

Area Under ROC Optimisation using a Ramp Approximation

Alan Herschtal
Telstra Corporation

Bhavani Raskutti
Telstra Corporation

Peter K. Campbell
Telstra Corporation

Abstract

This paper introduces AURORA, an algorithm that builds binary classifiers to optimise the area under the ROC curve (the AUC). AURORA builds non-linear classifiers of the data by binarising the raw input features. Feature selection is performed using a fast heuristic routine, and gradient descent is used to optimise the coefficients of the selected features. Both feature selection and gradient descent are designed to be optimal for AUC. Non-differentiability of the objective function is overcome using a ramp-based approximation for the AUC. The use of this ramp-based approximation also allows the AUC to be calculated in near $O(n)$ time, where n is the number of labelled observations available. The entire AURORA algorithm then also has computational complexity near $O(n)$. AURORA is compared with several other classifiers, over eight binary classification tasks, and generally produces classifiers which are significantly more accurate. AURORA is also highly scalable with increasing number of training examples, and increasing number of input features. **Keywords:** ROC, AUC, Gradient Descent, Binary Classification, Mann-Whitney Statistic

1 Introduction

Consider a general binary classification task, in which an algorithm, which will be referred to as an *inducer*, is used to develop a function of the data, known as a *classifier*. The aim of the inducer is to induce a classifier which classifies the data observations into the two classes as accurately as possible. In many such binary classification tasks, the aim of the classification is to sort the observations into a list so that the minority class observations are concentrated as tightly as possible towards the top of the list. That way, if only a small subset of all observations can be acted upon due to limited resources, and the size of that subset is not known a priori, the sorting will enable as high a percentage as possible of the observations of interest (the minority class observations) to be included in this subset. In other words, for any given decision threshold value, it is desirable to have as many as possible of the minority class observations above the threshold (high true positive rate) together with as few as possible of the majority class observations (low false positive

rate). Graphing the true positive rate against the false positive rate as the decision threshold is varied yields the Receiver Operating Characteristic, or ROC curve. The area under the ROC curve, or the AUC, is a decision threshold independent measure of classifier goodness, and has often been used as such [2, 15].

Most binary classifier inducers, however, have as their objective function some other metric, such as mean square error, one-sided linear penalty or one-sided square penalty. If the real objective is to optimise the sorting order, as measured by the AUC, such inducers are actually solving the wrong problem. Hence they are likely to perform sub-optimally when the performance metric is the AUC. This has been found to be the case empirically on a wide variety of datasets [10]. Conversely, if the inducer's objective function is closely related to the AUC, then it yields classifiers with better AUC [3, 5, 16].

In this paper, we introduce AURORA, an algorithm that optimises the AUC directly. AURORA searches for a non-linear classifier of the data that is optimal for the AUC, using gradient descent to optimise the classifier's coefficients. It is compared with a number of other non-linear and linear inducers for binary classification, namely: RankBoost [3, 4]; C5.0 [12, 13]; an SVM with one-sided square penalty (SVM-L2) [6, 11, 14]; logistic regression; and RankOpt [5].

Sec. 2 describes AURORA's objective function. Sec. 3 discusses details of AURORA's algorithm, including the feature selection and gradient descent steps. Sec. 4 describes the experimental procedure, the datasets that AURORA was tested on, and the results. Conclusions and future work are discussed in Sec. 5.

2 Objective Function

Consider a rectangular dataset containing n i.i.d. observations, drawn from a population. P of the n observations belong to the minority, or positive, class, and Q to the majority, or negative, class. The positive observations are denoted by vectors \vec{x}_j^+ , $j = 1 \dots P$, and the negative ones by \vec{x}_k^- , $k = 1 \dots Q$, where each element of vector \vec{x}_j^+ or \vec{x}_k^- represents the value of a raw predictor, or feature, for the corresponding ob-

servation. The dataset has m predictor variables, so $\vec{x}_j^+ = (x_{1,j}^+, x_{2,j}^+, \dots, x_{m,j}^+)$, and $x_{i,j}^+$ is the j^{th} instance of random variable X_i^+ . Furthermore, \vec{x}_j^+ is the j^{th} instance of the vector random variable \vec{X}^+ . Equivalent definitions hold for the majority class.

Consider an observation pair, consisting of exactly one observation drawn at random from each class $\{\vec{x}_j^+, \vec{x}_k^-\}$. The AUC of a classifier on a given dataset can be expressed as the probability that for such an observation pair, the score of the minority class observation is greater than that of the majority class observation [1]. That is, for a classifier of the data $f(\cdot)$, ignoring ties, the AUC is given by

$$AUC(f) = \Pr(f(\vec{x}_j^+) > f(\vec{x}_k^-)).$$

This is simply the Mann-Whitney statistic [8] scaled by $\frac{1}{PQ}$ [16]. If we take the heaviside function, $g(x)$, defined as

$$g(x) = \begin{cases} 0, & x < 0, \\ 0.5, & x = 0, \\ 1, & x > 0 \end{cases}$$

then the minimum variance unbiased estimator of the true value of $AUC(f)$, based on our finite population dataset, is given by

$$(2.1) \quad \widehat{AUC}(f) = \frac{1}{PQ} \sum_{j=1}^P \sum_{k=1}^Q g(f(\vec{x}_j^+) - f(\vec{x}_k^-))$$

2.1 Classifier Candidates. It is desirable not to limit $f(\vec{X})$ to be a linear function of the data, as this will limit the potential accuracy of the classifier for any problems in which the linear classifier is a poor fit of the data. However in the non-linear function space, it is necessary to restrict $f(\vec{X})$ to some family of functions, the parameters of which may then be determined from the training data. This section describes that family.

Firstly, the m raw predictors, X_i , $i = 1 \dots m$ are binarised, giving rise to binary derived predictors Z_h . The binarisation method differs between ordinal and cardinal variables. Amongst ordinal valued predictors, we distinguish between those that are discrete and those that are continuous, as follows.

For a discrete valued raw predictor X_i , the cardinality of X_i , denoted by C_i , is measured by counting the number of unique values, or levels, observed for X_i over the training set. If $C_i \leq 22$, then $(C_i - 1)$ binary predictors are generated by placing a threshold between each consecutive pair of levels of X_i . For this purpose, consecutive levels with $< 0.1\%$ of the data are amalgamated.

For raw predictors which are either continuous, or discrete valued with $C_i > 22$, binarisation proceeds as follows. For each such raw predictor X_i , consider a range of thresholds t_{ip} . For each value of index p , we define a single binary-valued derived predictor, $Z_{ip} = (X_i \geq t_{ip})$. The values assigned to t_{ip} are determined by certain fixed percentile values of the corresponding raw predictor X_i , measured on the training data. These percentiles are the same for all predictors X_i , $i = 1 \dots m$. In all experimentation to be described here, every 5^{th} percentile point was used as a threshold, i.e. (5, 10, 15, 20, \dots , 95). This means that the values of t_{ip} for any i are simply the 5^{th} , 10^{th} , 15^{th} , etc. percentiles of X_i on the training set. Re-indexing the Z_{ip} s so that they are indexed by a scalar counter h yields a set of boolean-valued derived predictors denoted by Z_h .

This method of binarising the data will give rise to an even spread of thresholds for most raw predictor distributions. One circumstance under which it fails to do so, however, is when a continuous valued predictor contains a spike. This could happen, for example, when a predictor has missing values which have been imputed with some constant value, such as the mean. Under such circumstances the above algorithm may put many thresholds at the same value, namely the value at the spike.

To overcome this, we first “de-spike” the raw predictors. For a continuous valued predictor, a subset of the training data is selected such that the number of observations with any given value is no more than 1% of the total number of observations. Any observations in excess of this are not included in the subset, and it is this subset that is used for the purposes of calculating the percentile based thresholds.

Cardinal valued raw predictors may be binarised by converting a single C -level cardinal predictor into C individual one-vs-rest binary predictors.

Having generated binary predictors, they may be combined via the boolean operators *AND* and *OR*, to form binary predictors, $(Z_{h_1} \text{ AND } Z_{h_2} \dots)$ and $(Z_{h_1} \text{ OR } Z_{h_2} \dots)$. The number of base predictors *AND*-ed or *OR*-ed together is referred to as the “order” of the predictor. AURORA then considers only classifiers that are linear functions of either the first-order binary predictors Z_h , or the higher order binary predictors derived from them using *AND* and *OR*.

For notational efficiency, we will also denote the predictors which are higher than first-order by Z_h , with each binary predictor being given a unique value of h , regardless of its order.

Since $f(\cdot)$ is restricted to be linear in the predictors \vec{Z} , it may be written as $f(\vec{X}) = \sum_{h=1}^H (\beta_h \cdot Z_h) = \vec{\beta} \cdot \vec{Z}$, where the Z s are derived from the X s as shown above.

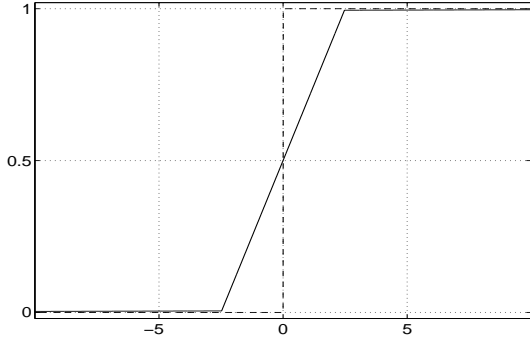


Figure 1: The ramp and heaviside functions

Therefore, if the thresholds t_{ip} are known and fixed, and the training data is known, the vector $\vec{\beta}$ is sufficient to define the classifier.

Note that for real world problems the number of derived predictors may be very large (in the thousands), but only a small number of them will ordinarily be used in the final classifier. In other words, for most values of h , β_h will equal zero in the final classifier.

2.2 The Ramp Statistic. Since the heaviside function upon which the AUC is based is undifferentiable, and we desire to use gradient descent, it is replaced by a ramp function, $r(x)$, shown in Figure 1.

We refer to the resulting approximation to the AUC as the “ramp statistic”, $R(\vec{\beta})$, defined as

$$(2.2) \quad R(\vec{\beta}) = \frac{1}{PQ} \sum_{j,k}^{P,Q} r(\vec{\beta}(z_j^+ - z_k^-)),$$

where z_j^+ and z_k^- are defined in an analogous way to \vec{x}_j^+ and \vec{x}_k^- . Note that as $\|\vec{\beta}\|$ increases, the ramp function approaches the heaviside function, and hence the ramp statistic becomes a good approximation to the true AUC.

Gradient descent techniques can then be applied to optimise the value of $R(\vec{\beta})$. The ramp function consists of 3 segments: a central segment which has a high gradient, and 2 side segments. Rather than being completely flat, the side segments are given a very small but non-zero gradient (set to be 1/1000th of the gradient of the central section of the ramp), to prevent a zero value being calculated for the gradient of $R(\vec{\beta})$ at any stage during gradient descent.

The value of the AUC statistic, which is the true objective function, depends on the direction of $\vec{\beta}$ only and not on its magnitude. The ramp statistic, on the other hand, depends on both the magnitude and the direction of $\vec{\beta}$. Hence we are not interested in optimising

$\|\vec{\beta}\|$, since it has no effect on the AUC which is the true objective function. So we should hold $\|\vec{\beta}\|$ constant, and optimise the direction of $\vec{\beta}$ only. Hence we may approximate a classifier which optimises the AUC by optimising $R(\vec{\beta})$, constrained to the hypersphere $\|\vec{\beta}\| = B$, in $\vec{\beta}$ -space, with B fixed and fairly large. Formally, we are after $\vec{\beta}_{OPT} = \operatorname{argmax} R(\vec{\beta})$, s.t. $\|\vec{\beta}\| = B$. A discussion on determining the value of B can be found in Sec. 3.3.

2.3 Computational Efficiency. Instead of being calculated as a double sum of heaviside functions, as shown in eqn. 2.1, the AUC can alternatively be calculated as a function of the sum of the ranks of the positive class observations. This means that the computational complexity of calculating $AUC(\vec{\beta})$ consists of 2 components: a sort, which is $O(n \log(n))$; and a summation of ranks, which is order $O(n)$. This means that the overall complexity of calculating $AUC(\vec{\beta})$ is $O(n \log(n))$.

It is of interest to know what claims can be made for the computational efficiency of the ramp based AUC approximation discussed in Sec. 2.2. Since the ramp function is piecewise linear, we may express its i^{th} piece as $a_i(x) + b_i$, $i = 1 \dots 3$, a_i being the slope of the i^{th} piece. For any given fixed value of $\vec{\beta}$, each observation pair falls into one of the 3 segments, so we may write $R(\vec{\beta}) = \sum_{i=1}^3 R_i(\vec{\beta})$, where $R_i(\vec{\beta})$ is the component of $R(\vec{\beta})$ due to all observation pairs $\{z_j^+, z_k^-\}$ that fall into the i^{th} piece of the ramp function.

We may write

$$(2.3) \quad R_i(\vec{\beta}) = \frac{1}{PQ} \sum_{j,k} r(\vec{\beta}(z_j^+ - z_k^-)),$$

with the values of j, k restricted as above. Using the property of piecewise linearity,

$$\begin{aligned} R_i(\vec{\beta}) &= \frac{1}{PQ} \left(\sum_{j,k} a_i(\vec{\beta}(z_j^+ - z_k^-)) + b_i \right) \\ &= \frac{1}{PQ} \left(\sum_{j,k} a_i(\vec{\beta}(z_j^+)) - \sum_{j,k} a_i(\vec{\beta}(z_k^-)) + \sum_{j,k} b_i \right) \\ &= \frac{1}{PQ} \left(\sum_j C_{ji}^+ a_i(\vec{\beta}(z_j^+)) \right. \\ &\quad \left. - \sum_k C_{ki}^- a_i(\vec{\beta}(z_k^-)) + C_{jk} b_i \right) \end{aligned}$$

C_{ji}^+ is the number of negative class observations which, when paired with the j^{th} positive class observation, fall into piece i of the ramp. Conversely, C_{ki}^- is the number of positive class observations which, when paired with the k^{th} negative class observation, fall into

piece i of the ramp. C_{jk} is the total number of observation pairs which fall into piece i of the ramp.

The expression for $R_i(\vec{\beta})$ above is clearly $O(n)$, and once the data observations have been sorted, the values of the C s can be found by a simple counting exercise, which is also $O(n)$.

Hence we can make identical computational efficiency claims for the ramp statistic as we can for the AUC statistic itself, namely that once the data observations have been sorted, which is an $O(n \log(n))$ operation, everything else is $O(n)$.

When the algorithmic details of AURORA are discussed in Sec. 3 it will become apparent that the requirement to sort the data arises relatively infrequently. This means that unless the amount of data is very large, the $O(n)$ components will dominate, leaving us with an overall algorithm that is very close to $O(n)$.

3 The AURORA Algorithm

AURORA’s classifier generation commences with a blank classifier (i.e. $\beta_h = 0, \forall h$.) It then selects the single best binary feature (the one that gives the best AUC when used on its own) using exhaustive search over all first-order candidate features obtained using binarisation, as defined in Sec. 2.1.

From this point on AURORA uses a heuristic to select the single next best candidate feature, and then performs gradient descent over the space defined by all features selected thus far. All other features, being as yet unselected, remain with a value of $\beta_h = 0$.

The process of successively selecting a feature and then performing gradient descent is repeated iteratively, with each added feature generating a classifier more complex than the previous. Learning stops when either the improvement in AUC on the training set between consecutive feature selections falls below a critical value, or an upper limit of the number of features that may be selected (i.e. the number of values of h for which β_h may be non-zero) is reached. This upper limit is fixed at 200 selections, which matches the number of boosting rounds for RankBoost (see Sec. 4.2).

The following subsections will discuss the feature selection heuristic and the gradient descent algorithm in detail. Pseudocode for AURORA’s overall algorithm is shown in Table 3.

3.1 Feature Selection. We seek a heuristic for selecting the k^{th} feature, after having selected $k - 1$ features.

Having already performed gradient descent over the feature space defined by the first $k - 1$ features, we now sit at the point in $(k - 1)$ -dimensional $\vec{\beta}$ -space for which $R(\vec{\beta})$ is maximised. We call this point $\vec{\beta}_{OPT,k-1}$.

Consider a candidate k^{th} feature, Z_{cand} . Ideally we would like to know the size of improvement in $R(\vec{\beta})$ that can be obtained by adding Z_{cand} as the k^{th} selected feature. In other words, for a particular candidate k^{th} feature Z_{cand} , what is the value of $R(\vec{\beta}_{OPT,k}) - R(\vec{\beta}_{OPT,k-1})$? The Z_{cand} for which this quantity is largest is the locally optimal choice for the k^{th} feature.

The exhaustive way of finding the locally optimal k^{th} feature is to consider each candidate in turn, and perform gradient descent in the resulting k -dimensional space, starting at $(\vec{\beta}_{OPT,k-1}, 0)$, (i.e. the initial value of β_k is set to zero). However we may have hundreds or even thousands of candidates, and gradient descent is computationally intensive. Hence we need a more efficient method.

To this end, we introduce two fast heuristic estimates of the size of improvement in $R(\vec{\beta})$ for a given new selected feature.

The first of these is the anti-polar gradient, or APG. The APG is the value of $|\frac{dR}{d\beta}|$ in k -dimensional space at the point $\vec{\beta} = (\vec{\beta}_{OPT,k-1}, 0)$. Figure 2 plots APG against improvement in test AUC for the 8 datasets used in experimentation. These datasets are fully described in Section 4.1. The left panel shows plots for the 5th feature selection, and the right panel for the 20th. As can be seen from these plots, the APG is sufficiently well correlated with the improvement in AUC, $R(\vec{\beta}_{OPT,k}) - R(\vec{\beta}_{OPT,k-1})$. Specifically, this figure shows that by taking the top few features by APG score as the candidate set, one will almost certainly include in that set one of the best features in terms of AUC improvement, i.e. a feature that is locally close to optimal. Hence the APG can be used to short-list the candidate features.

The second fast heuristic method of estimating the improvement in $R(\vec{\beta})$ is to measure $AUC(\vec{\beta}_k) - AUC(\vec{\beta}_{OPT,k-1})$ at $\vec{\beta}_k = (\vec{\beta}_{OPT,k-1} \cos \theta, B \sin \theta)$, for some value of θ . This is the AUC at an angle of θ along the contour line joining the $(k - 1)$ -dimensional solution $(\vec{\beta}_{OPT,k-1}, 0)$ to the pole where the new candidate feature is being considered on its own. We set θ to 30°, and refer to this measure as the contour midpoint. It has been found empirically that a locally optimal feature with a poor APG value, is highly likely to score well by the contour midpoint metric.

The feature selection algorithm then, proceeds as follows. All first-order candidate features are ranked by APG. Each feature that ranks in the top 20, is considered to be in the short-list, and is tested using a full gradient descent run. Further, the top 75 first-order candidate features by APG are ranked by contour midpoint, and the top 20 of these are also short-listed,

