Boosting in PN Spaces

Martin Scholz

Artificial Intelligence Group, University of Dortmund, Germany
scholz@ls8.cs.uni-dortmund

Abstract. This paper analyzes boosting in unscaled versions of ROC spaces, also referred to as PN spaces. A minor revision to AdaBoost's reweighting strategy is analyzed, which allows to reformulate it in terms of stratification, and to visualize the boosting process in nested PN spaces as known from divide-and-conquer rule learning. The analyzed confidence-rated algorithm is proven to take more advantage of its base models in each iteration, although also searching a space of linear discrete base classifier combinations. The algorithm reduces the training error quicker without lacking any of the advantages of original AdaBoost. The PN space interpretation allows to derive a lower-bound for the area under the ROC curve metric (AUC) of resulting ensembles based on the AUC after reweighting. The theoretical findings of this paper are complemented by an empirical evaluation on benchmark datasets.

1 Introduction

Boosting is one of the most popular learning techniques in practice, but not yet fully understood in terms of its selection metrics and convergence behavior. The classical AdaBoost algorithm [1] has been presented more than one decade ago, but is still on the agenda of research.

This paper shows how boosting translates into ROC spaces, more precisely into their unscaled counterparts which are referred to as *PN spaces*. ROC analysis provides a unifying framework for studying the behavior of different evaluation metrics [2], for illustrating how to handle class skews, asymmetric misclassification costs, and how to correctly choose a confidence threshold for soft classifiers in different settings [3]. Moreover, the area under the ROC curve (AUC) is the standard machine learning metric for the ranking performance of soft classifiers.

This paper analyzes a revised version of AdaBoost, which is basically subsumed by the framework of confidence-rated boosting [4]. The adapted algorithm allows for a simplified illustration in PN spaces. As its main advantage it allows AdaBoost to take more advantage of its base models, generally increasing the learning rate and generalization performance of boosting. The resulting algorithm still searches a linear combination of crisp base classifiers and can be reformulated in simpler terms, as to continuously stratify the target attribute.

The aim of the analysis is to foster a better understanding of the implicit AUC optimization property of boosting. Original AdaBoost's excellent ranking behavior has just recently been explained by showing its similarity to RankBoost

if equal loss is suffered from positive and negative examples [5]; another technical proof exists for Real AdaBoost [4]. This paper contributes an intuitive and much simpler proof of a tighter ranking error (AUC) bound for Real AdaBoost-like ensemble classifiers, derived from a geometric interpretation in PN spaces.

2 Formal framework and basic properties

ROC analysis has become a popular tool for analyzing classifier performances and to study evaluation metrics [6]. The unscaled counter-part, PN spaces [2], are well-suited to visualize the nested subspaces of divide-and-conquer rule learning. The only difference to ROC spaces is that the axes of PN spaces show the absolute numbers of positives and negatives, while in ROC spaces both axes are scaled to the range of $[0, 1]$. This paper confines itself to boolean classification problems, so models are functions mapping an instance space $\mathcal{X}$ to a boolean target label $\mathcal{Y} = \{+1, -1\}$. The notation used in this paper is chosen to be similar to the one used in [2] for rule learning in PN spaces.

Definition 1. *For a model $h : \mathcal{X} \to \mathcal{Y}$ and a given data set $\mathcal{E}$ with an absolute number of P positive examples and N negative examples, the absolute number of true positives is denoted as p, the number of false positives as n. Analogously, the absolute number of false negatives is denoted as $\overline{p}$, the absolute number of true negatives as $\overline{n}$.*

Fig. 1 shows nested PN spaces from specific to general as obtained when adding a single rule to a decision list in each iteration. The $p+n$ examples for which the rule applies are removed from further consideration, and the remaining examples are represented by nested rectangles of shrinking size. Analogously, refining a rule adding one literal at a time shrinks the covered subsets from general to specific.

ROC spaces can be considered to show *stratified* versions of PN spaces; axes represent classes and are normalized to the same scale. The term stratification will reoccur in this paper. It can be interpreted as the process of altering class proportions in the training set, so that all classes are equally frequent. It is a common preprocessing step in the machine learning literature, e.g. for training classifiers under skewed class distributions or varying misclassification costs [3].

Stratification can be realized by reweighting or by subsampling. The goal in the former case is to obtain the same total example weight for each class. To this end, all examples sharing a class receive a common weight, chosen inverse proportionally to the frequency of the class in the training set. In the latter case one samples with equal probability from each class, hence implicitly samples from another than the i.i.d. distribution underlying the training data. The following definition captures the resulting implicit new target distribution.

Definition 2. *For a distribution $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ over an instance space $\mathcal{X}$ and a target label $\mathcal{Y}$ the stratified random sample distribution D' of D is*

$$D'(x, y) := \frac{D(x, y)}{|\mathcal{Y}| \cdot P_{(x',y')\sim D}(y = y')}.$$

A typical application of *ROC* analysis in machine learning is to visualize soft classifiers, that yield continuous confidence scores for examples being positive. Each crisp classifier obtained by applying a threshold to a soft classifier is represented in a ROC diagram as a point depicting the resulting true positive rate p/P and false positive rate n/N [3]. The area under the resulting graph is referred to as the *area under the ROC curve (AUC)*, which equals the probability that a randomly selected positive example is ranked higher (higher confidence) than a randomly selected negative example. Maximizing the AUC is a learning task of its own right and has also been shown to lead to a competitive but more robust selection of models regarding maximization of predictive accuracy [7].

In this paper a quantity closely related to the AUC is analyzed, the area *over* the curve. An asterisk indicates quantities in PN space rather than ROC space.

Definition 3. *For a given soft classifier and example set $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ of positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$ the* area over the curve in PN spaces (AOC*) *is defined as the number of misranked tuples $(e^+, e^-) \in \mathcal{E}^+ \times \mathcal{E}^-$, that is the number of pairs for which e^- is predicted positive with higher confidence than e^+. Example pairs (e^+, e^-) with associated weights w^+ and w^- are accounted for by $w^+ \cdot w^-$ misranks.*

Ties are considered to be broken randomly, so half of all equally ranked pairs are considered to be misranked. It is easily seen that $AOC^* = (1 - AUC) \cdot P \cdot N$ for the unweighted case, which is a special case of the weighted case with $w^{+/-} := 1$. An example with a weight of w naturally represents an example *set* of size w.

The main diagonal in ROC/PN space represents the performances of default classifiers and random classifiers that do not incorporate any data at all, but predict $y = +1$ with a fixed probability. The AOC* of such uninformed models equals half the area of the corresponding PN space, i.e. $AOC^* = (P \cdot N)/2$. Changing the class proportions clearly does not affect this ranking performance. To preserve fundamental semantics it is hence suggested not to change the AOC* during steps of skewing the data, as it reflects the absolute ranking error. This is further justified at a later point. It translates into the constraint $P' \cdot N' = P \cdot N$ for the new values P' and N' obtained by skewing P and N, respectively.

The learning algorithms used in this paper are assumed to implicitly normalize the training set, so that the weights describe a distribution. Hence, the only quantities of interest for stratification are the class ratios P/N and P'/N'.

Proposition 1. *The reweighting rule for changing the ratio of P/N by a factor c while meeting the constraint $P \cdot N = P' \cdot N'$ is unique:*

$$w'(x, y) := w(x, y) \cdot \begin{cases} \sqrt{c}, \text{ for positive } y \\ \frac{1}{\sqrt{c}}, \text{ for negative } y \end{cases}$$

Proof. It directly follows that after reweighting we have

$$P' = \sqrt{c}P \ , \ N' = \frac{N}{\sqrt{c}} \ , \ \frac{P'}{N'} = c \ , \ P' \cdot N' = P \cdot N.$$

Multiplying positives with a constant of c' requires to divide negatives by the same constant to satisfy the constraint. Only $c' = \sqrt{c}$ is valid, since $P'/N' = c'^2$.

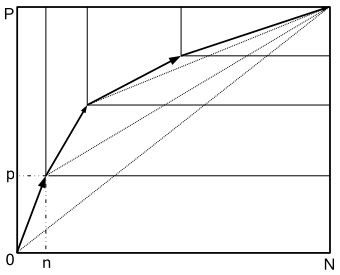

Fig. 1. Nested PN-Spaces

Initialize weights $w_1(x_i, y_i) := 1$ for $(x_i, y_i) \in \mathcal{E}$
for $t = 1$ to k **do**
  $h_t \leftarrow$ `base learner`$(\mathcal{E}, w_t)$
  Compute $\epsilon_t := \sum_{i=1}^{|\mathcal{E}|} w_t(x_i, y_i) I[h_t(x_i) \neq y_i]$
  Let $\alpha_t := \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
  $w_{t+1}(x_i, y_i) := w_t(x_i, y_i) \cdot \exp(-y_i \alpha_t h_t(x_i))$
end for
Output classifier: Predict sign $\left(\sum_{i=1}^{k} \alpha_t h_t(x)\right)$

Fig. 2. AdaBoost for $y \in \{+1, -1\}$

As required, the weighting does not change the AOC* (Def. 3). Stratification is a specific case of skewing the data, leading to equal class proportions. It has a further important property in the context of boosting, as will be shown in Sec. 3.

Proposition 2. *Among all skewing operations preserving $P \cdot N$, stratifying the data by choosing $c = N/P$ leads to the minimal total example weight of $2\sqrt{PN}$.*

Proof. For valid reweightings we have $P' \cdot N' = P \cdot N$. The weight to be minimized is $P' + N' = \sqrt{c}P + N/\sqrt{c} =: f(c)$. Setting the derivative to 0 yields the result.

3 Boosting

3.1 AdaBoost

Combining individual classifiers to weighted ensembles is an effective way to increase predictive accuracy and other metrics like the AUC. The best known, most studied, and probably the most frequently applied ensemble method is AdaBoost [1], depicted in Fig. 2. It fits a sequence of base models $h_t : \mathcal{X} \to \mathcal{Y}$, each to a reweighted version of the training set, or to an analogously constructed subsample, respectively. The term $I[\cdot]$ used in the algorithm refers to the indicator function. To simplify subsequent analysis the algorithm is formulated without the step of normalizing weights, which is left to the base learner.

The reweighting scheme of AdaBoost gives higher weight to the "hard" examples of the training set, and finally predicts based on a weighted majority vote. A different perspective has been fostered in [8], pointing out AdaBoost's similarity to additive logistic regression. From an optimization point of view AdaBoost fits into the broader AnyBoost framework [9], as it performs gradient descent in function space in order to minimize the exponential loss function $\exp(-y_i \sum_{t=1}^{k} \alpha_t h_t(x_i))$. For a (weighted) error rate of ϵ_t of the base classifier in iteration t and $\beta_t(x) := (1-\epsilon_t)/\epsilon_t = \exp(2\alpha_t)$ the reweighting strategy computes

$$w_{t+1}(x, y) = w_t(x, y) \cdot \exp(-y \alpha_t h_t(x_i)) = \prod_{t=1}^{k} (\sqrt{\beta_t})^{-y \cdot h_t(x)} \qquad (1)$$

as the new weight for each example (x, y), starting with uniform weights. All examples with a final weight w_{t+1} of less than 1 are classified correctly, since

$$\sum_{t|h_t(x)=y} \alpha_t > \sum_{t|h_t(x)\neq y} \alpha_t \Leftrightarrow 1/2 \left(\sum_{t|h_t(x)=y} \ln \frac{1-\epsilon_t}{\epsilon_t} - \sum_{t|h_t(x)\neq y} \ln \frac{1-\epsilon_t}{\epsilon_t} \right) > 0$$

$$\Leftrightarrow \prod_{t=1}^{k} \sqrt{\beta_t}^{(y \cdot h_t(x))} > 1 \;\Leftrightarrow\; w_{t+1}(x, y) < 1.$$

In turn, examples with a weight of greater than 1 are misclassified. For this reason one of the most important properties of AdaBoost is that it reduces the total weight quickly if the base learner provides useful classifiers h_t.

3.2 Ada^{2}Boost

One disadvantage of AdaBoost is that it does not take full advantage of its base models. For illustration we consider a classification rule covering significantly less than half of the examples (respecting weights), but having a low error rate for this subset. Such a model is generally useful for ensemble learning. However, it is not necessarily useful for AdaBoost, because the error rate of the large uncovered part might be significantly higher, resulting in a value of $\alpha_t \approx 0$.

Such asymmetric cases can be handled by using separate estimates of the error rate for the *covered* part $\mathcal{C}_t := \{(x, y) \in \mathcal{E} | h_t(x) = +1\}$ and the *uncovered* part $\overline{\mathcal{C}}_t := \{(x, y) \in \mathcal{E} | h_t(x) = -1\}$. Both local error rates, denoted as

$$\epsilon^+ := n/(p+n) \text{ for } \mathcal{C}_t, \text{ and } \epsilon^- := \overline{p}/(\overline{p}+\overline{n}) \text{ for } \overline{\mathcal{C}}_t$$

can easily be computed from the contingency matrix and will usually differ. Please note, that for $\overline{\mathcal{C}}_t$ the *negative* examples are the correctly classified ones. We will replace the static values of β_t by functions $\beta_t(h_t(x))$ that depend only on the prediction of their corresponding base model $h_t(x) \in \{+1, -1\}$. This leads to two separate factors, the odds ratio for $\mathcal{C}_t$, and the inverse odds ratio for $\overline{\mathcal{C}}_t$:

$$\beta(+1) := \frac{1-\epsilon^+}{\epsilon^+} = \frac{p}{n} \;, \qquad \beta(-1) := \frac{1-\epsilon^-}{\epsilon^-} = \frac{\overline{n}}{\overline{p}} \tag{2}$$

With $\alpha(h(x)) := (\ln \beta(h(x))/2$ the weight update of AdaBoost translates into:

$$w_{t+1}(x_i, y_i) := w_t(x_i, y_i) \cdot \exp\left[-y_i \cdot \alpha_t(h_t(x_i)) \cdot h_t(x_i)\right]$$

The rule for predicting a label $\hat{y} \in \{+1, -1\}$ is changed accordingly:

$$\hat{y} := \text{sign}\left(\sum_{t=1}^{k} \alpha_t(h_t(x)) h_t(x) \right) \tag{3}$$

This adapted version of AdaBoost is referred to as *Ada^{2}Boost* in this paper. It is only analyzed in combination with plain boolean base classifiers, which does not

require regression-capabilities of base learners, as e.g. LogitBoost [8] that uses working responses and weights at the same time.

Ada^{2}Boost is similar to the confidence-rated Real AdaBoost [4], which allows for continuous predictions $h_t : \mathcal{X} \to \mathbb{R}$. Real AdaBoost reweights examples using the same rule as shown for AdaBoost, but the more general setting of continuous functions h_t requires to optimize α_t "manually" (not based on ϵ_t) to minimize the total example weight. The prediction rule is identical to the one shown in Fig. 2. If each h_t takes only values from $\{-1, +1\}$ the choice of asymmetric model weights made by Ada^{2}Boost reduces weights optimally. Differences to Real AdaBoost are that Ada^{2}Boost (i) uses boolean *crisp* base classifiers, adding confidence-like scores as part of the boosting procedure, and (ii) that it incorporates confidence ratings only in a very moderate form, which constrains the potential to overfit to the training data; when using the same fixed number of base models, Ada^{2}Boost selects ensemble models from almost the same search space as AdaBoost:

Proposition 3. *If estimated error rates are bounded away from zero the search space of AdaBoost and Ada^{2}Boost for boolean classification tasks are identical up to a constant additive offset.*

Proof. A model of the form given by eqn. (3) can be transformed into another classifier of the form

$$\hat{y} := \operatorname{sign}\left(\alpha'_0 + \sum_{t=1}^{k} \alpha'_t h_i(x)\right), \tag{4}$$

with offset α'_0 and model weights $\alpha'_1, \ldots, \alpha'_k \in \mathbb{R}$ by computing for each model

$$\operatorname{avg}_{0,t} := \frac{\alpha_t(+1) + \alpha_t(-1)}{2}, \qquad \alpha'_t := \alpha_t(+1) - \operatorname{avg}_{0,t} \qquad \alpha'_0 := \sum_{t=1}^{k} \operatorname{avg}_{0,t}.$$

The transformed model (4) is obviously identical to the original model.

Aiming to minimize generalization error it is quite natural to bound the error rates away from 0, e.g. by using Laplace or m-estimates for pure subsets.

Although the difference in expressiveness seems marginal, it allows Ada^{2}Boost to take more advantage of its base models, reflected by quicker weight reduction.

Theorem 1. *If $\epsilon = \epsilon^+ = \epsilon^-$ then the reweighting strategies of AdaBoost and Ada^{2}Boost are identical. Otherwise Ada^{2}Boost reduces the weights more quickly.*

Proof. AdaBoost reweights the $p + \overline{n} = 1 - \epsilon$ correctly classified examples multiplying with $\sqrt{\beta}$, and misclassified examples dividing by the same term. Hence the total weight $W_{t+1} = \sum_{i=1}^{|\mathcal{E}|} w_{t+1}(x_i, y_i)$ in iteration $t+1$ can be computed as

$$W_{t+1} = \frac{1}{\sqrt{\beta}}((1-\epsilon)W_t) + \sqrt{\beta}(\epsilon W_t) = 2W_t\sqrt{\epsilon \cdot (1-\epsilon)} = 2W_t\sqrt{(p+\overline{n})(\overline{p}+n)}.$$

Ada^{2}Boost reweights the $p+n$ covered examples ($\mathcal{C}$) multiplying with $\sqrt{\beta(+1)}^{(\pm 1)}$ Since positives are divided by and negatives are multiplied with $\sqrt{p/n}$ the weight

of $\mathcal{C}$ reduces from $W_t \cdot (p+n)$ to $W_t \left(p/\sqrt{p/n} + n\sqrt{p/n}\right) = 2W_t\sqrt{p \cdot n}$. The weight of $\overline{\mathcal{C}}$ changes analogously, applying the factor $\sqrt{\beta(-1)}^{(\pm 1)} = \sqrt{\overline{n}/\overline{p}}^{(\pm 1)}$ instead, so the new total weight for Ada^{2}Boost is $W_{t+1} = 2W_t \cdot \left(\sqrt{p \cdot n} + \sqrt{\overline{p} \cdot \overline{n}}\right)$. We hence need to show $\sqrt{(p+\overline{n})(\overline{p}+n)} \geq \sqrt{p \cdot n} + \sqrt{\overline{p} \cdot \overline{n}}$, which follows from

$$\sqrt{(p+\overline{n})(\overline{p}+n)} \geq \sqrt{p \cdot n} + \sqrt{\overline{p} \cdot \overline{n}} \Leftrightarrow (p+\overline{n})(\overline{p}+n) \geq pn + \overline{pn} + 2\sqrt{pn\overline{pn}}$$
$$\Leftrightarrow p\overline{p} + n\overline{n} \geq 2\sqrt{pn\overline{pn}} \Leftrightarrow (p\overline{p})^2 + 2pn\overline{pn} + (n\overline{n})^2 \geq 4pn\overline{pn} \Leftrightarrow (p\overline{p} - n\overline{n})^2 \geq 0.$$

Both strategies yield the same result, iff $p\overline{p} = n\overline{n}$. This is equivalent to

$$\frac{p}{n} = \frac{\overline{n}}{\overline{p}} \Leftrightarrow \quad \frac{p/(p+n)}{n/(p+n)} = \frac{\overline{n}/(\overline{n}+\overline{p})}{\overline{p}/(\overline{n}+\overline{p})} \Leftrightarrow \quad \frac{1-\epsilon^+}{\epsilon^+} = \frac{1-\epsilon^-}{\epsilon^-} \Leftrightarrow \epsilon^+ = \epsilon^-$$

If n or $\overline{p}$ are 0, then either *both* error rates need to be 0, or one of the local error rates is undefined, because the subset contains no examples. The fact that ϵ is a weighted average of ϵ^+ and ϵ^- completes the proof.

The following proposition summarizes some useful properties of Ada^{2}Boost.

Proposition 4. *After Ada^{2}Boost reweights for the first time we have $P' = N'$. After each iteration t the subsets $\mathcal{C}_t$ and $\overline{\mathcal{C}}_t$ corresponding to model h_t are both stratified. The error rates ϵ_t^+ and ϵ_t^- of h_t are exactly $1/2$ with respect to w_{t+1}.*

Ada^{2}Boost performs especially well in in the case of conditionally independent base classifiers. Denoting with $P_t(\cdot)$ probabilities based on weights w_t it uses a product of β_t terms as defined in eqn. (2) to compute odds-ratio estimates

$$\widehat{\beta(x)} = \frac{P(y=+1 \mid h_1(x)...h_k(x))}{P(y=-1 \mid h_1(x)...h_k(x))} = \prod_{t=1}^{k} \frac{P_t(y=+1 \mid h_t(x))}{P_t(y=-1 \mid h_t(x))} \tag{5}$$

This happens to be identical to the estimate of NaïveBayes on top of the base model predictions $h_t(x)$, which yields the Bayes' optimal decision rule in this setting. This simple interpretation requires discrete prediction domains for base models. Even in cases where conditional independence is lacking Ada^{2}Boost continuously fits an additive model to the log-odds, similar to logistic regression, which suggests that it may yield good estimates of conditional class distributions.

3.3 An analysis in PN spaces

The reweighting strategy of Ada^{2}Boost is very similar to the stratification proposed in Sec. 2. In each iteration both the covered ($\mathcal{C}$) and the uncovered subsets ($\overline{\mathcal{C}}$) are stratified in the sense of Prop. 1.

These results suggest a reformulation of boosting in terms of stratification. The use of exponential or logarithmic functions (α_t values) seems to be unnecessarily complicated in this setting, because Ada^{2}Boost only requires the more intuitive β_t values. As a further simplification the algorithm uses β'_t, which *always* refers to the odds ratio, while β_t refers to the inverse odds ratio whenever a

```
// Init with uniform weights:
Let w_1(x,y) := 1 for all (x,y) ∈ E
// Train k base classifiers:
for t = 1 to k do
  h_t ← base learner(E, w_t)
  // Compute odds ratios:
  β'_t(+1) := p_t/n_t, β'_t(−1) := p̄_t/n̄_t
  // Stratification:
  w_{t+1}(x,y) := w_t(x,y) / sqrt(β'_t(h_t(x)))^y
end for
Output: β̂(x) = ∏_{t=1}^{k} β'_t(h_t(x))
P̂(y = +1 | x) := β̂(x)/(1 + β̂(x))
```

Fig. 3. Ada^{2}Boost for $y \in \{+1, -1\}$

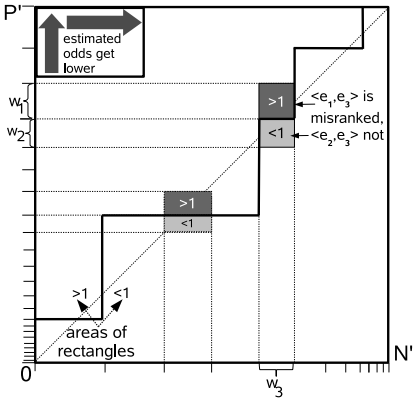

Fig. 4. Illustration for proof of theorem 2.

base classifier predicts negatively. Ada^{2}Boost (Fig. 3) boosts boolean base classifiers in a very simple fashion. In each iteration t another stratified (for $t > 1$, see Prop. 4) example set is presented to the base learner. The learner returns a base classifier $h_t : \mathcal{X} \to \{+1, -1\}$ that partitions the example set into "unstratified" subsets $\mathcal{C}$ and $\overline{\mathcal{C}}$; this automatically happens when maximizing accuracy [10]. The terms p_t to $\overline{n}_t$ denote the true positives to false negatives of model h_t using example weights w_t. For both partitions, $\mathcal{C}$ and $\overline{\mathcal{C}}$, the odds are computed and stored for later predictions, before stratifying the subsets separately, respecting the constraint stated in Prop. 1. When predicting a label the local odds are combined applying eqn. (5), which easily allows to derive probability estimates.

Fig. 5 shows a step of stratification for two partitions, e.g. for a classification rule in PN space. The two rectangles represent the performances of the two dual rules $(h_t(x) = +1) \to (y = +1)$ and $(h_t(x) = -1) \to (y = -1)$, where the slope of the diagonal is $\beta'_t(+1) = p/n$ in the former, and $\beta'_t(-1) = \overline{p}/\overline{n}$ in the latter. Stratification turns each of these rectangles into a square of equal size (hence Ada^2). This can also be thought of as a transformation of the underlying distribution (see Def. 2) that can be inverted precisely when making predictions.

It is interesting to note that the role of the base learner can as well be stated as to divide $\mathcal{E}$ into "unstratified" subsets, which are continuously stratified (conquered) by the meta-algorithm as long as the base learner succeeds.

In divide-and-conquer *rule learning* examples that are covered are removed from subsequent learning iterations. Similarly, boosting can be considered to probabilistically discard examples. Shifting both squares to the upper right, as depicted on the right side of Fig. 5, we reach at a visualization of boosting as nested PN-spaces. The weight lost by this transformation shows as the part of the axes of the original PN space below and left to the embedded PN space. As for AdaBoost, the total weight upper-bounds the number of misclassified examples, because only examples with a weight of greater than 1 are misclassified. Prop. 2 states, that Ada^{2}Boost reduces the total weight as much as possible,

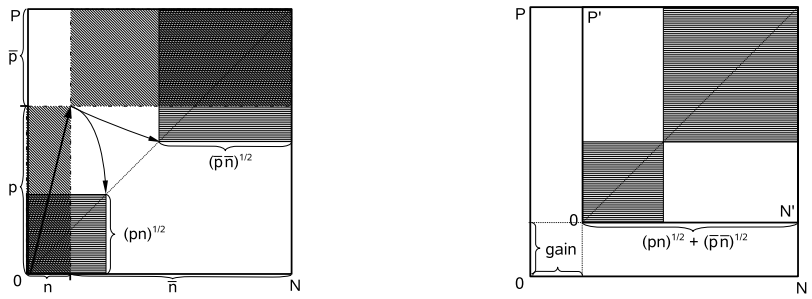

Fig. 5. Ada^{2}Boost transforms the boxes representing $\mathcal{C}$ and $\overline{\mathcal{C}}$ into squares (left) by reweighting. Moving these squares to the upper right (right) yields a PN subspace.

while respecting the constraint to preserve the area of each subset in PN space. The advantage of this constraint is that the areas of the nested PN spaces reflect the progress in minimizing the ranking error at the same time.

Theorem 2. *The absolute ranking error (AOC*) of Ada^{2}Boost ensemble models for the original (unweighted) data is upper-bounded by the AOC* of the model for the inner nested PN space (reweighted example set).*

Proof. The crucial observation is, that final confidences and weights are closely related. A pair of examples (e^+, e^-) with weights w^+ and w^- will be misranked, iff the estimated confidence of being positive is higher for e^- than for e^+. The confidences are monotone in the estimated odds $\hat{\beta}(e^+)$ and $\hat{\beta}(e^-)$. Applying the reweighting scheme of Ada^{2}Boost recursively and computing $\hat{\beta}(x)$ we find that

$$w_{k+1}(x,y) := \prod_{t=1}^{k} \sqrt{\beta'_t(h_t(x))}^{(-y)} \quad \text{and} \quad \hat{\beta}(x) = \prod_{t=1}^{k} \beta'_t(h_t(x))$$

for an ensemble of size k. This implies $w^+ = \sqrt{1/\hat{\beta}(e^+)}$ and $w^- = \sqrt{\hat{\beta}(e^-)}$.

If $\hat{\beta}(e^-) > \hat{\beta}(e^+)$, then we have $(w^-)^2 > 1/(w^+)^2 \Leftrightarrow w^+ \cdot w^- > 1$. This means that each misranked pair (e^+, e^-) "occupies" a rectangle with an area of at least 1 in the inner nested PN space. All rectangles representing different pairs (e^+, e^-) are disjoint. Hence, if the nested PN space has a size of $P' \cdot N'$, then this quantity upper-bounds the AOC* of the ensemble for the original data.

When ordering examples by confidence, as for soft classifier ROC plots, weights of positives ascend along the P' axis, while weights of negatives descend along the N' axis (see Fig. 4). The areas of rectangles representing example pairs grow monotonically towards the upper left corner $(0, P')$. The border where areas become larger than 1 is depicted as a thick line. Example pairs share their estimated odds along the border, since $w^+ \cdot w^- = 1 \Rightarrow \hat{\beta}(e^+) = \hat{\beta}(e^-)$, so a threshold is associated to each point. Apart from the scale, Fig. 4 provides a ROC plot of the ensemble for the *reweighted* example set. By construction we expect the ensemble to perform as good as random guessing after reweighting,

having an AOC* of $(P' \cdot N')/2$. In this case only *half* of the nested PN space represents misclassified pairs (areas ≥ 1), which also halves the AOC* upper-bound for the original data. The same argument applies for any other AOC* score.

The proof does not require base classifiers to provide *boolean* partitionings of $\mathcal{X}$, so theorem 2 holds for Real AdaBoost-like ensemble classifiers in general. The derived bounds are tighter than those provided in [4], based only on the worst case AOC* of $P' \cdot N'$. To the best of the author's knowledge theorem 2 provides the first AUC (AOC*) bound based on ranking performances after reweighting.

Corollary 1. *For an example set $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ a reduction of the initial weight of $|\mathcal{E}|$ to W results in an AUC $\geq 1 - W^2/(8 \cdot |\mathcal{E}^+| \cdot |\mathcal{E}^-|)$ if the AUC of the ensemble is at least $1/2$ (random guessing) for the reweighted example set.*

For AdaBoost nested PN spaces are less intuitive, but also share the semantics of the quantity $P' \cdot N'$. Improved weight reduction strategies imply a more efficient reduction of this quantity, however. This becomes obvious when finally adding a classifier with constant predictions to each AdaBoost ensemble, which just stratifies the example set, so that the weight determines $P' \cdot N'$.

4 Evaluation

This section empirically evaluates generalization performances of the previously analyzed metrics on 4 benchmark datasets taken from the UCI library [11], and on a 10k sample of the quantum physics datasets from KDD Cup 2004. The evaluated metrics are accuracy (ACC), the area under the ROC curve (AUC), and finally, since evidence has been provided that Ada^{2}Boost may perform well in estimating conditional class probabilities, the root mean squared error (RMSE).

Ada^{2}Boost was implemented in YALE[1] [12] without any optimizations like LaPlace estimates. The Weka library [13] provides AdaBoost and decision stumps as base learners. Decision stumps are popular base classifiers and do not apply a greedy search themselves, which eases the evaluation of greedily operating boosting techniques. In the proposed experimental setting Real AdaBoost with "reasonable" confidence ratings for each decision stump yields the same predictions as Ada^{2}Boost. The focus of the evaluation is on AUC maximization and its relation to ACC optimization; for error rate minimization refer to [4] or [14].

Fig. 6 shows the learning curves for different numbers of base models. Each point in the plots is the result of a ten-fold cross-validation. The results illustrate that boosting in fact maximizes all three considered metrics simultaneously, with AUC and RMSE providing finer-grained indicators of progress than ACC. Moreover, the moderate adaptation of AdaBoost does not only improve the ACC learning rate, as shown earlier, but leads to similar improvements for the metrics AUC and RMSE. The difference between AdaBoost and Ada^{2}Boost for Credit-G containing very few examples *and* attributes (16) are the smallest (if any), lying within half a standard deviation. For adult (32K examples) improvements are

[1] `http://yale.sf.net/`

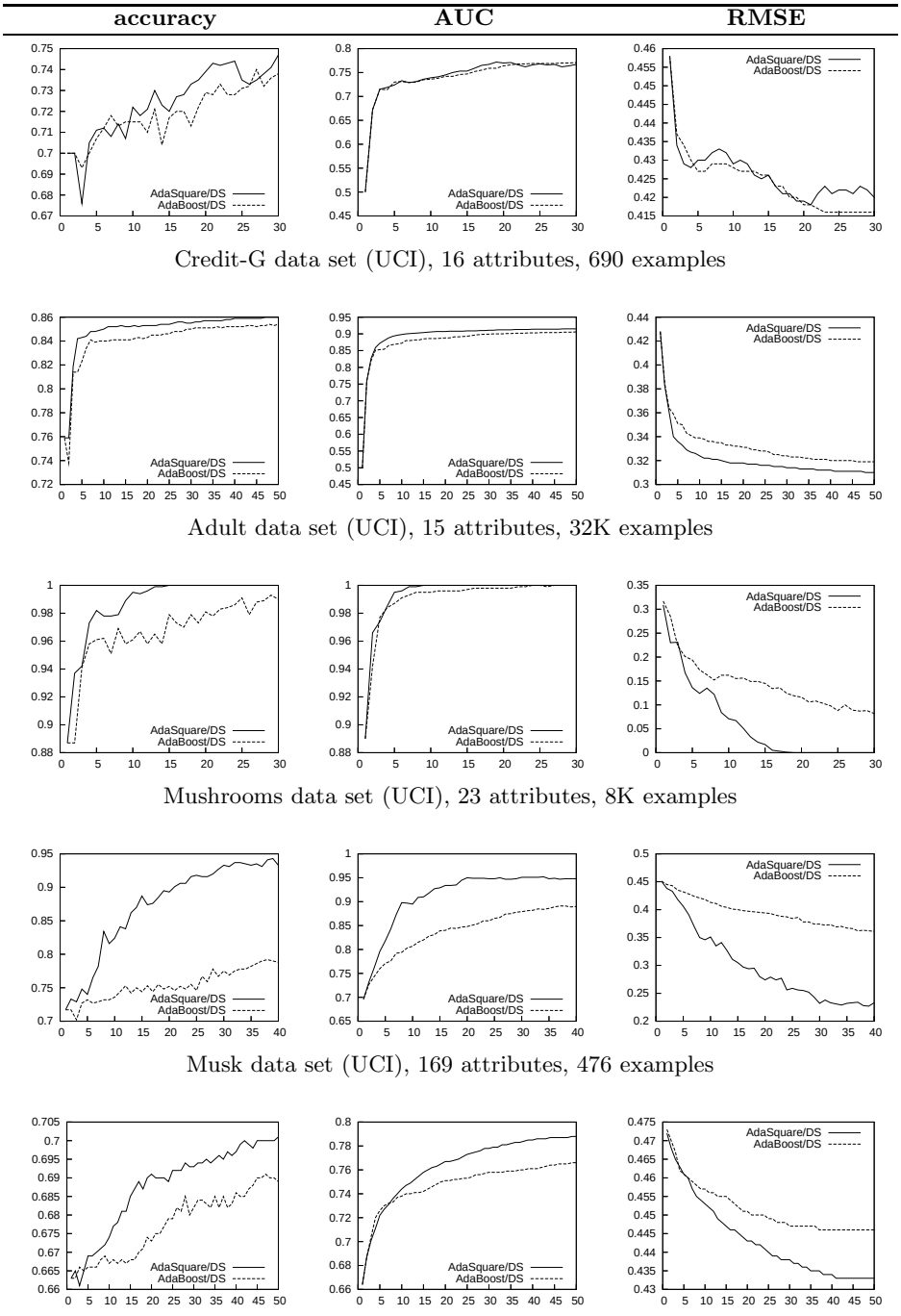

Fig. 6. Generalization performances: AdaBoost vs. Ada^{2}Boost for decision stumps.

small but significant: For all 3 metrics and e.g. 10 or 50 stumps it passes a t-test at a level of 2%. Advantages are much clearer for the remaining 3 data sets. On mushrooms, Ada^{2}Boost produces a perfect ranking with only 9, and perfect *soft* predictions with 19 stumps. In contrast, AdaBoost requires 24 stumps to rank perfectly and 100 stumps to reach an RMS of 2%. The curves differ most drastically for musk, having few examples but 169 attributes. The monotonicity of the AUC plots, well visible e.g. for the KDD Cup data, reflects the high robustness of this metric. AUC and RMSE behave similarly for all data sets.

5 Conclusions

A simplified confidence-rated AdaBoost variant based on stratification was analyzed. Visualizing the boosting process in nested PN spaces allowed to point out similarities between boosting and rule learning. Theoretical results have been provided that ease to understand (i) why boosting with accuracy as the objective function implicitly maximizes the AUC, and (ii) why confidence-rated strategies perform even better. Finally, a tighter than the commonly known bound for the AUC of boosting ensembles has been derived from a PN space analysis. An empirical study confirmed the results on implicit AUC maximization, indicating similar benefits for minimizing the root mean squared error.

References

1. Freund, Y., Schapire, R.R.: A decision–theoretic generalization of on-line learning and an application to boosting. Computer and System Sciences **55**(1) (1997)
2. Fürnkranz, J., Flach, P.: ROC 'n' Rule Learning – Towards a Better Understanding of Covering Algorithms. Machine Learning **58**(1) (2005)
3. Fawcett, T.: ROC Graphs: Notes and Practical Considerations for Researchers Tech report HPL-2003-4. HP Laboratories, Palo Alto, CA, USA (2004)
4. Schapire, R.E., Singer, Y.: Improved Boosting Using Confidence-rated Predictions. Machine Learning **37**(3) (1999)
5. Rudin, C., Cortes, C., Mohri, M., Schapire, R.E.: Margin-Based Ranking Meets Boosting in the Middle. In Proc. of COLT (2005)
6. Flach, P.A.: The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics. In Proc. of ICML (2003)
7. Rosset, S.: Model Selection via the AUC. In Proc. of ICML (2004)
8. Friedman, J.H., Hastie, T., Tibshirani, R.: Additive logistic regression: A statistical view of boosting. Annals of Statistics (28) (2000)
9. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent in function space. Technical report, Australian National University (1999)
10. Scholz, M.: Sampling-Based Sequential Subgroup Mining. In Proc. of KDD (2005)
11. Blake, C., Merz, C.: UCI Repository of machine learning databases (1998)
12. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In Proc. of KDD (2006)
13. Witten, I., Frank, E.: Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (2000)
14. Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In Proc. of ICML (1999)