 (0.09)**	**0.12 (0.08)**	0.20 (0.08)	0.16 (0.09)
soybean-large	0.56 (0.08)	0.62 (0.09)	0.61 (0.10)	**0.33 (0.06)**
soybean-small	**0.16 (0.09)**	**0.12 (0.08)**	0.21 (0.11)	0.17 (0.09)
wine	**0.06 (0.09)**	**0.06 (0.09)**	**0.07 (0.08)**	**0.03 (0.06)**

Table 4: Error - 10% noise, decision stumps

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	0.07 (0.03)	0.06 (0.03)	**0.06 (0.02)**	**0.05 (0.02)**
ecoli	0.26 (0.08)	**0.23 (0.06)**	**0.22 (0.07)**	**0.20 (0.09)**
glass	0.39 (0.10)	**0.38 (0.10)**	**0.34 (0.09)**	**0.35 (0.11)**
heart-disease	0.27 (0.08)	0.26 (0.09)	**0.18 (0.06)**	**0.19 (0.05)**
ionosphere	0.16 (0.05)	0.16 (0.05)	**0.11 (0.06)**	**0.11 (0.07)**
iris	**0.09 (0.10)**	**0.06 (0.08)**	**0.07 (0.07)**	**0.05 (0.06)**
liver-disorders	**0.32 (0.07)**	**0.31 (0.07)**	**0.29 (0.08)**	**0.30 (0.06)**
pima-indians-diabetes	**0.26 (0.04)**	0.26 (0.04)	**0.24 (0.04)**	**0.25 (0.04)**
satimage	**0.15 (0.01)**	**0.16 (0.01)**	0.17 (0.01)	0.18 (0.01)
sonar	0.27 (0.08)	0.27 (0.09)	**0.21 (0.07)**	**0.25 (0.12)**
soybean-large	0.34 (0.11)	**0.29 (0.09)**	**0.32 (0.09)**	0.38 (0.11)
soybean-small	0.29 (0.16)	**0.26 (0.14)**	**0.21 (0.10)**	**0.21 (0.11)**
wine	0.12 (0.08)	**0.11 (0.09)**	**0.10 (0.08)**	**0.08 (0.07)**

Table 5: Error - 20% noise, decision stumps

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	0.09 (0.04)	0.09 (0.04)	**0.06 (0.02)**	0.07 (0.02)
ecoli	0.33 (0.08)	0.29 (0.07)	**0.22 (0.04)**	**0.21 (0.05)**
glass	0.40 (0.10)	0.43 (0.10)	**0.34 (0.09)**	**0.33 (0.11)**
heart-disease	0.26 (0.07)	0.26 (0.06)	**0.21 (0.07)**	**0.22 (0.09)**
ionosphere	0.25 (0.06)	0.24 (0.07)	**0.12 (0.06)**	**0.13 (0.03)**
iris	0.20 (0.12)	0.18 (0.11)	0.09 (0.08)	**0.05 (0.04)**
liver-disorders	**0.32 (0.06)**	**0.32 (0.05)**	**0.34 (0.05)**	**0.34 (0.06)**
pima-indians-diabetes	**0.26 (0.05)**	**0.27 (0.05)**	**0.25 (0.03)**	**0.26 (0.05)**
satimage	**0.16 (0.01)**	**0.15 (0.01)**	0.17 (0.01)	0.17 (0.01)
sonar	0.34 (0.07)	0.35 (0.11)	**0.29 (0.10)**	**0.27 (0.08)**
soybean-large	0.40 (0.09)	**0.33 (0.08)**	**0.29 (0.07)**	0.34 (0.07)
soybean-small	0.30 (0.10)	0.29 (0.08)	**0.21 (0.13)**	**0.22 (0.12)**
wine	0.15 (0.07)	0.14 (0.06)	**0.09 (0.06)**	**0.10 (0.06)**

Table 6: Error - 0% noise, decision tree (depth 8)

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	**0.05 (0.02)**	**0.05 (0.02)**	**0.05 (0.02)**	**0.04 (0.03)**
ecoli	**0.14 (0.06)**	**0.14 (0.05)**	0.19 (0.06)	**0.12 (0.05)**
glass	0.28 (0.12)	0.27 (0.10)	0.28 (0.11)	**0.21 (0.08)**
heart-disease	**0.24 (0.07)**	**0.24 (0.08)**	**0.24 (0.07)**	**0.21 (0.07)**
ionosphere	0.12 (0.06)	0.12 (0.06)	**0.11 (0.06)**	**0.09 (0.05)**
iris	**0.06 (0.06)**	**0.06 (0.06)**	**0.06 (0.05)**	**0.05 (0.05)**
liver-disorders	**0.28 (0.06)**	**0.28 (0.05)**	0.31 (0.08)	**0.29 (0.06)**
pima-indians-diabetes	**0.25 (0.04)**	**0.25 (0.04)**	**0.24 (0.05)**	**0.26 (0.05)**
satimage	**0.07 (0.01)**	**0.07 (0.01)**	0.08 (0.01)	**0.08 (0.01)**
sonar	**0.28 (0.09)**	**0.28 (0.09)**	**0.30 (0.10)**	**0.26 (0.07)**
soybean-large	**0.08 (0.04)**	**0.08 (0.04)**	0.12 (0.06)	**0.08 (0.04)**
soybean-small	0.16 (0.09)	0.17 (0.10)	**0.16 (0.10)**	**0.11 (0.09)**
wine	**0.09 (0.06)**	**0.09 (0.05)**	**0.07 (0.06)**	**0.09 (0.06)**

Table 7: Error - 10% noise, decision tree (depth 8)

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	**0.06 (0.03)**	**0.06 (0.03)**	**0.05 (0.03)**	0.07 (0.03)
ecoli	**0.15 (0.07)**	**0.15 (0.07)**	**0.18 (0.07)**	**0.15 (0.07)**
glass	0.27 (0.09)	**0.25 (0.08)**	0.33 (0.10)	**0.22 (0.07)**
heart-disease	**0.24 (0.09)**	**0.23 (0.08)**	**0.24 (0.07)**	**0.25 (0.09)**
ionosphere	0.15 (0.07)	0.17 (0.06)	**0.14 (0.07)**	**0.11 (0.05)**
iris	**0.12 (0.10)**	**0.13 (0.09)**	**0.13 (0.07)**	**0.10 (0.08)**
liver-disorders	**0.33 (0.09)**	**0.34 (0.10)**	**0.35 (0.09)**	**0.32 (0.07)**
pima-indians-diabetes	**0.27 (0.04)**	**0.27 (0.04)**	**0.25 (0.05)**	**0.27 (0.04)**
satimage	**0.08 (0.01)**	**0.08 (0.01)**	0.09 (0.01)	**0.09 (0.01)**
sonar	**0.35 (0.11)**	**0.32 (0.10)**	**0.32 (0.09)**	**0.29 (0.12)**
soybean-large	0.19 (0.08)	0.18 (0.08)	**0.16 (0.07)**	**0.14 (0.06)**
soybean-small	0.26 (0.13)	0.21 (0.10)	0.26 (0.14)	**0.15 (0.10)**
wine	**0.14 (0.07)**	**0.15 (0.07)**	**0.15 (0.09)**	**0.12 (0.07)**

Table 8: Error - 20% noise, decision tree (depth 8)

Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer-wisconsin	**0.11 (0.04)**	0.12 (0.04)	**0.09 (0.04)**	0.12 (0.04)
ecoli	**0.18 (0.06)**	**0.17 (0.07)**	**0.18 (0.07)**	**0.17 (0.06)**
glass	0.30 (0.09)	**0.27 (0.11)**	0.34 (0.10)	**0.25 (0.10)**
heart-disease	**0.27 (0.07)**	**0.26 (0.06)**	**0.30 (0.11)**	**0.28 (0.07)**
ionosphere	**0.20 (0.07)**	0.21 (0.08)	0.22 (0.07)	**0.17 (0.07)**
iris	**0.18 (0.09)**	**0.19 (0.10)**	**0.20 (0.10)**	**0.15 (0.10)**
liver-disorders	**0.36 (0.06)**	**0.38 (0.08)**	**0.36 (0.06)**	**0.34 (0.07)**
pima-indians-diabetes	0.31 (0.04)	0.30 (0.04)	**0.28 (0.04)**	0.32 (0.05)
satimage	**0.09 (0.01)**	**0.09 (0.01)**	0.09 (0.01)	0.10 (0.01)
sonar	0.38 (0.11)	0.38 (0.09)	0.34 (0.07)	**0.26 (0.09)**
soybean-large	0.24 (0.10)	0.25 (0.11)	**0.18 (0.10)**	**0.21 (0.09)**
soybean-small	**0.29 (0.12)**	0.29 (0.09)	**0.29 (0.13)**	**0.23 (0.11)**
wine	0.21 (0.11)	0.19 (0.09)	0.21 (0.11)	**0.09 (0.09)**

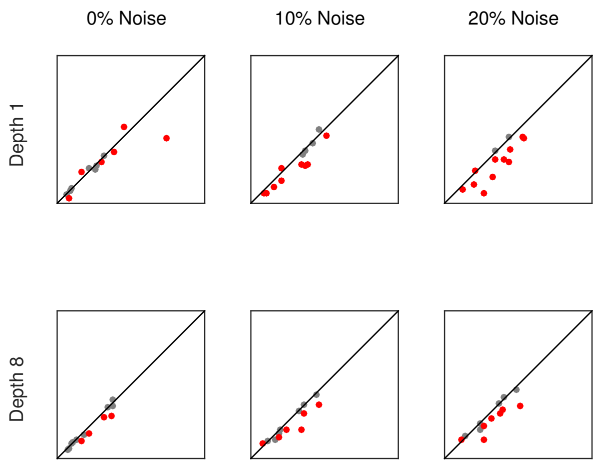

Figure 6: Plots of ValidBoost errors (vertical axes) versus AdaBoost errors (horizontal axes) on UCI datasets for different levels of noise and decision trees of different depths as base classifiers. All axes go from 0 to 0.75 error rate.

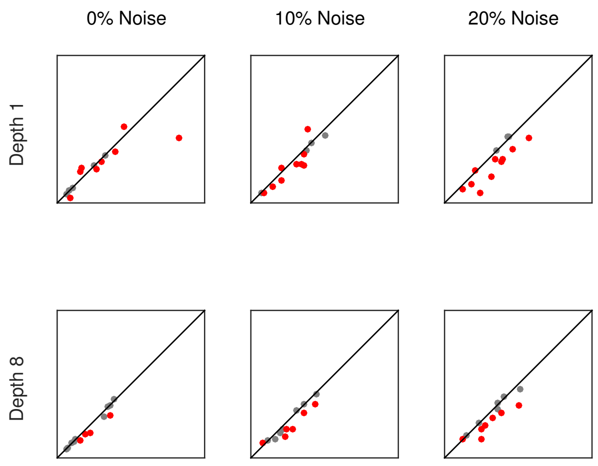

Figure 7: Plots of ValidBoost errors (vertical axes) versus LogitBoost errors (horizontal axes) on UCI datasets for different levels of noise and decision trees of different depths as base classifiers. All axes go from 0 to 0.75 error rate.

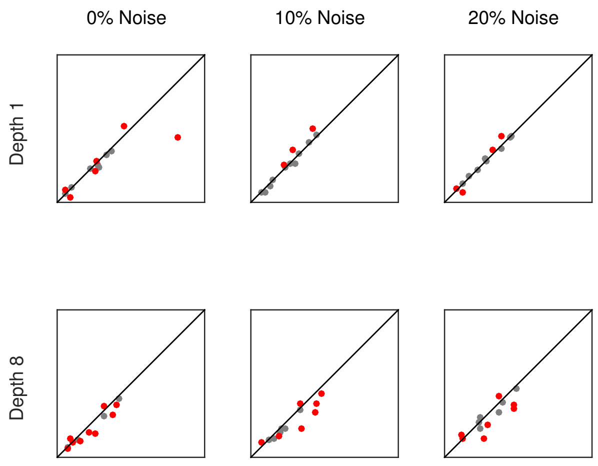

Figure 8: Plots of ValidBoost errors (vertical axes) versus Bylander & Tate's errors (horizontal axes) on UCI datasets for different levels of noise and decision trees of different depths as base classifiers. All axes go from 0 to 0.75 error rate.

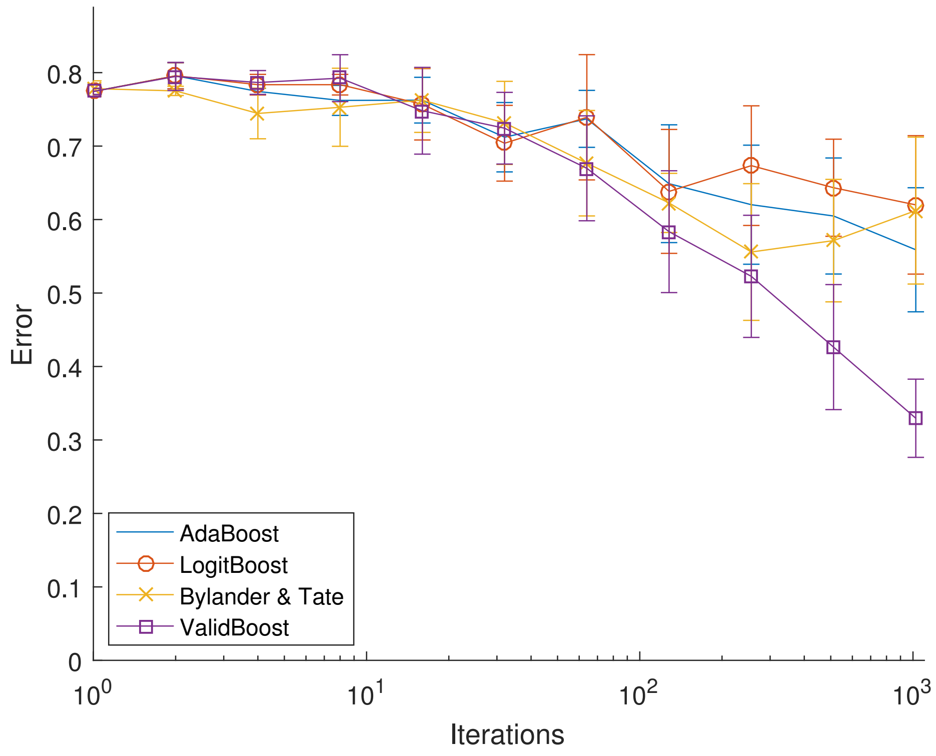

Figure 9: Behavior of the four algorithms on the Soybean-Large dataset, with no added noise and using decision stumps as base classifiers.

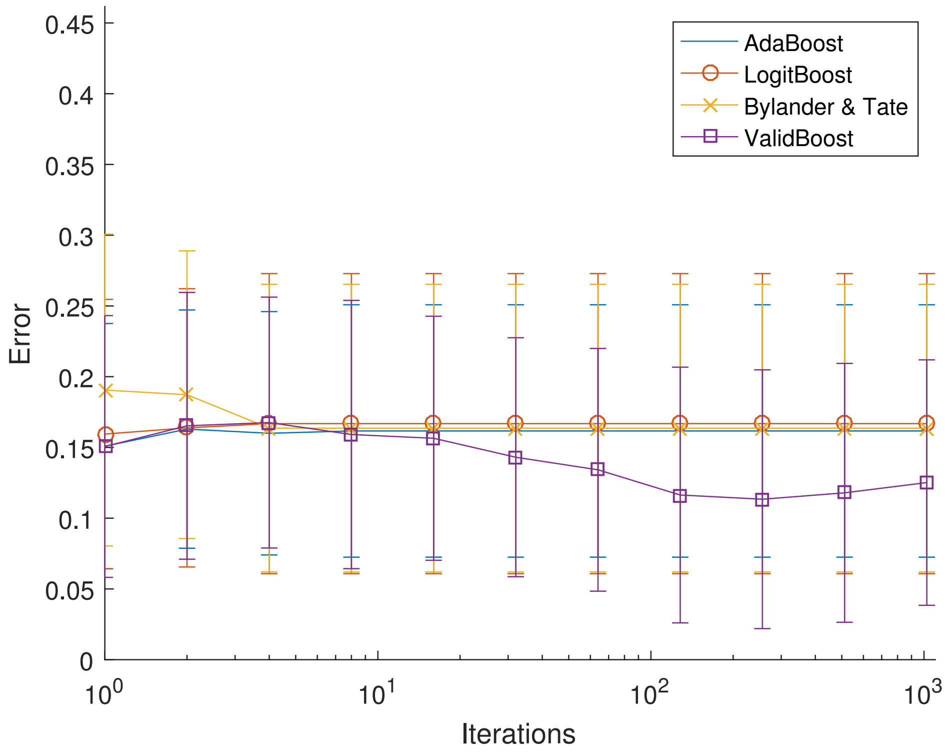

Figure 10: Behavior of the four algorithms on the Soybean-Small dataset, with no added noise and using decision trees as base classifiers.

5 Beyond ValidBoost

After observing that ValidBoost does in fact improve upon AdaBoost, the question arises, "Can we use this knowledge in a different domain?". It is trivial to implement the changes into other boosting algorithms, as all these rely on iterative classifier training and error assessment of each base classifier. An opportunity lies in expanding beyond boosting, into other machine learning techniques.

To this end, we take a step back and view the critical parts of ValidBoost in a non-boosting context, abstracting away from this domain, so that we can specify what it could look like in different domains. We suggest the following:

> The central idea behind ValidBoost is to increase the dependency on, and size of a validation set over the duration of the algorithm, to initially train with all the data, but eventually prevent overfitting the training error.

In order for the dependency on a validation set to increase, there need at least be multiple steps, for the dependency to increase over. This can be iterative, like with boosting, or perhaps recursive, where the 'duration' can be replaced with 'depth', or something else entirely.

5.1. Bagging

Bagging, an amalgamation of 'bootstrap aggregating', is a popular ensemble method for classification, most well known for the success of Leo Breiman's Random Forests [1]. The idea is to sample N objects from a dataset of N objects, *with replacement*, train a base classifier on this sample and repeat this B times. In the end, all base classifiers are combined by majority vote with equal weight. Like boosting, bagging is an iterative process.

The generalization error of a base classifier for bagging is usually estimated by the out-of-bag error, the classification error on all the objects that were *not* in the sample on which the base classifier was trained. The out-of-bag error can then be used to decide not to use a specific base classifier, if the out-of-bag error is "too high".

The accuracy of the out-of-bag error obviously scales with the number of objects on which it is calculated, which is correlated to the size of the sample we take for the training of each base classifier. When we sample N objects from a dataset of size N, the expected number of objects not in the sample is N/e [10].

A possible way to apply our insights to bagging could be to not take a sample of N objects, but a sample of M objects, where M decreases between $b = 1$ and $b = B$. Initially, we would use a lot of the objects to train, and have very little objects to calculate the out-of-bag error on, but as M decreases in later iterations, we have less data used for training, and more data used to estimate the out-of-bag error. In this way, we can reject base classifiers that have an out-of-bag error above a certain threshold, which will lower the average out-of-bag error of the ensemble and hopefully lower the generalization process.

Because of the way the accuracy of the out-of-bag error scales with M, the threshold could possibly be a function of M (or b) as well, decreasing the threshold and rejecting more classifiers, as our estimate of the error becomes more accurate.

We investigated a simple rendition of such an algorithm. We trained a random forest of 250 decision trees on the UCI datasets (table 1). The trees select a single random feature to split on at each node. We compare the resulting ensemble with an ensemble of just the 200 trees with the lowest out-of-bag error. Finally, we compare this with a random forest where the sample size at each iteration is logarithmically decreased from N to $N/2$, and we reject the 50 trees with the highest product of out-of-bag error and out-of-bag size. This way, we use both the error itself and our approximated accuracy of the error, in the form of the number of objects on which the error is estimated. The results are shown in tables 9, 10 and 11, for 0%, 10% and 20% label noise in the training set respectively. All results were obtained with 10-fold crossvalidation, five times repeated.

We observe that this method works well for some datasets (ecoli being a prime example) and poorly for others. In general, we cannot be certain whether removing base classifiers from the ensemble will increase the performance of the ensemble, but these results show that it *might*. As noise is added to the training set, the benefit of rejecting base classifiers becomes more clear. The difference between just rejecting the classifiers with the largest out-of-bag error and the method based on ValidBoost is not apparent. Both seem to work well, with both having some wins over the other, but neither is strictly better. In the end we have to conclude that while this method may have benefits, using it effectively will require some more refinement.

5.2. Neural Networks

A neural network is a method to estimate or approximate any function, inspired by the central nervous system in the brain. It is a network of some layers of artificial neurons. Each artificial neuron calculates a weighted sum of the output of the neurons in the previous layer, passes this through a transfer function and outputs the result to the artificial neurons in the next layer.

A common way to train neural networks is backpropagation. All the weights are randomly initialized, then the training objects are input into the network, and the network's output is compared to the expected output (the object labels in the case of a classification problem). We calculate the gradient of this error with respect to each individual weight, and subtract a fraction of this gradient from each weight. In this way, the total error is expected to decrease. This process is repeated a large number of times. The fraction of the gradient used is also known as the 'learning rate' and it determines how fast the weights adapt to the reinforcement. A higher learning rate will increase the speed with which the network learns, but also adds the possibility of overshooting the target.

A possible way to extend the idea behind ValidBoost to neural networks is easily imagined, initially we use the entire dataset for the backpropagation training process, but gradually increase the size of a holdout sample that is no longer use for backpropagation. The performance on this holdout sample should be inspected at each iteration of the backpropagation algorithm, because as the network is learning, we expect not just the training error to decrease, but the validation error as well.

The validation error can simply be used as a stopping criterion, the training should be stopped when back-propagation no longer decreases (or even increases) the validation error. This allows for quicker training of the network than keeping a validation set apart from the start, because with more training objects in the earlier iterations, the network will learn faster.

We performed a simple experiment to see if an implementation in neural networks has potential. We used a neural network with 1 hidden layer consisting of 10 neurons, and a single neuron in the output layer, constricting this experiment to two-class datasets. Each neuron has a sigmoidal activation function. The learning rate was set to a constant $\alpha = 10^{-4}$. The stopping criterion is that the classification error has not improved in the last 50 iterations, with a minimum of 100 and a maximum of 100000 iterations. The networks are trained by full-batch back-propagation.

We investigate three scenarios, in the first we use the entire training set for back-propagation training, determining the stopping criterion on the same training set. In the second scenario, we set apart 20% of the training set as a validation set, and use this validation set to determine the stopping criterion, but not for the back-propagation. In the third scenario, we calculate $\tau = \log t / \log T$, where t is the current iterations number and $T = 100000$ is the maximum number of iterations, and take a growing validation set from the training set of size $\tau N/5$. Note that unlike in ValidBoost, the validation set is not randomly sampled every iteration, but increments in size over the training process. To be able to make a more fair comparison, the validation set this third method accumulates over the course of the training process is the same as the validation set the second method uses. The error is identical to the error defined for ValidBoost, $\epsilon = \tau\epsilon_v + (1-\tau)\epsilon_t$, a weighted average of the error on the training set and the error on the validation set. The stopping criterion is calculated on this error. All three networks were initialized with the same random weights and biases.

The results are shown in tables 12, 13, 14 and 15, for 0%, 10%, 20% and 30% label noise in the training set respectively. All results were obtained with 10-fold crossvalidation, ten times repeated. We observe that the method of incrementally growing the validation set seems to be working rather sporadically. When it achieves a better classification performance than the vanilla network, it is often still beaten by the network with a constant validation set. There seems to be no common pattern to distinguish when it does and does not work at first sight, and a better way to implement the idea behind ValidBoost will have to be designed before any conclusions about its possible application in Neural Networks can be drawn.

5.3. Decision Trees

Because decision trees are inherently recursive, they too might lend themselves to the use of validation sets of increasing size. A decision tree, given labeled objects, determines a threshold on one of the features and splits the space in two subspaces. It then recurses into these subspaces and repeats until either a maximum depth is reached, or all objects in the subspace belong to the same class.

We propose the following: at each node of the tree, we take a fraction of the objects apart to use for a validation set. As we recurse into the tree, we accumulate a larger holdout sample for every depth. This will allow us to evaluate a split before committing to it. For 'vanilla' decision trees, this might not seems that useful, after all, why not just use this data to train? But it is common practice for decision trees to select a random subset of features as every node, instead of inspecting all the features. Not only does this speed up the training, it also increases the variance between two different trees, which is the goal of bagging. With the validation set, we might be able to avoid splitting on features that are not as discriminative as features that were not selected, by looking at the difference between the performance of such a split on the training set and on the validation set.

It should be noted that unlike in bagging and boosting, we are only training a single classifier. In bagging and boosting, we do not care much about the individual results of our random selection of validation objects, because we are training many classifiers and the sampling bias will be averaged out. In the case of a decision tree, this method may result in (for example) an undertrained classifier, if we happen to take the most descriptive objects as our validation objects. Because we do not come back to this point at a later stage, like we would with bagging and boosting, the result of bad sampling is permanent.

5.4. General Thoughts on Effectiveness

As we have seen, the idea behind ValidBoost can be used in other domains, but with limited success. This is expected, as it was not designed with these domains in mind. In this section, we want to explore what parameters have influence on its effectiveness, in order to be able to give a recommendation of when (and when not) to use a validation set of increasing size and influence.

We expect the amount of label noise and outliers in the training data to have a lot of influence. We note in our trials that without added noise the unregularized methods, which use the entire training set for training, usually performed quite well and not a lot of improvement is made by introducing a validation set. But it should be noted that not all noise is *added* noise. A classifier will more easily overfit on a dataset with a high Bayes error than on one with a low Bayes error. To test this hypothesis, we take a very rough estimate of the Bayes error: the classification error the unregularized method makes, and compare this to the relative improvement of our respective method with a validation set of increasing size. Figures 11, 12 and 13 show this comparison for AdaBoost, Random Forests and Neural Networks respectively. It seems that this assumption holds for AdaBoost, but not for Random Forests or Neural Networks. At the very least, this shows that the potential improvement on random forests and neural networks is an order of magnitude smaller than it is for AdaBoost. This may be either because overfitting is less common for these methods, or because our regularizations (rejecting base classifiers and early stopping) are less appropriate than the adjusted error estimation was for AdaBoost.

The gradual introduction of the validation set is based on a pattern found in AdaBoost's learning process: most error reduction happens in early iterations, and most overfitting happens in late iterations. This pattern does not hold for random forests, the order of the training is irrelevant and all classifiers can be trained simultaneously, as no information is transmitted between base classifiers (as opposed to AdaBoost, where each base classifier changes the object weights). For neural networks, this interaction is less clear. Training happens based on the gradients of the transfer functions of each neuron and because of random initialization of the weights and the biases, we cannot be sure *when* most of the learning happens. This could be in early iterations or in late iterations, depending on both the starting parameters and the hyperparameters of the network.

Perhaps the conclusion we have to make is that the success of ValidBoost is limited to the original use case, improving AdaBoost, because it is based on principles and assumptions that hold for AdaBoost, but translate to other domains poorly.

Table 9: Classification error for Random Forests - 0% noise

Datasets	Random Forest	with 20% rejection	with 20% rejection and decreasing sample size
breast-cancer-wisconsin	**0.03 (0.01)**	0.04 (0.02)	0.04 (0.02)
ecoli	0.22 (0.03)	0.23 (0.04)	**0.18 (0.05)**
glass	0.23 (0.06)	**0.20 (0.04)**	0.32 (0.06)
heart-disease	**0.12 (0.07)**	0.21 (0.06)	0.23 (0.08)
ionosphere	**0.07 (0.06)**	**0.06 (0.02)**	0.07 (0.03)
iris	**0.03 (0.03)**	0.09 (0.03)	**0.03 (0.03)**
liver-disorders	0.29 (0.07)	**0.22 (0.06)**	0.27 (0.07)
pima-indians-diabetes	0.26 (0.03)	0.24 (0.01)	**0.22 (0.03)**
satimage	0.10 (0.00)	0.09 (0.01)	**0.09 (0.01)**
sonar	**0.16 (0.09)**	**0.14 (0.04)**	0.21 (0.07)
soybean-large	0.24 (0.02)	**0.21 (0.08)**	0.24 (0.07)
soybean-small	**0.11 (0.10)**	0.17 (0.07)	**0.14 (0.06)**
wine	**0.01 (0.02)**	**0.01 (0.02)**	**0.01 (0.02)**

Table 10: Classification error for Random Forests - 10% noise

Datasets	Random Forest	with 20% rejection	with 20% rejection and decreasing sample size
breast-cancer-wisconsin	0.03 (0.02)	0.05 (0.02)	**0.02 (0.01)**
ecoli	0.27 (0.06)	**0.21 (0.02)**	**0.21 (0.07)**
glass	0.25 (0.04)	**0.21 (0.10)**	**0.23 (0.05)**
heart-disease	0.22 (0.03)	**0.14 (0.10)**	0.19 (0.03)
ionosphere	0.11 (0.09)	**0.07 (0.05)**	**0.06 (0.02)**
iris	0.07 (0.04)	**0.03 (0.03)**	**0.03 (0.03)**
liver-disorders	0.33 (0.06)	**0.23 (0.05)**	0.27 (0.05)
pima-indians-diabetes	**0.22 (0.04)**	0.25 (0.03)	**0.23 (0.07)**
satimage	0.10 (0.01)	**0.09 (0.01)**	0.09 (0.01)
sonar	0.23 (0.05)	0.23 (0.06)	**0.19 (0.06)**
soybean-large	**0.25 (0.06)**	**0.24 (0.05)**	**0.24 (0.10)**
soybean-small	**0.14 (0.10)**	0.20 (0.12)	0.21 (0.05)
wine	**0.00 (0.00)**	0.01 (0.02)	0.02 (0.03)

Table 11: Classification error for Random Forests - 20% noise

Datasets	Random Forest	with 20% rejection	with 20% rejection and decreasing sample size
breast-cancer-wisconsin	0.05 (0.02)	0.05 (0.02)	**0.02 (0.01)**
ecoli	0.28 (0.03)	0.25 (0.02)	**0.19 (0.05)**
glass	**0.18 (0.03)**	0.25 (0.11)	0.25 (0.04)
heart-disease	0.19 (0.05)	0.20 (0.04)	**0.13 (0.07)**
ionosphere	**0.09 (0.04)**	**0.10 (0.01)**	**0.09 (0.05)**
iris	**0.07 (0.04)**	**0.07 (0.04)**	**0.07 (0.04)**
liver-disorders	**0.30 (0.07)**	0.34 (0.05)	**0.29 (0.05)**
pima-indians-diabetes	**0.26 (0.03)**	**0.26 (0.08)**	0.29 (0.03)
satimage	**0.10 (0.01)**	**0.10 (0.01)**	**0.10 (0.00)**
sonar	**0.24 (0.09)**	**0.24 (0.05)**	**0.26 (0.09)**
soybean-large	**0.28 (0.10)**	**0.28 (0.10)**	**0.27 (0.06)**
soybean-small	**0.17 (0.07)**	**0.19 (0.13)**	0.37 (0.05)
wine	**0.02 (0.03)**	**0.02 (0.03)**	0.04 (0.04)

Table 12: Classification error for Neural Networks - 0% noise

Datasets	Neural Network	with 20% validation set	with growing validation set
banana	**0.04 (0.01)**	0.05 (0.02)	0.05 (0.02)
breast-cancer-wisconsin	0.04 (0.02)	**0.03 (0.02)**	0.04 (0.02)
heart-disease	0.23 (0.08)	**0.21 (0.05)**	**0.20 (0.03)**
ionosphere	0.19 (0.07)	**0.16 (0.08)**	0.22 (0.07)
liver-disorders	0.36 (0.08)	0.37 (0.08)	**0.29 (0.07)**
pima-indians-diabetes	0.26 (0.06)	**0.25 (0.03)**	0.28 (0.06)
sonar	0.32 (0.11)	**0.27 (0.06)**	0.29 (0.06)

Table 13: Classification error for Neural Networks - 10% noise

Datasets	Neural Network	with 20% validation set	with growing validation set
banana	**0.04 (0.01)**	0.05 (0.03)	0.04 (0.01)
breast-cancer-wisconsin	0.05 (0.02)	**0.04 (0.02)**	**0.04 (0.02)**
heart-disease	0.25 (0.10)	0.21 (0.08)	**0.19 (0.06)**
ionosphere	**0.21 (0.07)**	0.24 (0.08)	**0.22 (0.09)**
liver-disorders	**0.32 (0.07)**	0.35 (0.07)	0.35 (0.05)
pima-indians-diabetes	0.28 (0.03)	**0.27 (0.03)**	**0.28 (0.08)**
sonar	0.44 (0.08)	**0.36 (0.10)**	0.41 (0.08)

Table 14: Classification error for Neural Networks - 20% noise

Datasets	Neural Network	with 20% validation set	with growing validation set
banana	**0.05 (0.02)**	**0.05 (0.02)**	**0.05 (0.02)**
breast-cancer-wisconsin	**0.07 (0.03)**	**0.07 (0.03)**	**0.07 (0.03)**
heart-disease	**0.21 (0.07)**	0.28 (0.08)	0.27 (0.08)
ionosphere	**0.25 (0.08)**	0.29 (0.08)	**0.25 (0.07)**
liver-disorders	**0.34 (0.06)**	**0.35 (0.07)**	**0.35 (0.08)**
pima-indians-diabetes	0.30 (0.05)	**0.28 (0.04)**	**0.27 (0.05)**
sonar	**0.33 (0.08)**	0.39 (0.08)	0.36 (0.10)

Table 15: Classification error for Neural Networks - 30% noise

Datasets	Neural Network	with 20% validation set	with growing validation set
banana	0.06 (0.03)	**0.05 (0.02)**	**0.05 (0.02)**
breast-cancer-wisconsin	0.10 (0.04)	**0.08 (0.04)**	0.11 (0.04)
heart-disease	**0.29 (0.10)**	**0.30 (0.07)**	**0.30 (0.09)**
ionosphere	**0.27 (0.09)**	0.31 (0.10)	**0.28 (0.06)**
liver-disorders	**0.37 (0.09)**	**0.38 (0.10)**	**0.38 (0.09)**
pima-indians-diabetes	**0.31 (0.04)**	**0.29 (0.06)**	**0.31 (0.05)**
sonar	**0.35 (0.09)**	0.43 (0.09)	0.41 (0.10)

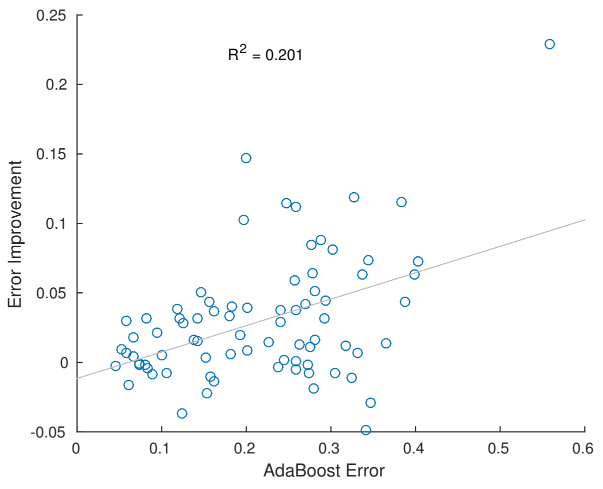

Figure 11: Scatterplot of the classification error AdaBoost makes versus the relative improvement in error of ValidBoost.

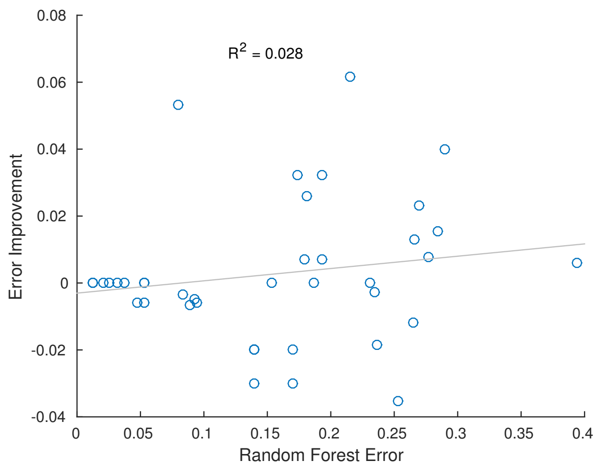

Figure 12: Scatterplot of the classification error Random Forests make versus the relative improvement in error of our method with validation set of increasing size.

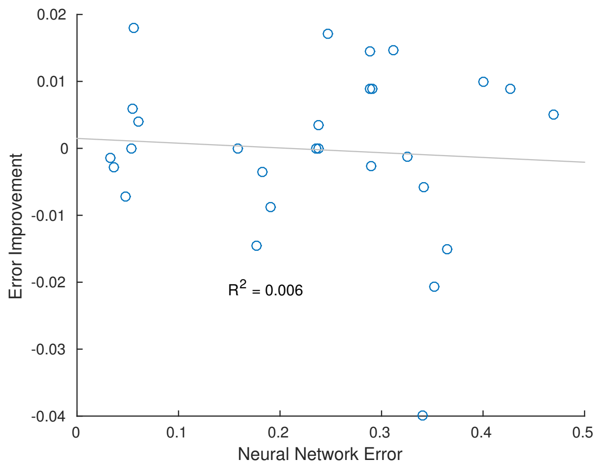

Figure 13: Scatterplot of the classification error Neural Networks make versus the relative improvement in error of our method with validation set of increasing size.

6 Discussion

6.1. Summary

In this work, we have shown that AdaBoost is prone to overfitting when the training set contains mislabeled objects. We proposed that this is in part because the error estimate used to weight base classifiers and (indirectly) objects is biased in this scenario. We have shown that an unbiased estimator can prevent the overfitting, but such an estimator is not easily found when the training set is untrustworthy.

To remedy this, we introduced the ValidBoost algorithm, which tries to unbias the error estimation AdaBoost makes by taking a validation set from the noisy training set. The size of the validation set increases logarithmically with the number of iterations, reaching 50% of the entire training set in the final iteration. The error ValidBoost uses is a weighted average of the default training error and this validation error, with the weight of the validation error also increasing logarithmically with the number of iterations.

We have seen that ValidBoost performs well in comparison to AdaBoost and LogitBoost in the presence of label noise. Compared to Bylander & Tate's method, a similar variant of AdaBoost also based on the use of validation sets to avoid overfitting, it performs very similar when using decision stumps as base classifiers, and slightly better when using decision trees as classifiers. Perhaps this is because we are using more training data than Bylander & Tate are and decision trees require this extra input in order for the base classifiers themselves not to overfit.

The fact that ValidBoost samples a new validation set at every iteration also lead to positive effects. Whereas the other methods might stagnate or stop early because a classifier has been trained with an error above the threshold imposed by boosting, ValidBoost will have a different training set next iteration, thus no reason to stop early and a different error (possibly below the error threshold). This effect can also be seen when using non-deterministic base classifiers with AdaBoost.

Because of the relative success of ValidBoost, we reasoned about its core idea and how this can be translated into other domains. Several suggestions were made for possible ways to incorporate it into bagging, neural networks or decision trees. Experiments on such an implementation in bagging (random forests specifically) showed promise, while experiments on an implementation in neural networks worked less well.

6.2. Future Work

This work has focused on overfitting in boosting, and has shown that validation sets are a useful tool to improve classifier performance when the training set is noisy. The proposed changes to AdaBoost are shown to improve performance in these cases, but an important aspect of AdaBoost has overlooked in the design: AdaBoost is an attractive method because it can be proven (see [28]) to be trained to an *arbitrarily low* generalization error. In other words, any dataset, no matter its complexity, can be fitted by AdaBoost, given enough iterations and sufficiently strong base classifiers. In the design of ValidBoost, this has not been given much thought, and as a result, we do not expect to reach the same mathematical standard. This work has observed AdaBoost and applied intuitive changes, without looking at the implications for the mathematics underlying AdaBoost.

Secondly, however well ValidBoost works, it introduces additional parameters that need to be set. The choice of $\tau = \log t / \log T$ is explained, but obviously not the only possible choice. The added reliance on T might cause issues, in AdaBoost stopping early at iteration T_s is identical to having set $T = T_s$ initially, but for ValidBoost this is not the case. In addition, the choice to end up with a 50/50 split in training and validation set is arbitrary, and some datasets will be better off with a different ratio. These parameters add complexity for the user and the need for additional tuning, on top of the needs of AdaBoost and the base classifiers themselves.

In conclusion, we expect that there is more to ValidBoost (and the related methods and underlying ideas) than we could have discussed here. The theoretical implications of validation sets on boosting are not very well researched, and the possible applications of validation sets inside and outside of boosting are literally endless.

Bibliography

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Gavin Brown and Ludmila I Kuncheva. ‘good’ and ‘bad’ diversity in majority vote ensembles. In *Multiple classifier systems*, pages 124–133. Springer, 2010.

[3] Tom Bylander and Lisa Tate. Using validation sets to avoid overfitting in adaboost. In *FLAIRS Conference*, pages 544–549, 2006.

[4] Padraig Cunningham and John Carney. Diversity versus quality in classification ensembles based on feature selection. In *Machine Learning: ECML 2000*, pages 109–116. Springer, 2000.

[5] Christian Dietrich, Günther Palm, and Friedhelm Schwenker. Decision templates for the classification of bioacoustic time series. *Information Fusion*, 4(2):101–109, 2003.

[6] Nigel Duffy and David Helmbold. Potential boosters? In *Advances in Neural Information Processing Systems 12*. Citeseer, 2000.

[7] Robert PW Duin. The combining classifier: to train or not to train? In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 765–770. IEEE, 2002.

[8] Robert PW Duin and David MJ Tax. Experiments with classifier combining rules. In *Multiple classifier systems*, pages 16–29. Springer, 2000.

[9] Bradley Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[10] Bradley Efron and Robert Tibshirani. Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[12] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2): 337–407, 2000.

[13] Adam J Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pages 692–699, 1998.

[14] Ludmila Kuncheva et al. ” fuzzy” versus” nonfuzzy” in combining classifiers designed by boosting. *Fuzzy Systems, IEEE Transactions on*, 11(6):729–741, 2003.

[15] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[16] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[17] Ludmila I Kuncheva, James C Bezdek, and Robert PW Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern recognition*, 34(2):299–314, 2001.

[18] Louisa Lam. Classifier combinations: implementations and theoretical issues. In *Multiple classifier systems*, pages 77–86. Springer, 2000.

[19] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[20] Philip M Long and Rocco A Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010.

[21] Hamed Masnadi-Shirazi and Nuno Vasconcelos. On the design of loss functions for classification: theory, robustness to outliers, and savageboost. In *Advances in neural information processing systems*, pages 1049–1056, 2009.

[22] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. NIPS, 1999.

[23] David Mease and Abraham Wyner. Evidence contrary to the statistical view of boosting. *The Journal of Machine Learning Research*, 9:131–156, 2008.

[24] Elżbieta Pękalska, Robert PW Duin, and Marina Skurichina. A discussion on the classifier projection space for classifier combining. In *Multiple Classifier Systems*, pages 137–148. Springer, 2002.

[25] Elżbieta Pękalska and Robert PW Duin. Classifiers for dissimilarity-based pattern recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 12–16. IEEE, 2000.

[26] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, (5):401–409, 1969.

[27] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[28] Robert E Schapire. Theoretical views of boosting and applications. In *Algorithmic Learning Theory*, pages 13–25. Springer, 1999.

[29] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.

[30] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[31] Rocco A Servedio. Smooth boosting and learning with malicious noise. *The Journal of Machine Learning Research*, 4:633–648, 2003.

[32] Joaquín Torres-Sospedra, Carlos Hernández-Espinosa, and Mercedes Fernández-Redondo. Improving adaptive boosting with k-cross-fold validation. In *Intelligent Computing*, pages 397–402. Springer, 2006.

[33] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[34] Wikipedia. Curse of dimensionality — Wikipedia, the free encyclopedia, 2004. URL `https://en.wikipedia.org/wiki/Curse_of_dimensionality`. [Online; accessed 29-October-2015].

[35] Kevin Woods, Kevin Bowyer, and W Philip Kegelmeyer Jr. Combination of multiple classifiers using local accuracy estimates. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 391–396. IEEE, 1996.

[36] Abraham J Wyner. On boosting and the exponential loss. In *Proceedings of the Ninth Annual Conference on AI and Statistics Jan*, pages 3–6. Citeseer, 2003.

[37] Lei Xu, Adam Krzyżak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, man and cybernetics, IEEE transactions on*, 22 (3):418–435, 1992.

[38] Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

Appendix: Conference Paper

This appendix contains a research paper, written during this thesis project. This paper is set to be published in the proceedings of the International Conference on Pattern Recognition (ICPR) 2016.

Regularizing AdaBoost with validation sets of increasing size

Dirk W.J. Meijer and **David M.J. Tax**
Delft University of Technology
Delft, The Netherlands
d.w.j.meijer@student.tudelft.nl, d.m.j.tax@tudelft.nl

***Abstract*—AdaBoost is an iterative algorithm to construct classifier ensembles. It quickly achieves high accuracy by focusing on objects that are difficult to classify. Because of this, AdaBoost tends to overfit when subjected to noisy datasets. We observe that this can be partially prevented with the use of validation sets, taken from the same noisy training set. But using less than the full dataset for training hurts the performance of the final classifier ensemble. We introduce ValidBoost, a regularization of AdaBoost that takes validation sets from the dataset, increasing in size with each iteration. ValidBoost achieves performance similar to AdaBoost on noise-free datasets and improved performance on noisy datasets, as it performs similar at first, but does not start to overfit when AdaBoost does.**

Keywords: Supervised learning, Ensemble learning, Regularization

I. Introduction

In 1997 Freund and Schapire introduced AdaBoost ([1], algorithm 1), a highly successful, iterative ensemble method that adapts to difficult objects by re-weighting based on correct or incorrect classification in earlier iterations. A problem with this method is that it might overfit when run for too many iterations, i.e. the performance on an unseen test set may deteriorate compared to earlier iterations ([2], [3]). Although this point is heavily debated (see [4] for example), the presence of mislabeled training objects seems to increase the likelihood of overfitting drastically ([5], [6]).

A common method to avoid overfitting a classifier is to use a holdout sample as validation set to estimate to actual error a method is making, instead of relying only on the training error. An extension of this is cross-validation, we rotate the set in such a way that every training object is in a holdout sample exactly once, and we average the obtained validation errors. This allows you to still use the entire training set, while still relying on a holdout sample for validation.

Bylander and Tate ([7]) incorporated validation sets in AdaBoost for the same reasons. In their method they perform a stratified split of the training data once, train AdaBoost twice, using both sets as training set and validation set in turn and combining the results of both into a single ensemble classifier. They show very promising results, especially in the presence of label noise. It should be noted however, that an unlucky initial split can cause problems.

Torres-Sospedra et. al ([8]) go a little further and perform k-fold cross-validation within AdaBoost. Their method of implementing it somewhat curious however, the number of folds and the number of iterations is equal by definition and each iteration uses a different validation fold. This limits its application, as we have to keep this in mind when choosing a number of iterations to run AdaBoost for, and particularly small datasets will limit this method to a low number of iterations. This method was also made to work specifically with neural networks, which may explain why they have no need for a large number of base classifiers, as neural networks are already a relatively strong classifier.

Algorithm 1: AdaBoost

```
Input: Dataset $\mathbf{x}$, consisting of $N$ objects
       $\langle x_1, \dots, x_N \rangle \in X$ with labels
       $\langle y_1, \dots, y_N \rangle \in Y = \{1, \dots, c\}$
Input: Weak learning algorithm WeakLearn
Input: Number of iterations $T$
Initialize: Weight vector $w_i^1 = \frac{1}{N}$ for $i = 1, \dots, N$
for $t = 1$ to $T$ do
    Call WeakLearn, providing it with weights $w_i^t$
    Get back hypothesis $h_t : X \to Y$
    Compute $\epsilon = \sum_{i=1}^{N} w_i^t [h_t(x_i) \neq y_i]$
    if $\epsilon > \frac{1}{2}$ then
        Set $\alpha_t = 0$
        Set $w_i^{t+1} = w_i^t$
    else
        Compute $\alpha_t = \log((1 - \epsilon)/\epsilon)$
        Set $w_i^{t+1} = w_i^t \exp(\alpha_t [h_t(x_i) \neq y_i])$ for all $i$
        Normalize $w_i^{t+1}$
    end
end
Output: Hypothesis $H(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t [h_t(x) = y]$
```

The exact reason for overfitting is not entirely clear. It has been attributed to the fact that outliers get extremely large weights and the choice of the exponential loss function, which has spawned methods such as LogitBoost [2] and DOOM II [9], that regularize the way weights are assigned in one way or another. While it may certainly be true that the weights play a role, we believe that a large part of the problem is that fact that the error estimation for the classifier weights (line 4 in algorithm 1) is biased.

In this paper we investigate the overfitting behavior of

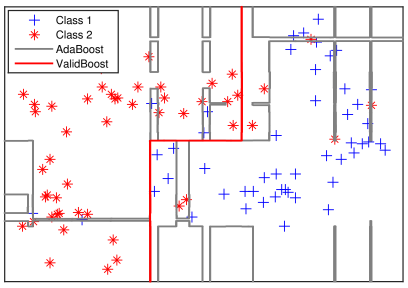

Fig. 1: Two-dimensional, two-class, banana-shaped dataset with 10% noise

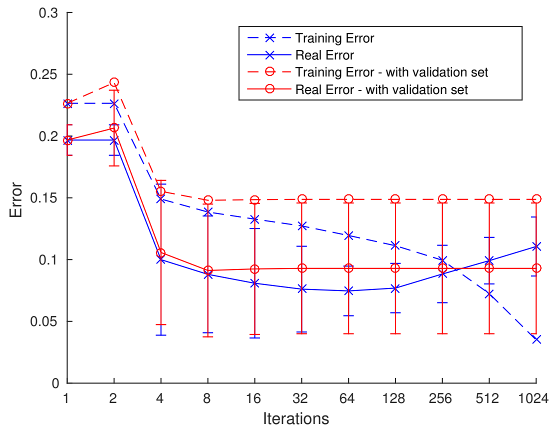

Fig. 2: Behavior of AdaBoost and the altered algorithm with an unbiased error estimator. Averaged over 20 trials, with the standard deviation in error bars

AdaBoost. To this end we start in section II by showing that overfitting behavior can be suppressed by good error estimates ε. In section III a practical algorithm ValidBoost is proposed to avoid this overfitting. In section IV ValidBoost is compared to AdaBoost and a more robust versions of boosting, and the paper is concluded in section V with some discussion.

II. AdaBoost overfitting

To illustrate the overfitting behavior of AdaBoost, we look at an artificial example. A two-dimensional two-class banana-shaped dataset with 10% added label noise is used, as shown in Figure 1. The 10% label noise is added to this dataset by picking 10% of the objects at random and changing the label to that of the opposite class. We run AdaBoost for a large number of iterations (1024), using decision stumps as the base classifiers, and generate a large test set from the same distribution as the (noise-free) training set to assess the classification error made by AdaBoost.

The results are shown in Figure 2. The dashed blue graph shows the error on the (noisy) training set, as a function of the number of boosting iterations. As can observed, this error decreases almost to zero, suggesting that AdaBoost is overfitting on the training set. The solid blue graph, indicating the true (noise-free) test error, confirms this. This error increases with increasing number of iterations. Note that due to the noise in the training set, the error on the training set is a bit higher than on the test set.

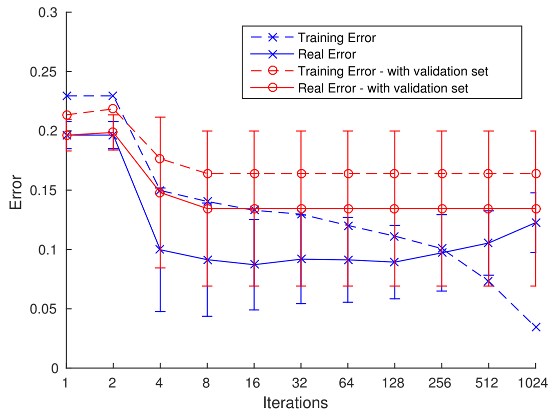

Fig. 3: Behavior of AdaBoost and the altered algorithm with half the dataset as training data, the other half as validation set. Averaged over 20 trials, with the standard deviation in error bars

We then make a small change to AdaBoost, we use the same training set to train the base classifiers, but now we use the test set to calculate the error ϵ and assign weights α_t to the base classifiers. The results are indicated by the red graphs in Figure 2. Clearly, the overfitting is avoided. This is highly artificial of course, but it shows what would happen if we had a perfect, unbiased estimate of the error a base classifier makes.

As we can see in figure 2, having a better estimate of the error a classifier make can certainly stop AdaBoost from overfitting. This specific example relies on having a noise-free validation set, which is an unfeasible requirement in practically all real-world cases. More realistically, we can split the noisy training set into two sets, and use one set to train the base classifiers and the other set to calculate ϵ and α. The results of this experiment on our same banana-shaped set are shown in figure 3. While this method seems to be effective in preventing overfitting, it does not give an error rate remotely as good as AdaBoost does. One obvious reason for this, is that we are using only half of our dataset.

III. ValidBoost

AdaBoost is known to reach high accuracy very quickly, which can also be observed in our graphs. To get the best of both worlds, we propose as next step to gradually increase the influence of our validation set. Initially, we want our method to be identical to AdaBoost, so that it can reach low error levels, but over time, we want to introduce a validation set (taken from our training set) to avoid overfitting like we did in the last few experiments. The question then becomes, how gradually?

We observe that the biggest changes occur in the first few iterations of AdaBoost. This is expected, as each iteration adds

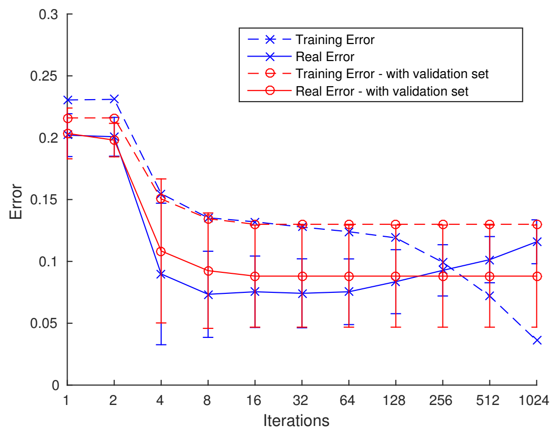

Fig. 4: Behavior of AdaBoost and the altered algorithm with half the dataset as training data, the other half as validation set, with logarithmically increasing influence of the validation set. Averaged over 20 trials, with the standard deviation in error bars.

a single classifier to a growing ensemble of classifiers. At iteration t, the 'impact' of a new classifier on the current ensemble is $1/t$. Because of this, we suggest a logarithmic increase for the introduction of the validation set, to take full advantage in the earlier iterations, when the impact of each iteration is larger. We define

$$\tau = \frac{\log t}{\log T} \tag{1}$$

$$\epsilon = \tau\epsilon_v + (1-\tau)\epsilon_t \tag{2}$$

Where t is the current iteration, T is the total number of iterations, ϵ_v is the weighted error on the validation set and ϵ_t is the weighted error on the training set. In the first few iterations, τ will be small and we will be relying heavily on the training error, akin to vanilla AdaBoost. As t increases, τ will tend towards 1 and the influence of the validation will quickly become larger, in order to avoid overfitting on the training set. The performance with this change on the banana-shaped dataset are shown in figure 4.

These results are again an improvement, but one detail has not changed, we are still using only half of the training set, even in the initial iterations where we are hardly using the validation set at all. This brings us to the final step in our train of thought, we change the size of the validation set, linear in τ. At the start of each iteration, we perform a stratified random split of the training set $\mathbf{x}$ into a validation set $\mathbf{x}_v$ of size $\tau N/2$, and the remainder, $\mathbf{x}_t$, which will be used as training set for this particular iteration. As such, we start out with an empty validation set and a full training set, and end up with a 50/50 split between the two at iteration T.

The final algorithm is shown in algorithm 2. The main difference with algorithm 1 is that it splits the training set into a new training set $\mathbf{x}_t$ and validation set $\mathbf{x}_v$ (line 3), and that the error ϵ is now a weighted combination of the training error ϵ_t and validation error ϵ_v.

The performance on the banana-shaped dataset can be seen in figure 5. It can be observed that ValidBoost converges as fast as AdaBoost to a low error, but in contrast to AdaBoost, the true error does not increase.

It has to be noted that ValidBoost has a higher reliance on the hyperparameter T than AdaBoost. If we run ValidBoost for a particularly short number of iterations, τ may increase

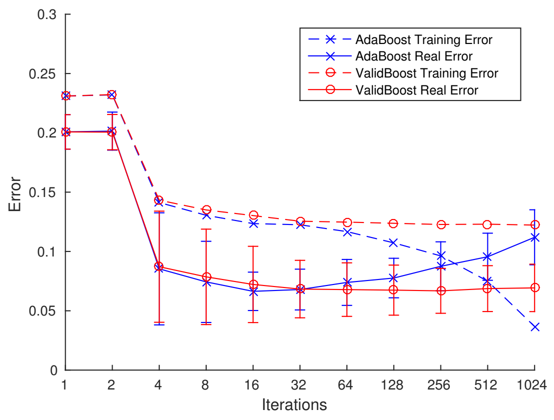

Fig. 5: Behavior of AdaBoost and ValidBoost. Averaged over 20 trials, with the standard deviation in error bars.

Algorithm 2: ValidBoost

```
Input: Dataset $\mathbf{x}$, consisting of $N$ objects
       $\langle x_1,\dots,x_N\rangle \in X$ with labels
       $\langle y_1,\dots,y_N\rangle \in Y = \{1,\dots,c\}$
Input: Weak learning algorithm WeakLearn
Input: Number of iterations $T$
Initialize: Weight vector $w_i^1 = \frac{1}{N}$ for $i = 1,\dots,N$
1  for $t = 1$ to $T$ do
2      Set $\tau = \frac{\log t}{\log T}$
3      Split $\mathbf{x}$ into $\mathbf{x}_v$ of size $\tau N/2$ and $\mathbf{x}_t$
4      Call WeakLearn on $\mathbf{x}_t$, providing it with weights
       $w_i^t$ for $\{i : x_i \in \mathbf{x}_t\}$
5      Get back hypothesis $h_t : X \to Y$
6      Compute $\epsilon_v = \sum_{\{i:x_i\in\mathbf{x}_v\}} w_i^t[h_t(x_i) \neq y_i]$
7      Compute $\epsilon_t = \sum_{\{i:x_i\in\mathbf{x}_t\}} w_i^t[h_t(x_i) \neq y_i]$
8      Set $\epsilon = \tau\epsilon_v + (1-\tau)\epsilon_t$
9      if $\epsilon > \frac{1}{2}$ then
10         Set $\alpha_t = 0$
11         Set $w_i^{t+1} = w_i^t$
12     else
13         Compute $\alpha_t = \log((1-\epsilon)/\epsilon)$
14         Set $w_i^{t+1} = w_i^t \exp(\alpha_t[h_t(x_i) \neq y_i])$ for all $i$
15         Normalize $w_i^{t+1}$
16     end
17 end
Output: Hypothesis $H(x) = \arg\max_{y\in Y} \sum_{t=1}^{T} \alpha_t[h_t(x) = y]$
```

too fast, leading to errors not as low as AdaBoost might give us. In the other extreme, if we pick a T that is too large, τ might not increase fast enough to prevent overfitting. But since τ scales with the logarithm of T, and not with T directly, these changes have to be rather extreme before an effect is observed.

IV. Experiments

TABLE I: Comparison between scaling τ for a few UCI dataset. $T = 1024$. As base classifier a tree with depth 4 is used, and except for the first dataset, no noise has been added.

dataset	$\log(t)/\log(T)$	t/T
Banana 20% noise	**24.4 (16.9)**	**29.4 (18.5)**
Ionosphere	**8.0 (4.4)**	**6.6 (5.0)**
Soybean-large	**8.6 (3.9)**	**8.6 (4.3)**
Soybean-small	**11.0 (6.9)**	**14.7 (9.6)**
Sonar	**25.6 (17.2)**	**26.1 (16.2)**
wine	**11.3 (9.8)**	**13.2 (10.4)**

In the first experiment, we compare the logarithmic scaling of $\tau = \log(t)/\log(T)$ with a linear scaling t/T. Table I shows the classification performances on different datasets. A fairly large number of runs is used, $T = 1024$, and a slightly more complicated decision tree, a tree with depth 4. Sometimes a bit improvement is observed using the logarithmic scaling, although the differences are never significant.

TABLE II: Comparison of ValidBoost with different values for T. As base classifier decision stumps are used and no noise had been added. Results are obtained using 10-fold crossvalidation.

Datasets	$T = 10$	$T = 100$	$T = 1000$	$T = 10000$
breast-cancer	**0.05 (0.02)**	**0.04 (0.02)**	**0.05 (0.02)**	**0.05 (0.02)**
ecoli	0.31 (0.06)	**0.22 (0.08)**	**0.20 (0.07)**	**0.26 (0.08)**
glass	**0.36 (0.07)**	**0.34 (0.11)**	**0.39 (0.11)**	**0.36 (0.10)**
heart-disease	**0.17 (0.02)**	**0.16 (0.03)**	**0.19 (0.02)**	**0.16 (0.04)**
ionosphere	0.13 (0.05)	**0.08 (0.04)**	**0.07 (0.05)**	**0.07 (0.03)**
iris	**0.05 (0.07)**	**0.07 (0.07)**	**0.07 (0.08)**	**0.05 (0.08)**
liver-disorders	**0.32 (0.07)**	**0.26 (0.08)**	**0.26 (0.08)**	**0.28 (0.08)**
diabetes	**0.26 (0.04)**	**0.24 (0.03)**	**0.25 (0.04)**	**0.24 (0.05)**
satimage	0.34 (0.07)	**0.21 (0.02)**	0.23 (0.01)	**0.21 (0.02)**
sonar	0.31 (0.11)	**0.18 (0.09)**	**0.18 (0.09)**	**0.20 (0.13)**
soybean-large	0.73 (0.04)	0.33 (0.05)	**0.27 (0.06)**	0.63 (0.08)
soybean-small	**0.20 (0.13)**	**0.18 (0.09)**	**0.18 (0.08)**	**0.12 (0.11)**
wine	**0.05 (0.06)**	**0.03 (0.04)**	**0.03 (0.04)**	**0.04 (0.04)**

For AdaBoost and most of its related algorithms, stopping early at iteration t_s is identical to the situation where we would have set $T = t_s$. For ValidBoost however, the initial choice of T has some more bearing, because it determines the speed with which the size and weight of the validation sets increases. Table II shows the classification error ValidBoost makes for different choices of T on several UCI datasets with decision stumps as base classifiers. As noted earlier, setting T too low might lead to performance worse than AdaBoost and setting T too high might lead to overfitting not being prevented. We note that for $T \geq 100$, these effects are almost not observable for these datasets, and $T = 1000$ seems to be a sensible choice.

To assess the performance of ValidBoost, we compare it to

TABLE III: Classification error of the boosting algorithms using decision stumps (tree depth 1) after 1024 iterations. Results are obtained using 10-fold crossvalidation. (five times repeated). The best result, and the results not significantly worse, are indicated in bold.

		No noise		
Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer	**0.05 (0.02)**	**0.04 (0.03)**	**0.04 (0.02)**	**0.05 (0.02)**
ecoli	**0.20 (0.07)**	**0.19 (0.05)**	**0.21 (0.05)**	**0.19 (0.06)**
glass	**0.34 (0.09)**	**0.34 (0.09)**	**0.34 (0.09)**	0.39 (0.10)
heart-disease	**0.19 (0.06)**	0.20 (0.06)	**0.17 (0.06)**	**0.17 (0.07)**
ionosphere	**0.07 (0.03)**	**0.08 (0.03)**	**0.08 (0.04)**	**0.07 (0.04)**
iris	0.07 (0.05)	0.06 (0.04)	**0.04 (0.04)**	0.06 (0.05)
liver-disorders	0.29 (0.05)	0.29 (0.05)	**0.28 (0.05)**	**0.26 (0.07)**
diabetes	**0.24 (0.04)**	**0.24 (0.04)**	**0.25 (0.04)**	**0.24 (0.03)**
satimage	0.23 (0.02)	0.22 (0.02)	**0.20 (0.02)**	0.21 (0.02)
sonar	**0.12 (0.09)**	**0.12 (0.08)**	0.20 (0.08)	0.16 (0.08)
soybean-large	0.56 (0.08)	0.62 (0.09)	0.61 (0.10)	**0.33 (0.05)**
soybean-small	0.16 (0.09)	**0.12 (0.08)**	0.22 (0.11)	0.18 (0.09)
wine	0.06 (0.09)	0.07 (0.09)	0.07 (0.07)	**0.03 (0.05)**
		20% noise		
Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer	0.09 (0.04)	0.09 (0.04)	**0.06 (0.02)**	0.07 (0.02)
ecoli	0.33 (0.07)	0.29 (0.07)	**0.22 (0.04)**	**0.21 (0.05)**
glass	0.40 (0.10)	0.43 (0.10)	**0.34 (0.08)**	**0.33 (0.10)**
heart-disease	0.26 (0.07)	0.26 (0.06)	**0.21 (0.07)**	**0.22 (0.09)**
ionosphere	0.25 (0.06)	0.24 (0.07)	**0.12 (0.06)**	**0.13 (0.03)**
iris	0.20 (0.12)	0.18 (0.11)	0.09 (0.07)	**0.05 (0.04)**
liver-disorders	**0.32 (0.06)**	**0.32 (0.05)**	**0.34 (0.05)**	**0.34 (0.06)**
diabetes	**0.26 (0.05)**	0.27 (0.05)	**0.25 (0.03)**	0.26 (0.05)
satimage	**0.16 (0.01)**	**0.15 (0.01)**	0.17 (0.01)	0.17 (0.01)
sonar	0.34 (0.07)	0.35 (0.11)	**0.29 (0.10)**	**0.27 (0.08)**
soybean-large	0.40 (0.09)	0.33 (0.08)	**0.29 (0.07)**	0.34 (0.07)
soybean-small	0.30 (0.10)	0.29 (0.08)	**0.21 (0.13)**	**0.22 (0.12)**
wine	0.15 (0.07)	0.14 (0.06)	**0.09 (0.06)**	**0.10 (0.06)**

AdaBoost*, to LogitBoost [2], and to the method by Bylander & Tate [7] on 13 real-world datasets from the UCI repository ([11]) See appendix A for a complete list. Experiments were done on a few more UCI datasets, but for some datasets no significant performance differences could be found for any setting. This happened for Abalone, Arrhythmia, and Letter-recognition. We compare performance on both the original datasets and the datasets with added label noise, as described in section I. As base classifiers we will use decision trees of varying depth, to account for different levels of complexity. All errors are calculated with 10-fold cross-validation.

In Table III the classification errors are shown for the four approaches on the collection of UCI datasets. On the top half, the original datasets are used, without noise. In the bottom part 20% label noise is added. The base classifier is in all situations a decision stump (or a decision tree with depth 1). Most of the original UCI datasets are relatively clean, and in most situations AdaBoost does not overfit. In these situations it does not pay off to introduce extra regularization to avoid overfitting. Only in soybean-large there is a significant improvement possible. The picture changes completely when noise is added. Then both the method of Bylander & Tate, and ValidBoost often significantly outperform AdaBoost or

*For multi-class ($c > 2$) datasets, we use SAMME ([10]) instead of the original AdaBoost.M1, with its less strict requirements on ϵ ($\epsilon < 1 - \frac{1}{c}$ instead of $\epsilon < \frac{1}{2}$). With decision stumps as base classifiers, this is more or less a requirement, because our hypothesis will only output 2 different classes and the error of a single base classifier will never be lower than $1 - 2/c$ in a dataset with equal prior probabilities for each class. This change is applied to all other algorithms as well.

TABLE IV: Classification error of the boosting algorithms using tree depth 8 after 1024 iterations. Results are obtained using 10-fold crossvalidation (five times repeated). The best result, and the results not significantly worse, are indicated in bold.

	20% noise			
Datasets	AdaBoost	LogitBoost	Bylander & Tate	ValidBoost
breast-cancer	0.11 (0.04)	0.11 (0.04)	**0.09 (0.04)**	0.11 (0.04)
ecoli	**0.18 (0.06)**	**0.17 (0.07)**	**0.18 (0.06)**	**0.18 (0.05)**
glass	0.29 (0.10)	**0.27 (0.10)**	0.35 (0.12)	**0.25 (0.10)**
heart-disease	**0.27 (0.07)**	**0.27 (0.06)**	**0.30 (0.10)**	**0.28 (0.08)**
ionosphere	0.20 (0.07)	0.21 (0.08)	0.22 (0.07)	**0.16 (0.07)**
iris	**0.18 (0.10)**	0.19 (0.10)	**0.18 (0.10)**	**0.15 (0.10)**
liver-disorders	**0.36 (0.07)**	**0.38 (0.08)**	**0.37 (0.08)**	**0.35 (0.08)**
diabetes	0.30 (0.04)	0.30 (0.04)	**0.27 (0.04)**	0.31 (0.05)
satimage	**0.09 (0.01)**	**0.09 (0.01)**	0.09 (0.01)	0.10 (0.01)
sonar	0.38 (0.11)	0.38 (0.09)	0.35 (0.08)	**0.27 (0.08)**
soybean-large	0.24 (0.10)	0.25 (0.11)	**0.18 (0.10)**	**0.20 (0.09)**
soybean-small	0.28 (0.11)	0.29 (0.09)	**0.28 (0.13)**	**0.23 (0.11)**
wine	0.20 (0.11)	0.19 (0.09)	0.20 (0.10)	**0.09 (0.09)**

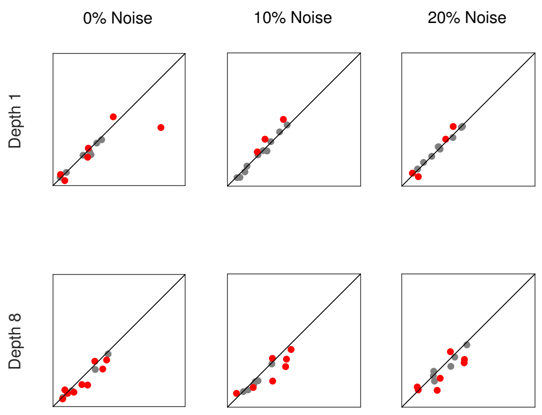

Fig. 6: Plots of ValidBoost errors (vertical axes) versus Bylander & Tate's errors (horizontal axes) on UCI datasets for different levels of noise and decision trees of different depths as base classifiers. All axes go from 0 to 0.75 error rate. Statistically significant differences are represented by the red dots.

LogitBoost.

To see more difference between the method of Bylander & Tate and ValidBoost, the decision trees with depth 8 can be used as base classifiers. The results are shown in Table IV. Here in Sonar, Wine and Ionosphere we see significant differences. These are the relatively high dimensional dataset, with a relatively small sample size. On these datasets the more flexible decision tree with depth 8 can easily overfit, which ValidBoost effectively avoids. On soybean-large one may also expect this behavior, but here the number of iterations T may be a bit too low for ValidBoost.

Figure 6 show the results of ValidBoost compared to Bylander & Tate's method [7]. The scatter plots show the pairwise errors of the two methods. When the two methods perform equally well on some UCI dataset, a dot appears on the diagonal line. We observe that especially with more complex base classifiers (decision trees at depth 8), ValidBoost compares favorably. In the other cases, neither seems to have an edge over the other.

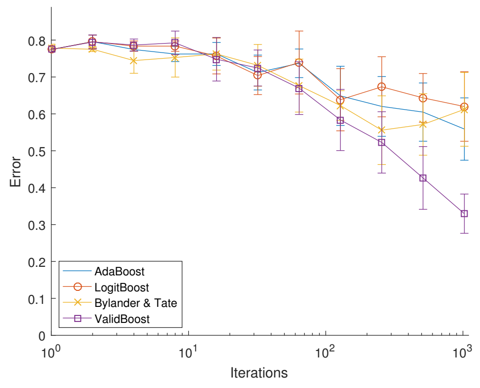

Fig. 7: Behavior on the Soybean-Large dataset with no added noise and tree depth 1.

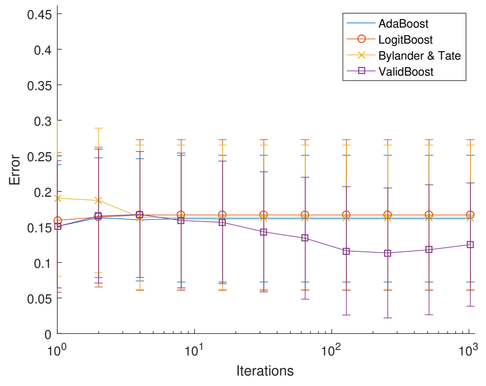

Fig. 8: Behavior on the Soybean-Small dataset with no added noise and tree depth 8.

Figure 7 shows the classification errors of the four compared algorithms on the Soybean-large dataset with no added noise and decision stumps as base classifiers at different intermediate iterations. All four methods perform similarly up to about iteration 128, after that ValidBoost keeps reducing its classification error while the other three methods seem to stagnate. It could be argued that Bylander & Tate's method even seems to start overfitting at that point, as it was initially performing better than AdaBoost and LogitBoost, but starts increasing in error again, eventually ending up with an error similar to AdaBoost and LogitBoost.

A similar but perhaps even more telling effect is observed on the Soybean-small dataset. Figure 8 shows the classification error on the Soybean-small dataset with no added noise and decision trees of depth 8 as base classifiers. When we observe AdaBoost, it seems little can be gained from boosting in this case, as after only four iterations the classification error no longer changes. This indicates that at this point, the base classifier that is found has an error in excess of the maximum error threshold for the multi-class case ($\epsilon > 1 - 1/c$). As such, the base classifier gets assigned a weight of zero, but in the next iteration, this exact same base classifier is found and every base classifier from this point on receives a weight of zero. ValidBoost however, keeps decreasing its error even after this point. We attribute this effect to the random sampling into training set and validation set that ValidBoost does. If a base classifier has a weight in excess of the threshold for the multi-class case, this does not mean we should stop the training process, because it is very likely we will not find the same base classifier in future iterations. In this case, we can see that this leads to a visible improvement.

V. Discussion

In this paper an adapted version of AdaBoost is proposed, that avoids the overfitting of AdaBoost. It has been shown that poor estimates of the error of the base classifiers lead to poor weights for the base classifiers and poor object weights. This in turn leads to an unwarranted focus on some objects with heigh weight. In particular when many noisy objects are present in the training set, the performance of the AdaBoost deteriorates significantly.

The proposed algorithm ValidBoost incorporates two adaptations to the basic AdaBoost algorithm. First, it extracts a validation set from the training set for a more reliable error estimate. Instead of estimating the error on the training set only, it uses a weighted combination of the training and validation error. Second, it increases the size of the validation set during the boosting iterations, and the relative weight of the validation error to the final error estimate. These two adaptations result in both a rapid error decrease during the boosting rounds, and the avoidance of overfitting on noisy data.

When data is well-sampled and clean, i.e. there are not many outlier objects, the differences between the standard AdaBoost, LogitBoost, Bylander & Tate and the proposed ValidBoost are small. Significant differences appear when the classification problem becomes hard: in high dimensional feature spaces, with noise and more flexible base classifiers. Experiments show that this occurs when some classes in a classification problem are small, and when boosting algorithms stop their boosting updates because their base classifiers obtain very high errors. In these situations ValidBoost is able to avoid overfitting and keep adding informative base classifiers to the ensemble.

An open issue is the stronger dependence of ValidBoost to the choice of T, the number of boosting rounds. This dependence of ValidBoost on T is stronger than the other boosting algorithms because both the size of the validation set and the computation of the error of the base classifier depends on this. When the number of boosting rounds is set too low, ValidBoost does not have enough time to converge. Experiments show that for most datasets $T = 1024$ is sufficient, although for some datasets it may be set a bit higher.

References

[1] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[2] J. Friedman, T. Hastie, R. Tibshirani *et al.*, “Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors),” *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[3] A. J. Grove and D. Schuurmans, “Boosting in the limit: Maximizing the margin of learned ensembles,” in *AAAI/IAAI*, 1998, pp. 692–699.

[4] D. Mease and A. Wyner, “Evidence contrary to the statistical view of boosting,” *The Journal of Machine Learning Research*, vol. 9, pp. 131–156, 2008.

[5] R. A. Servedio, “Smooth boosting and learning with malicious noise,” *The Journal of Machine Learning Research*, vol. 4, pp. 633–648, 2003.

[6] P. M. Long and R. A. Servedio, “Random classification noise defeats all convex potential boosters,” *Machine Learning*, vol. 78, no. 3, pp. 287–304, 2010.

[7] T. Bylander and L. Tate, “Using validation sets to avoid overfitting in adaboost.” in *FLAIRS Conference*, 2006, pp. 544–549.

[8] J. Torres-Sospedra, C. Hernández-Espinosa, and M. Fernández-Redondo, “Improving adaptive boosting with k-cross-fold validation,” in *Intelligent Computing*. Springer, 2006, pp. 397–402.

[9] L. Mason, J. Baxter, P. Bartlett, and M. Frean, “Boosting algorithms as gradient descent in function space.” NIPS, 1999.

[10] J. Zhu, H. Zou, S. Rosset, and T. Hastie, “Multi-class adaboost,” *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[11] M. Lichman, “UCI machine learning repository,” 2013. [Online]. Available: http://archive.ics.uci.edu/ml

Appendix A
UCI Datasets

Dataset	**Objects**	**Dimensions**	**Classes**
breast-cancer-wisconsin	699	9	2
ecoli	336	7	8
glass	214	9	4
heart-disease	297	13	2
ionosphere	351	34	2
iris	150	4	3
liver-disorders	345	6	2
pima-indians-diabetes	768	8	2
satimage	6435	36	6
sonar	208	60	2
soybean-large	266	35	15
soybean-small	136	35	4
wine	178	13	3

"All models are wrong,
but some are useful."

– George E. P. Box