Efficient AUC Optimization for Classification

Toon Calders[1] and Szymon Jaroszewicz[2]

[1] Eindhoven University of Technology, the Netherlands
[2] National Institute of Telecommunications, Warsaw, Poland

Abstract. In this paper we show an efficient method for inducing classifiers that directly optimize the area under the ROC curve. Recently, AUC gained importance in the classification community as a mean to compare the performance of classifiers. Because most classification methods do not optimize this measure directly, several classification learning methods are emerging that directly optimize the AUC. These methods, however, require many costly computations of the AUC, and hence, do not scale well to large datasets. In this paper, we develop a method to increase the efficiency of computing AUC based on a polynomial approximation of the AUC. As a proof of concept, the approximation is plugged into the construction of a scalable linear classifier that directly optimizes AUC using a gradient descent method. Experiments on real-life datasets show a high accuracy and efficiency of the polynomial approximation.

1 Introduction

In binary classification, often, the performance of classifiers is measured using the Area under the ROC Curve (AUC). Intuitively, the AUC of a classification function f expresses the probability that a randomly selected positive example gets a higher score by f than a randomly selected negative example. This measure has proven to be highly useful for evaluating classifiers, especially when class distributions are heavily skewed.

Recently, several new classifier training techniques have been developed that directly optimize the AUC. The main problem these algorithms face is that computing the AUC is a relatively costly operation: it requires sorting the database, a cost of order $n \log(n)$ for a database of size n. Also, in contrast to, e.g., the mean squared error, the AUC is not continuous on the training set, which makes the optimization task even more challenging. Therefore, often the algorithms optimize a slight variant of the AUC, that is differentiable. We denote this variant *soft-AUC*. The complexity of computing this soft-AUC, however, is even worse: it is of order n^2 for a database of size n. These high computational demands of AUC and soft-AUC seriously impact the scalability of these methods to large databases. Most of these algorithms therefore rely on sampling.

In this paper we present another option, namely the use of polynomial approximations for the AUC and the soft-AUC. The polynomial approximation has the advantage that it can be computed in only one scan over the database, and hence, it does not require resorting the database every time the AUC for a

new or updated classification function is needed. Furthermore, when the classification function is only slightly changed, it is even possible to find the new AUC without a database scan, based on a small summary of the database.

We show experimentally that the polynomial approximation is very accurate and extremely efficient to compute; for soft-AUC, the traditional methods are already outperformed starting from a couple of hundred of tuples. Furthermore, the computation of the AUC can be plugged into all methods requiring repeated computations of the AUC. As a proof of concept, the approximation is plugged into a gradient descent method for training a linear classifier. It was implemented and tested on real-life datasets. With the approximation technique, similar AUC scores were reached as with existing techniques. The scalability and running times of the proposed approximation technique, however, were vastly superior.

To summarize, the main contribution of this paper is the development of an efficient procedure to approximate AUC and soft-AUC that scales very well with the size of the dataset. This method makes it possible to scale-up existing algorithms that optimize AUC directly.

2 Area Under the Curve and Classification

Consider the problem of assessing the quality of predictions for binary observations. Let $C(o)$ denote the class of an observation o. The *predicted* quantity might be a continuous quantity, e.g., a probability ranging from 0 to 1. This continuous quantity can be translated into a binary prediction by setting a threshold; if the predicted quantity is below the threshold, the result is a 0 prediction, otherwise, 1 is predicted. Depending on the threshold, there is a trade-off between precision and recall; on the one hand, if the threshold is low, recall of the 1-class will be high, but precision will be low, but on the other hand, if the threshold is high, precision will be high, but recall will be low. In order to characterize the quality of a predictor without fixing the threshold, *area under the ROC curve (AUC)* or its soft version *soft-AUC* can be used [2].

AUC. The AUC of a predictor f is defined as

$$AUC(f) \;:=\; P(f(x) < f(y) | C(x) = 0, C(y) = 1) \;\; .$$

Given a set of negative examples $\mathcal{D}^0$, and a set of positive examples $\mathcal{D}^1$, the following *Wilcoxon-Man-Whitney* statistic [6], which we denote $auc(f, \mathcal{D}^0 \cup \mathcal{D}^1)$, is an unbiased estimator of $AUC(f)$:

$$auc(f, \mathcal{D}^0 \cup \mathcal{D}^1) \;:=\; \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \mathbf{1}[f(t_0) < f(t_1)]}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|} \;\; ,$$

where $\mathbf{1}[f(t_0) < f(t_1)]$ denotes the *indicator function* of $f(t_0) < f(t_1)$; that is, $\mathbf{1}[f(t_0) < f(t_1)]$ is 1 if $f(t_0) < f(t_1)$ is true, and otherwise it is 0.

Given a dataset $\mathcal{D} = \mathcal{D}^0 \cup \mathcal{D}^1$, the exact value of $auc(f, \mathcal{D})$ can be computed in time $\mathcal{O}(|\mathcal{D}| \log(|\mathcal{D}|))$ by sorting the tuples t in the database with respect to the value of $f(t)$ in ascending order, after which we scan the data and maintain a count of 0-examples which have the value of f less than the current tuple.

Soft-AUC. For some classification algorithms, such as gradient descent, however, it is problematic that the statistic $auc(f, \mathcal{D})$ is not continuous in f. Furthermore, another disadvantage of the AUC measure is that no weights are assigned to the difference in scores; $auc(f, \mathcal{D})$ fully takes into account a pair of a higher scoring positive example with a lower scoring negative example, even if the margin is small. Both problems: the non-differentiability and the insensitivity to the difference in scores between positive and negative examples, are solved by the introduction of the following *soft-AUC* statistic (parameterized by β):

$$s\_auc_\beta(f, \mathcal{D}^0 \cup \mathcal{D}^1) \ := \ \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} sigmoid_\beta(f(t_1) - f(t_0))}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|} \ ,$$

where $sigmoid_\beta(x)$ is the function $\frac{1}{1+e^{-\beta x}}$. This function approximates the step function, but smoothes out the region around 0. For $\beta \to \infty$, $sigmoid_\beta$ pointwise converges to the step function. The computational cost of the soft-AUC, however, is quadratic in the number of tuples. Similar measures have been introduced in the literature to deal with the non-differentiability issue. We believe, however, that soft-AUC has its own merits, and thus propose we it as a measure in its own right.

Optimizing the AUC Directly. Recently, many new classification algorithms have been proposed that directly optimize the AUC measure [7, 3, 1, 5, 10, 9]. In these approaches, the AUC has to be computed repeatedly, as the classifier f is being changed during the training process. Because for large datasets $\mathcal{D}$, the cost of $\mathcal{O}(|\mathcal{D}| \log(|\mathcal{D}|))$ for every computation of the AUC-measure can be too high, it is often measured on only a small sample of the dataset.

In this paper, we propose another approach for optimizing the AUC directly. We propose the use of polynomial approximations of AUC and soft-AUC. These approximations have the advantage that they are more accurate than sampling, they can be computed in linear time, and it is possible to cache a concise summary of the dataset that allows, for small changes in the classification function, to compute the AUC without having to re-scan the dataset.

3 Polynomial Approximation of AUC and soft-AUC

The key observation is that the indicator function $\mathbf{1}[f(t_0) < f(t_1)]$ (resp. sigmoid) can be approximated by a polynomial. We only give the derivation for the AUC, because the soft-AUC case is similar.

To approximate the indicator function, we actually approximate the function $H(x) = \mathbf{1}[x > 0]$, which is the well-known *Heaviside step-function*. The required indicator function is then $H(f(t_1) - f(t_0))$. In Figure 1, a polynomial (Chebyshev) approximation of $H(x)$ has been plotted.

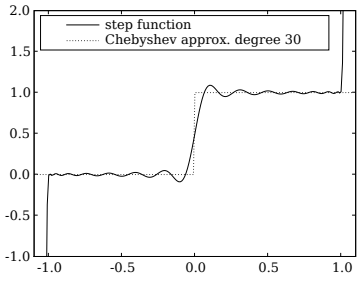

Fig. 1. Polynomial approximation of the Heaviside function

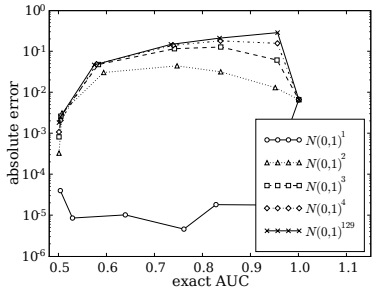

Fig. 2. Accuracy of the polynomial approximations.

Let now $\sum_{k=0}^{d} c_k x^k$ be a polynomial approximation of $H(x)$ of degree d. Then,

$$H(f(t_1) - f(t_0)) \approx \sum_{k=0}^{d} c_k (f(t_1) - f(t_0))^k = \sum_{k=0}^{d} c_k \sum_{l=0}^{k} \binom{k}{l} f(t_1)^l (-f(t_0))^{k-l}$$
$$= \sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} f(t_1)^l f(t_0)^{k-l}$$

where α_{kl} equals $c_k \binom{k}{l} (-1)^{k-l}$. This approximation of $H(x)$ leads directly to the following approximation for the *auc*. Let n_0 denote $|\mathcal{D}^0|$ and n_1 denote $|\mathcal{D}^1|$.

$$n_0 n_1 \, auc(f, \mathcal{D}) \approx \sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} f(t_1)^l f(t_0)^{k-l}$$
$$= \sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} \left(\sum_{t_1 \in \mathcal{D}^1} f(t_1)^l \right) \left(\sum_{t_0 \in \mathcal{D}^0} f(t_0)^{k-l} \right) \quad (1)$$

Notice that in (1), the quantities $\sum_{t_1 \in \mathcal{D}^1} f(t_1)^l$ and $\sum_{t_0 \in \mathcal{D}^0} f(t_0)^{k-l}$ for $1 \leq l \leq k \leq d$ can all be computed in one scan, and then combined afterwards. Following a similar convention as [8], we introduce the notation $s(f, \mathcal{D}) := \sum_{t \in \mathcal{D}} f(t)$. Following this convention, the approximation becomes:

$$auc(f, \mathcal{D}) \approx \frac{\sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} s(f^l, \mathcal{D}^1) s(f^{k-l}, \mathcal{D}^0)}{n_0 n_1} \quad (2)$$

Hence, we get an approximation of the *auc* in one linear scan over the database.

4 Training a Linear Classifier with the Approximation

In this section we show how the polynomial approximation of the *auc* can be plugged into a gradient descent method for linear discriminative analysis that optimizes the area under the ROC curve. Notice that the approximation can be plugged into other classification inducers as well, in a very similar way.

Suppose a dataset $\mathcal{D} = \mathcal{D}^0 \cup \mathcal{D}^1$ with m numerical attributes has been given. We will represent the elements of $\mathcal{D}$ as vectors $\mathbf{x} = [x_1 \dots x_m]$. The goal is now to find a vector of weights $\mathbf{w}$, such that the function $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ maximizes AUC; that is, $auc(f_{\mathbf{w}}, \mathcal{D}^0 \cup \mathcal{D}^1)$ is maximal w.r.t. $\mathbf{w}$. To find such an optimal vector of weights $\mathbf{w}$, we use a gradient descent method. The gain of using the polynomial approximation for the AUC will be three-fold: first, a costly sorting operation in the computation of the AUC is avoided, second, based on the approximation we can estimate the gradient, and third, we do not have to re-scan the dataset every time the weights are adjusted by storing a small summary of the dataset.

Before presenting the complete algorithm, we explain its components.

Approximating the Gradient. To apply a gradient descent method, we need to compute the gradient. The AUC of $f_{\mathbf{w}}$ for a fixed set of examples $\mathcal{D}$, however, is not continuous in the weights $\mathbf{w}$. We assume that there is an underlying infinite distribution of which $\mathcal{D}$ is only a sample, and the gradient of the AUC is approximated by applying the derivative of the polynomial approximation of the AUC on the sample. Another way to interpret this approach is that we actually optimize the accurate polynomial approximation, instead of the AUC.

The gradient of the AUC w.r.t. the weights $\mathbf{w}$ is $\left[\frac{\partial auc(f)}{\partial w_1}, \dots, \frac{\partial auc(f)}{\partial w_m}\right]$ and $\partial auc(f)/\partial w_i$ can be approximated by taking the partial derivatives of the polynomial approximation in Equation 2:

$$n_0 n_1 \frac{\partial auc(f)}{\partial w_i} \approx \sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} \left(\frac{\partial s(f^l, \mathcal{D}^1)}{\partial w_i} s(f^{k-l}, \mathcal{D}^0) + s(f^l, \mathcal{D}^1) \frac{\partial s(f^{k-l}, \mathcal{D}^0)}{\partial w_i} \right) \tag{3}$$

In the case of a linear classifier, $f(x) = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{m} w_i x_i$, we get

$$\frac{\partial s(f^l, \mathcal{D}^1)}{\partial w_i} = \sum_{x \in \mathcal{D}^1} \frac{\partial f(x)^l}{\partial w_i} = \sum_{x \in \mathcal{D}^1} l f(x)^{l-1} x_i = l \cdot s(x_i f^{l-1}, \mathcal{D}^1) \ . \tag{4}$$

Combining (3) and (4), we get the following approximation for the derivative:

$$n_0 n_1 \frac{\partial auc(f)}{\partial w_i} \approx \sum_{k=0}^{d} \sum_{l=0}^{k} \alpha_{kl} \left(l \cdot s(x_i f^{l-1}, \mathcal{D}^1) \cdot s(f^{k-l}, \mathcal{D}^0) + (k-l) \cdot s(f^l, \mathcal{D}^1) \cdot s(x_i f^{k-l-1}, \mathcal{D}^0) \right) \ . \tag{5}$$

Optimizing along the Gradient. We now show how to choose an optimal value of the learning rate; i.e., optimize the weights along the gradient direction.

Update Rule. Suppose that the current weights are $\mathbf{w}$, and we have approximated the gradient $\mathbf{g}$. When updating the weights $\mathbf{w} \leftarrow \mathbf{w} + \gamma\mathbf{g}$, the optimal value of the learning rate γ needs to be determined. In our case, instead of minimizing along the gradient, it is better to find the optimal angle α between old and new weights in the plane spanned by the current weight vector and the gradient. The reason for this is that the AUC does not depend on the length of $\mathbf{w}$, only on its direction. Hence, the weight vectors under consideration when selecting the optimal α are given by $\cos(\alpha)\mathbf{w} + \sin(\alpha)\mathbf{g}$ with α between 0 and 2π. We show how we can avoid scanning the database to get an updated value for the AUC every time we change α in order to find the optimal value.

Avoiding re-scanning. We can approximate $n_0 n_1 auc(f_{\cos(\alpha)\mathbf{w}+\sin(\alpha)\mathbf{g}}, \mathcal{D})$ as follows (the scaling factor $\frac{1}{\sqrt{2}}$ is explained in the next paragraph)

$$\begin{aligned}
&\sum_{k=0}^{d}\sum_{l=0}^{k} \alpha_{kl} \left(\sum_{x\in\mathcal{D}^1} \left(\tfrac{\cos(\alpha)}{\sqrt{2}}\mathbf{w}\cdot\mathbf{x} + \tfrac{\sin(\alpha)}{\sqrt{2}}\mathbf{g}\cdot\mathbf{x}\right)^l\right) \cdot \\
&\qquad \left(\sum_{x\in\mathcal{D}^0} \left(\tfrac{\cos(\alpha)}{\sqrt{2}}\mathbf{w}\cdot\mathbf{x} + \tfrac{\sin(\alpha)}{\sqrt{2}}\mathbf{g}\cdot\mathbf{x}\right)^{k-l}\right) \\
&= \sum_{k=0}^{d}\sum_{l=0}^{k} \alpha_{kl} \left(\textstyle\sum_{m=0}^{l} \beta_{l,m}\, s(f_{\mathbf{w}}^m f_{\mathbf{g}}^{l-m}, \mathcal{D}^1)\right) \cdot \left(\textstyle\sum_{m=0}^{k-l} \beta_{k-l,m}\, s(f_{\mathbf{w}}^m f_{\mathbf{g}}^{k-l-m}, \mathcal{D}^0)\right)
\end{aligned}$$

where $\beta_{l,m}$ denotes $\binom{l}{m} 2^{-\frac{l}{2}} \cos(\alpha)^m \sin(\alpha)^{l-m}$. Thus, after we have computed the gradient $\mathbf{g}$, one scan over the database is needed to compute $s(f_{\mathbf{w}}^m f_{\mathbf{g}}^{l-m}, \mathcal{D}^1)$ and $s(f_{\mathbf{w}}^m f_{\mathbf{g}}^{l-m}, \mathcal{D}^0)$, for all $1 \le l \le m \le d$. Based on this summary, the AUC of $f_{\cos(\alpha)\mathbf{w}+\sin(\alpha)\mathbf{g}}$ for all α can be computed without re-scanning the database.

Scaling of the Weights. One important problem we have to deal with in this application, is that the approximation of the Heaviside function is only accurate within the interval $[-1, 1]$. Outside of this interval, the approximation quickly deteriorates, as can be seen in Figure 1. Therefore, we have to make sure that for all points $t_1 \in \mathcal{D}^1$ and $t_0 \in \mathcal{D}^0$, the difference $(f(t_1) - f(t_0))$ is in the interval $[-1, 1]$. We show how this requirement can be met by re-scaling the weights vector $\mathbf{w}$. Obviously, re-scaling the weight vector only changes the magnitude of the scores of the classification function; the classifier and its AUC remain the same.

We need to re-scale the weights $\mathbf{w}$ in such a way that for all $\mathbf{x_0} \in \mathcal{D}^0$ and $\mathbf{x_1} \in \mathcal{D}^1$, the difference $f(x_1) - f(x_0) = \mathbf{w}\cdot\mathbf{x_1} - \mathbf{w}\cdot\mathbf{x_0}$ falls into the interval $[-1, 1]$. A straightforward solution is as follows. Let m^1, M^1, m^0, M^0 be the following numbers:

$$\begin{aligned}
m^0 &= \min_{x_0\in\mathcal{D}^0} \mathbf{w}\cdot\mathbf{x_0}, \quad M^0 = \max_{x_0\in\mathcal{D}^0} \mathbf{w}\cdot\mathbf{x_0}, \\
m^1 &= \min_{x_1\in\mathcal{D}^1} \mathbf{w}\cdot\mathbf{x_1}, \quad M^1 = \max_{x_1\in\mathcal{D}^1} \mathbf{w}\cdot\mathbf{x_1}.
\end{aligned}$$

From these numbers it can be derived that $f(x_1) - f(x_0)$ always falls into the interval $[m^1 - M^0, M^1 - m^0]$. Based on this interval, $\mathbf{w}$ can be re-scaled appro-

priately. In our implementation we have opted to re-scale $\mathbf{w}$ by dividing it by $\max(M^0 - m^1, M^1 - m^0)$.

For the optimization along the gradient, we have to guarantee correct scaling for every α. Let $\mathbf{w}'$ denote $\frac{1}{\sqrt{2}}(\cos(\alpha)\mathbf{w} + \sin(\alpha)\mathbf{g})$. Observe now, for all $\mathbf{x}_0 \in \mathcal{D}^0, \mathbf{x}_1 \in \mathcal{D}^1$:

$$\begin{aligned} |f_{\mathbf{w}'}(\mathbf{x}_1) - f_{\mathbf{w}'}(\mathbf{x}_0)| &= \frac{1}{\sqrt{2}}|(\cos(\alpha)(\mathbf{w}\mathbf{x_1} - \mathbf{w}\mathbf{x_0}) + \sin(\alpha)(\mathbf{g}\mathbf{x_1} - \mathbf{g}\mathbf{x_0}))| \\ &\leq \frac{1}{\sqrt{2}}(\cos(\alpha) + \sin(\alpha)) \ \leq \ \frac{1}{\sqrt{2}}(\sqrt{2}) \ = \ 1 \end{aligned}$$

Notice that in the derivation we implicitly assume that $\mathbf{w}$ and $\mathbf{g}$ are appropriately scaled. Hence, when using update rule $\mathbf{w} \leftarrow \frac{1}{\sqrt{2}}(\cos(\alpha)\mathbf{w} + \sin(\alpha)\mathbf{g})$, we are guaranteed that the weights are scaled correctly.

Complete Algorithm. The complete algorithm is given in Algorithm 2. The number of iterations is fixed to *maxiter*. In every iteration, first the gradient is computed (lines 2 to 5). To this end, the database is scanned once to collect the necessary supports (lines 3 and 4). These supports are then combined to form the gradient (line 5). Once the gradient is found, the optimal angle α is computed. Again, first the necessary supports are counted in one scan over the database (lines 7 and 8). These supports suffice to find the optimal α without re-scanning the database. In the implementation, finding the optimal α is done by ranging over many different values of α evenly spread over $[0, 2\pi]$, and selecting the one that gives the highest AUC. The AUC scores for the different values of α can be computed without re-scanning the database. This optimization of α is performed by Algorithm 1 (line 4). The method is quite crude, but any other linear optimizer could be used instead. Once the optimal α has been found, the weights are updated (line 11), and the next iteration is entered.

It seems that our weight rescaling method requires an extra database scan. In our implementation, however, we combine it with support counting. Rescaling is done (if needed) continuously as records are read (this happens in lines 3,4 and 7,8). Thus, only two database scans per gradient descent iteration are required.

Soft-AUC. As we discussed earlier, the AUC does not take into account how close the points are to the decision boundary. Whether a pair of points $(\mathbf{x}_0, \mathbf{x}_1)$ contributes to the AUC solely depends on $f(\mathbf{x}_1)$ being larger than $f(\mathbf{x}_0)$, not on the magnitude of this difference. It would be more natural if small differences were counted less than large differences, like it is also the case in, e.g., mean squared error. This observation is the main motivation for the soft-AUC measure:

$$s\_auc_\beta(f, \mathcal{D}^0 \cup \mathcal{D}^1) \ := \ \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} sigmoid_\beta(f(t_1) - f(t_0))}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|} \ .$$

For optimizing soft-AUC, our method works perfectly well; having a good polynomial approximation is even easier, as the main difficulty, the steep step

Algorithm 1 Optimize α

Input: $s(f^m_{\mathbf{w}} f^{l-m}_{\mathbf{g}}, \mathcal{D}^1)$, and $s(f^m_{\mathbf{w}} f^{l-m}_{\mathbf{g}}, \mathcal{D}^0)$ for all $1 \leq l \leq m \leq d$
Output: Optimal angle *ang*

1: $opt := 0$; $ang := 0$;
2: **for all** $\alpha := 0 \ldots 2\pi$ step .01 **do**
3: Approx. AUC of $f_{\frac{1}{\sqrt{2}}(\cos(\alpha)\mathbf{w}+\sin\alpha\mathbf{g})}$, using Equation (6).
4: **if** AUC $>$ *opt* **then**
5: *opt* :=AUC;
6: $ang := \alpha$;
7: **return** *ang*

Algorithm 2 Learning a linear classifier

Input: Database $\mathcal{D} = \mathcal{D}^0 \cup \mathcal{D}^1$ with m attributes, initial weights $\mathbf{w}$, maximal number of iterations *maxiter*.
Output: Weights $\mathbf{w}$, reached via a gradient descent method

1: **for** $iter := 1 \ldots maxiter$ **do**
2: {*Approximate gradient*}
3: Count $s(x_i f^{l-1}, \mathcal{D}^1)$, $s(f^l, \mathcal{D}^1)$ for $l = 1 \ldots d$, $i = 1 \ldots m$ in one scan over $\mathcal{D}^1$.
4: Count $s(x_i f^{l-1}, \mathcal{D}^0)$, $s(f^l, \mathcal{D}^0)$ for $l = 1 \ldots d$, $i = 1 \ldots m$ in one scan over $\mathcal{D}^0$.
5: Approx. gradient $\mathbf{g}$ based on the supports counted in steps 1 and 2, using Equation (5).
6: {*Approximate AUC of* $f_{\frac{1}{\sqrt{2}}(\cos(\alpha)\mathbf{w}+\sin(\alpha)\mathbf{g})}$}
7: Count $s(f^m_{\mathbf{w}} f^{l-m}_{\mathbf{g}}, \mathcal{D}^1)$, for all $1 \leq l \leq m \leq d$ in one scan over $\mathcal{D}^1$.
8: Count $s(f^m_{\mathbf{w}} f^{l-m}_{\mathbf{g}}, \mathcal{D}^0)$, for all $1 \leq l \leq m \leq d$ in one scan over $\mathcal{D}^0$.
9: {*Update weights* $\mathbf{w}$}
10: Find optimal α, with Algorithm 1
11: $\mathbf{w} := \frac{1}{\sqrt{2}}\cos(\alpha)\mathbf{w} + \sin(\alpha)\mathbf{g}$
12: **return** $\mathbf{w}$

in the Heaviside, is avoided. There are, however, still some problems we have to take into account. First of all, re-scaling the weights no longer leaves the objective function unchanged. Therefore, the optimization problem actually becomes: find optimal weights $\mathbf{w}$, with $\|\mathbf{w}\| = 1$, such that $s\_auc_\beta(f_w, \mathcal{D})$ is maximal. This requirement contradicts the scaling needed to keep the approximation accurate. Therefore, the re-scaling is kept, but, every time we need the approximations, the polynomial coefficients are recomputed, such that not $s\_auc_\beta(f_w, \mathcal{D})$ is approximated, but $s\_auc_{\beta/\|\mathbf{w}\|}(f_w, \mathcal{D})$. Put otherwise, instead of requiring that $\|\mathbf{w}\| = 1$, and $s\_auc_\beta(f_w, \mathcal{D})$ is optimal, we equivalently require that $s\_auc_{\beta/\|\mathbf{w}\|}(f_w, \mathcal{D})$ is optimal. We do not go into detail here due to lack of space.

5 Experimental evaluation

We implemented the linear approximation of AUC and soft-AUC, and a linear classifier inducer based on these approximations. For both the approximation in

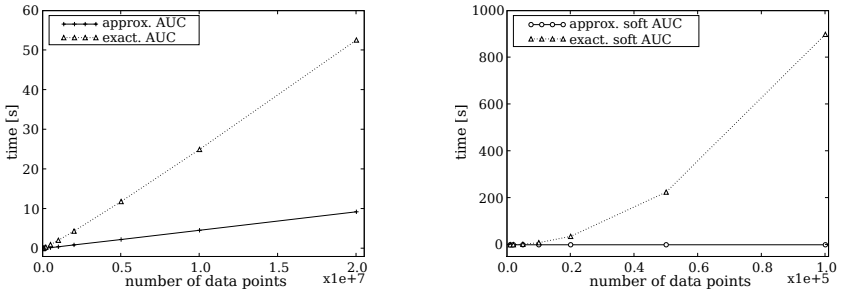

Fig. 3. Performance of the polynomial approximations.

isolation and the classifier inducer we test both the accuracy and the running times. For the polynomial approximations used in the experiments, a degree of 30 was chosen as a compromise. Higher values for the degree did not give a significant increase in accuracy, while decreasing performance. Numerical stability problems do become visible for high degrees, but for the degree of 30 no such problems occurred on any of the datasets used. It should be noted, however, that the optimal degree highly depends on the numerical precision of the computations and even on the architecture of the computer used.

Datasets. The characteristics of the datasets used for testing are given below. In case of the forest cover dataset only the two most frequent classes were kept. We tried two versions of the forest cover data, one with only 10 numerical attributes kept, and another with all attributes. This allowed us to see how binary attributes influence accuracy.

dataset	records	attrs
sonar	208	60
KDD Cup 04 physics	50000	78
forest cover 10 numeric attrs	495141	10
forest cover all attrs	495141	54
KDD Cup 98 all attrs	95412	464

All experiments in this section have been performed with 10 fold cross-validation.

Performance of the Polynomial Approximation. To test the accuracy of the polynomial approximation, we used synthetically generated data. The data was generated by randomly drawing positive examples with f-values with mean m_1 and standard deviation 1 and negative examples with mean m_2 and standard deviation 1 following a normal distribution, and raising this number to the power p. By varying the difference $m_1 - m_2$, different AUC values are obtained. The higher the value of p becomes, the smaller the average distances between the scores become, making the approximation difficult since many values will fall in the poorly approximated region of the Heaviside function. As can be seen

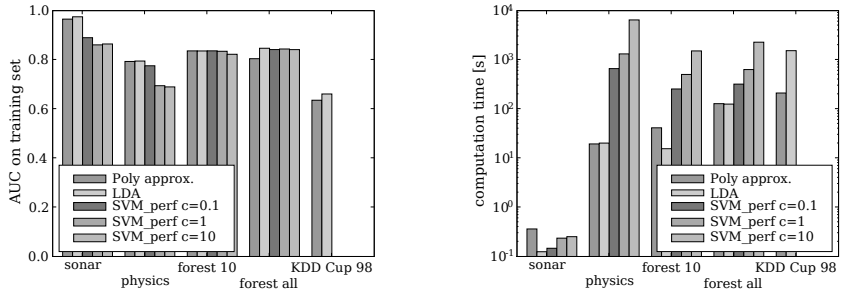

Fig. 4. AUC computation time for linear models built using polynomial approximation, Linear Discriminant Analysis and SVM_perf. Data on KDD Cup 98 is missing for SVM_perf due to excessive computation time.

in Figure 2, the accuracy of the approximation is very high in general, but deteriorates slightly when there are only small differences between the scores (high powers p). In the graphs in Figure 3 the running times for the approximations of the AUC and soft-AUC are given, showing significant performance gains.

Performance of the Linear Classifier. We begin by examining the performance and accuracy of training an AUC-maximizing linear classifier based on polynomial approximations. We used the maximum of 30 iterations, and the polynomial degree was set to 30. Figure 4 shows the results of comparing our approach with Linear Discriminant Analysis and SVM_perf, a version of Support Vector Machine minimizing AUC directly [9]. Since the SVM's performance depends on a parameter c we used three different values of this parameter.

Our approach achieves better values of AUC than the SVM and is often more than an order of magnitude faster. We were, e.g., not able to run the SVM on the KDD Cup'98 dataset. This is probably due to large number of attributes in this dataset. The main step of our method, the linear search, is totally independent of the number of attributes. Our approach minimized AUC directly without any performance problems. Forest cover gives worse results when all attributes are present. This is due to binary attributes which cause the occurrence of values of f very close to each other, thus causing significantly worse approximation.

In order to check the usefulness of direct AUC minimization we also compared it with Linear Discriminant Analysis, a standard linear classification technique. Due to the efficiency of our approach, we were able to perform direct AUC optimization on large datasets and thus obtain meaningful comparison. As it turns out, minimizing AUC directly does not give any visible improvement over classifiers built using LDA. This seems to confirm results presented in [4].

In [7] a method of fast AUC computation based on sampling is presented. We modified our algorithm to compute AUC directly on a small sample at each minimization step to obtain a similar approach. Figure 5 shows the results. It

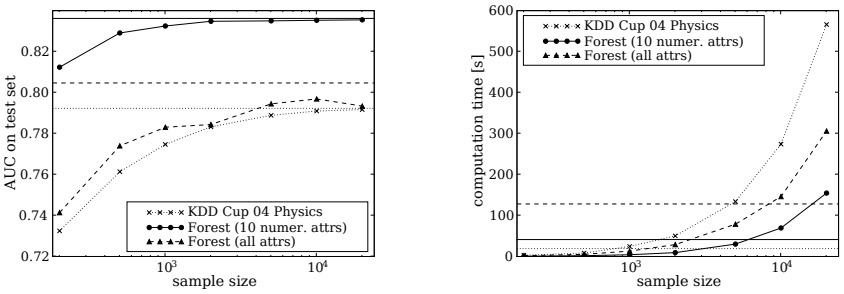

Fig. 5. Test set AUC and computation time for linear models built using exact AUC computation on samples. Horizontal lines denote test set AUC and computation time for respective models built using polynomial approximation.

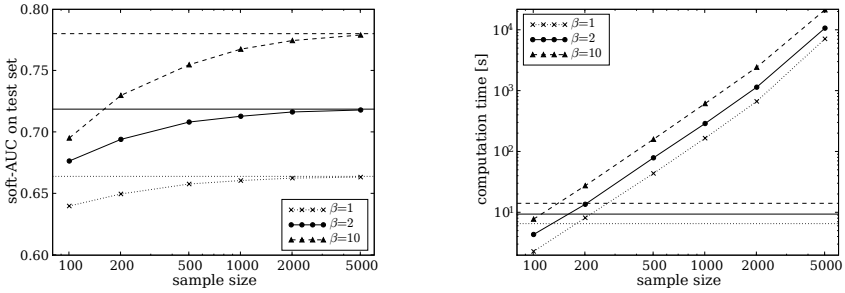

Fig. 6. Sampling vs. polynomial approximation for minimizing soft-AUC on KDD Cup 04 Physics datasets. Horizontal lines denote test set soft-AUC and computation time for respective models built using polynomial approximation.

can be seen that polynomial approximations achieve higher accuracy in shorter time.

We now present some experiments on minimizing soft-AUC. We compared the method with a sampling based version. Figure 6 has the results. Again, it can be seen that our polynomial approximation gives better results than sampling. The experiment was extremely time consuming, since computing the exact soft-AUC for the final classifier took hours (quadratic time in number of records). At the same time, *building* the classifier using our approach took just seconds.

Summary of Experimental Results. For the linear approximation, we tested the accuracy and the performance in comparison with the exact versions. The presented experiments support our claim that the approximation is very accurate and that there is a large performance gain in running time. For the linear classifier inducer, we compared both the performance w.r.t. running time and

w.r.t. predictive power of the learned model, in comparison with sampling, Linear Discriminative Analysis (a linear model learner optimizing accuracy), and SVM_perf [9] (a version of Support Vector Machine learner, minimizing the AUC directly). The experiments show that the running times of our method are comparable to LDA, which is significantly lower than the time required by SVM_perf. On the other hand, sampling is not efficient as it requires too many examples to reach the same accuracy as our approximation. Hence, both in running time and predictive performance, our method is always comparable to the winner, hence combining the advantages of the different methods.

6 Summary and Conclusion

A polynomial approximation of the Area Under the ROC Curve, computable in linear time, has been presented, and was applied to inducing a classifier that optimizes AUC directly. We also proposed a soft-AUC measure which does not give simple 0/1 scores to points close to the decision border.

Experimental evaluation has shown that the method is efficient and accurate compared to other methods for approximating the AUC. As a proof of concept, the method was plugged into the training of a linear classifier by optimizing the AUC or soft-AUC directly. With the approximation technique, similar AUC scores were reached as with existing techniques. The scalability and running times of the proposed approximation technique, however, are vastly superior.

Future work will include extending the approach to nonlinear classifiers.

References

1. K. Ataman, W. N. Street, and Y. Zhang. Learning to rank by maximizing auc with linear programming. In *IEEE International Joint Conference on Neural Networks (IJCNN 2006)*, pages 123–129, 2006.
2. A.P. Bradley. Use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
3. U. Brefeld and T. Scheffer. AUC Maximizing Support Vector Learning. In *Proc. ICML workshop on ROC Analysis in Machine Learning*, 2005.
4. C. Cortes and M. Mohri. Auc optimization vs. error rate minimization. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.
5. C. Ferri, P. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the ROC curve. In *ICML*, pages 139–146, 2002.
6. J.A. Hanley and B.J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.
7. A. Herschtal and B. Raskutti. Optimising area under the roc curve using gradient descent. In *ICML*, pages 49–56. ACM Press, 2004.
8. S. Jaroszewicz. Polynomial association rules with applications to logistic regression. In *KDD*, 2006.
9. T. Joachims. A support vector method for multivariate performance measures. In *ICML*, 2005.
10. A. Rakotomamonjy. Optimizing Area Under Roc Curve with SVMs. In *Workshop on ROC Analysis in Artificial Intelligence*, 2004.