

# 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  $x_{ij}^+$  is the  $j^{\text{th}}$  instance of random variable  $X_i^+$ . Furthermore,  $\vec{x}_j^+$  is the  $j^{\text{th}}$  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  $5^{\text{th}}$  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  $5^{\text{th}}$ ,  $10^{\text{th}}$ ,  $15^{\text{th}}$ , etc. percentiles of  $X_i$  on the training set. Re-indexing the  $Z_{ips}$  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.

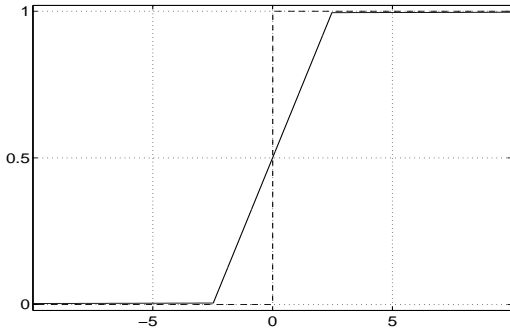


Figure 1: The ramp and heaviside functions

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  $h$ ,  $\beta_h$  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  $\vec{z}_j^+$  and  $\vec{z}_k^-$  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 1/1000'th 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  $\beta_h$  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  $\beta_k$  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  $|\frac{dR}{d\vec{\beta}}|$  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 5<sup>th</sup> feature selection, and the right panel for the 20<sup>th</sup>. 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  $\theta$ . This is the AUC at an angle of  $\theta$  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  $\theta$  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,

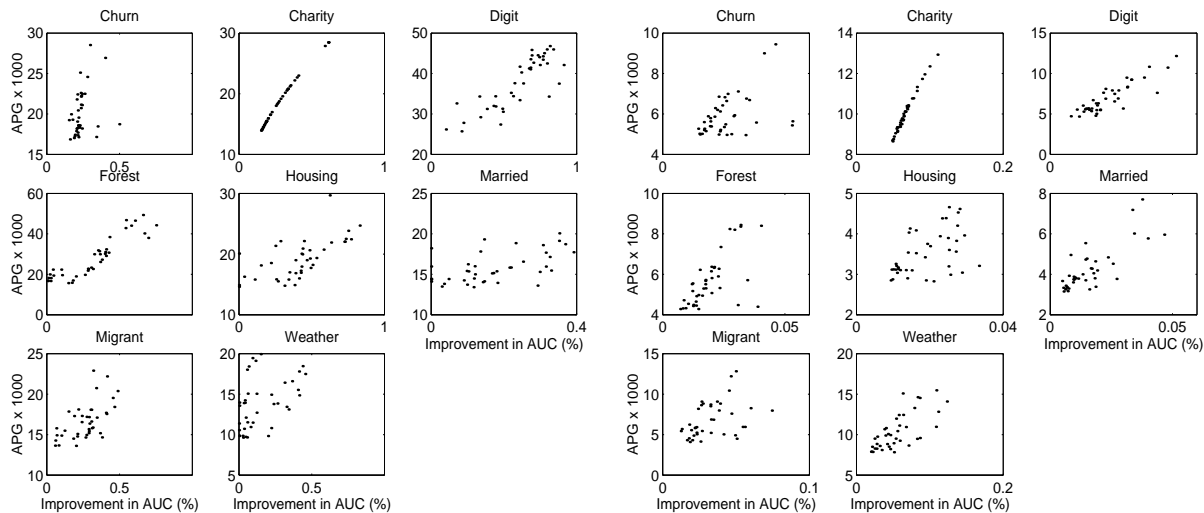


Figure 2: Scatterplot of APG score against AUC improvement, for 5<sup>th</sup> and 20<sup>th</sup> feature selections

and tested using a full gradient descent run. If the improvement in the ramp statistic on the training set of the best of these features is greater than a particular threshold,  $\Delta R_{min}$ , then it is selected as the  $k^{th}$  feature. Otherwise, second-order features are considered in a similar way. In order to limit the search space, only the best  $b_2$  first-order binary features (consisting of the best  $b_2/2$  ranked by APG and the best  $b_2/2$  ranked by midcontour point) are combined pairwise to generate the candidate second-order binary features. This results in  $2 \times \binom{b_2}{2}$  second-order predictors, one set of  $\binom{b_2}{2}$  for the AND-ed first-order predictors, and another for the OR-ed first-order predictors. If the best of the second-order features improves the ramp statistic by an amount greater than  $\Delta R_{min}$ , then it is selected. Otherwise  $\Delta R_{min}$  is halved, and the feature selection returns to searching for first-order features. In all experimentation described in this paper, only first and second-order features were considered, and  $b_2$  was set to 40.

An extensive set of experiments was undertaken to investigate the relationship between depth of search, and the resulting AUC values of AURORA. Results for “shallow”, “standard”, and “deep” depths of search are shown in Tables 1 and 2. The “standard” depth of search parameters are those discussed above, namely 20, 75, and 40. The “shallow” depth of search settings were  $\sim 2/3$  of the standard ones, and the “deep” depth of search settings were  $\sim 1.5$  times standard. The last figure in each row is an approximate standard error on the figures quoted for that dataset. (The standard error showed little variation as depth of search varied.) These results show that AURORA is highly insensitive to

these depth of search parameters over a wide range of values. The “standard” depth of search parameters were selected since it is clear that the benefit of implementing a depth of search any greater than this is very small.

Table 1: Effect of depth of search on AUC with 6k training observations

dataset	shallow	standard	deep	std. error
charity	57.97	57.95	58.02	0.19
churn	76.90	76.89	76.90	0.04
digit	95.32	95.42	95.34	0.11
forest	84.65	84.72	84.86	0.05
housing	98.12	98.12	98.13	0.01
married	84.10	84.09	84.23	0.06
migrant	68.84	68.77	68.91	0.13
weather	70.30	70.36	70.42	0.10

It is noteworthy that all of the testing of candidate features by APG happens without any need to change  $\tilde{\beta}$ , and hence without any need to resort the data. It is only once the short-list of candidates has been found, and they are tested by the midcontour point test and by a full gradient descent test, that  $\tilde{\beta}$  changes, and resorting is necessary. Hence the  $O(n \log(n))$  component of the algorithm is relatively minor, and, as stated in Sec. 2.3 will not dominate unless the amount of data is very large.

**3.2 Gradient Descent.** The gradient of the piecewise linear ramp function is constant within any piece. Hence the gradient of the ramp statistic may be computed very efficiently using the same method as that

Table 2: Effect of depth of search on AUC with 30k training observations

dataset	shallow	standard	deep	std. error
charity	59.63	59.65	59.69	0.06
churn	78.12	78.17	78.16	0.025
digit	96.68	96.80	96.79	0.02
forest	86.31	86.70	86.86	0.04
housing	98.36	98.40	98.43	0.01
married	85.44	85.61	85.66	0.05
migrant	70.89	70.91	70.97	0.11
weather	73.30	73.54	73.73	0.08

used for the ramp statistic itself in Sec. 2.3.

Recall that the gradient descent algorithm is constrained to the hypersphere  $\sum_i \beta_i^2 = B$ . Hence when performing gradient descent, we need to calculate the component of the gradient parallel to the hypersphere surface. The unconstrained gradient is first calculated. It is then projected onto the hypersphere surface to find its component in that direction (i.e. perpendicular to  $\vec{\beta}$ ). The secant method is used for gradient descent. It was found empirically that the secant method almost always converges in just a few iterations. Furthermore, in high dimensional space, the ramp statistic surface is nearly perfectly quadratic in the vicinity of the minimum, and the secant method converges in just one or two iterations. This fact significantly enhances the computational efficiency of the AURORA algorithm.

**3.3 Selection of Hypersphere Radius Value.** If the hypersphere radius,  $\sqrt{B}$ , is too small, then the arguments to the ramp function will generally be small, and the ramp statistic will be a poor estimator of the true AUC. Alternatively, if  $\sqrt{B}$  is too large, the ramp statistic approaches a sum of step functions, and therefore starts to contain many small regions that are nearly flat, connected by extremely steep inclines, much like terraces on a mountainside. Such a surface is hard to do gradient descent over. This tradeoff between poor approximation of the true AUC and creating steps in the ramp statistic surface has been observed by [16], and [5], both of whom encounter the same problem when using a sigmoid based approximation to the AUC. Yan et al. (2003) get around the problem by using a polynomial approximation to the heaviside function, while Herschtal & Raskutti employ a series of increasing values of  $B$ , so that by the time the problem of steps arises, most of the gradient descent has already been done. In the context of AURORA it was found that although the optimal value of  $B$  is not the same for all problems, the accuracy of the

Table 3: Pseudocode for the AURORA algorithm, using first-order and second-order terms

---

```

binarise all raw features (Sec. 2.1)
select the best single 1st-order binarised feature
while the stopping condition has not yet been met {
    identify the best 1st-order feature (Sec. 3.1)
    if it improves the ramp statistic by  $> \Delta R_{min}$  {
        select it (Sec. 3.1)
        do gradient descent (Sec. 3.2)
    }
}
else {
    identify the best 2nd-order feature (Sec. 3.1)
    if it improves the ramp statistic by  $> \Delta R_{min}$  {
        select it (Sec. 3.1)
        do gradient descent (Sec. 3.2)
    }
}
else
    halve  $\Delta R_{min}$ 
}
}
select the classifier with the best ramp statistic value on
the validation set (Sec. 3.5)
calculate the final  $\vec{\beta}$  value by pooling the training and
validation sets (Sec. 3.5)

```

---

solution is fairly insensitive to  $B$ , and a value of  $B$  that is close to optimal can be found for any dataset using simple binary interpolation with a very small number of iterations ( $< 5$ ). Therefore in AURORA this procedure was followed to find a value for  $B$  that is approximately optimal for each of the test datasets. For all experimentation described in this paper, the value of  $\sqrt{B}$  was in the range  $[1, 10]$ .

**3.4 Local Minima.** In performing gradient descent, one must contend with the possibility of local minima on the error surface. After extensive empirical testing involving tracing a wide range of contour lines across the ramp statistic surface in high dimensional problem spaces, no local minima were found. It is therefore expected that if there are local minima, AURORA's performance will not be significantly affected by them.

**3.5 Model Selection Rule.** Feature selection and gradient descent proceed as described in Secs. 3.1 and 3.2, until the stopping criterion is met. This constitutes the classifier generation phase of the AURORA algorithm. In this phase, only 3/4 of the training data is used, with the remaining 1/4 being set aside to form an independent validation, or hold-out set. This validation set is now used in the classifier selection phase of the

algorithm.

Recall that each iteration of feature selection and gradient descent in the classifier generation phase yields a new classifier, and hence a new value of AUC and of the ramp statistic on the validation set. These values form a series of ramp statistic and AUC values for progressively more complex classifiers. The values of the ramp statistic on the validation set are smoothed using a triangular smoothing filter of width 15. The classifier that is then selected is the one which results in the maximum ramp statistic value on the validation set after smoothing.

As a final post-processing step, the training and validation datasets are pooled together, and a final gradient descent run is performed on the resulting ramp statistic surface,  $R(\beta)$ , starting from the point in  $\beta$ -space where the gradient descent run stopped for the selected classifier. Performance of the classifier is measured by the AUC on an entirely independent, unseen test set.

## 4 Experimental Description and Results

Experiments were performed on eight real world datasets (Sec. 4.1) from different domains. Each of the four non-linear binary classifier inducers, namely AURORA, RankBoost, C5.0, and SVM (Sec. 4.2) was applied to each of the datasets. Some experimentation was also performed using the linear binary classifier inducers RankOpt and logistic regression (Sec. 4.3). The experimental method used is described in Sec. 4.4. Results are analysed in Secs. 4.5 and 4.6.

**4.1 Datasets. Forest Cover Type (forest):** Data was downloaded from the UCI KDD repository. Each observation contains attributes of a  $30 \times 30$  metre cell of forest, and classifies that cell into one of seven cover types based on the cell's dominant tree type. The two most populous classes (Spruce-Fir and Lodgepole Pine) were extracted, and the binary classification task consisted of distinguishing between these classes. A total of 10 predictors were used, these being the 10 continuous valued attributes supplied for the data.

**Housing Mortgage (housing):** Data was downloaded from the U.S. Census Bureau 5% Public Use Microdata Sample (PUMS) containing individual records of the characteristics of a 5% sample of housing units for the state of Florida. A dataset was extracted consisting of all housing units which had a mortgage. The binary classification task was to distinguish between those housing units for which the mortgage had been paid off, and those for which it hadn't. The 12 ordinal predictor variables included the total household income, the room and bedroom counts, rate costs (electricity, water and

gas), the property tax rate, insurance rate and property value.

**Telecommunications Churn (churn):** Data on mobile phone customers of a large telecommunications carrier was used to learn to distinguish between those that churned to a competitor in the following three months and those that didn't. After reducing the degree of imbalance,<sup>1</sup> the minority class was  $\sim 40\%$  of the data. A set of 31 ordinal variables was used for prediction, including billing and product information.

**Marital Status (married):** As for the housing mortgage dataset, data was downloaded from the U.S. Census Bureau PUMS. From this dataset a 1% sample of individual records from the state of California was extracted. The binary classification task was to distinguish between individuals who have been married (whether currently married or not), and individuals who have never been married. The predictors were 11 ordinal variables, including ones relating to age, education level, income, and working hours.

**Migrant Status (migrant):** A sample of U.S. census data from 1880 was downloaded from the Integrated Public Use Microdata Series (IPUMS). The binary classification task was to distinguish between individuals who were migrants to the U.S. ( $\sim 25\%$ ) and individuals who were not. The predictors were 9 ordinal variables, mostly relating to family demographics.

**Donations to a Charity (charity):** This dataset consists of the competition data used for the 1998 KDD Cup. The classification task was to distinguish between donors and non-donors to a charitable organisation, in response to a mail campaign. The 27 predictors used were a collection of statistics on issues such as history of past donations, and demographics of the potential donor's neighbourhood.

**Handwritten Digit Recognition (digit):** Data was downloaded from the MNIST handwritten digit database. Each observation in this dataset consists of a bitmap of  $28 \times 28$  grayscale values, representing a handwritten digit. Each observation was converted to lower resolution ( $7 \times 7$  pixels) to reduce the dimensionality of the problem. The classification task was to distinguish between the digit '0' and all other digits. To make the problem more challenging, only the top 3 rows of pixels (21 pixels) were used. Further, pixels near the corners which contain almost no information were discarded. The result was a 17 dimensional dataset.

<sup>1</sup>The proportion of minority class observations for this dataset was very small. Hence the imbalance of the data was reduced so as to include sufficient minority class observations without using a prohibitively large amount of majority class data. The AURORA algorithm does not require this re-balancing, but does require sufficient observations of each class for training.

**Weather Season Prediction (weather):** There is a grid of weather buoys in the equatorial region of the Pacific Ocean. These take meteorological measurements, including wind speed and direction, air and sea temperature, and humidity at regular intervals. The resulting data is available at the website of the Tropical Atmosphere Ocean project [9]. Hourly measurements of 8 weather characteristics for all buoys over the period from May 1999 to April 2000 were downloaded. The classification task was to distinguish meteorological readings made during the northern hemisphere Autumn months (October, November and December) from those made in other months.

**4.2 Algorithmic Settings.** The parametric settings with which RankBoost, the SVM, and C5.0 were run are described here.

Wherever possible, equivalence in parameter settings was maintained across the inducers. For example, the maximum number of feature selections for AURORA and RankBoost is the same. Another example is that just as RankBoost contains no facility for deleting a feature once it has been selected, so AURORA was designed in the same way.

**RankBoost:** RankBoost works by boosting decision stumps. At each stage (or “round”) of the algorithm, the boosting is done in order to maximise the AUC of the resulting classifier. One parameter that can be adjusted by the user is the number of boosting rounds. Each boosting round corresponds to the selection of a single binarised feature, which parallels the feature selection in AURORA. It was found that 200 boosting rounds is sufficient to achieve convergence for the classification problems described in Sec. 4.1. This also matches the maximum number of feature selections in AURORA.

As was the case for AURORA, 1/4 of the training data was set aside to be used as a validation set. The process of selecting the final classifier for RankBoost is then identical to that for AURORA, described in Sec. 3.5.

**SVM:** The primary parameters that an SVM user needs to set are the degree of the polynomial in the error term,  $p$ , and the penalty, commonly denoted by  $c$ , which is a trade-off between the training error and the margin of the decision boundary. In this work,  $p$  was set to 2, giving a square penalty error term, often denoted by SVM-L2.

The full set of experiments was run with 3 different values of  $c$ : 10, 1000, and 100,000. For each dataset, results for the best  $c$  value are shown.

**C5.0:** C5.0 classifies data observations by building decision trees which minimise classification error. Hence if the data are too heavily imbalanced, C5.0 will simply

achieve a low classification error by classifying all observations as majority class. Thus it was found that the C5.0 results were quite sensitive to data imbalance. To overcome this, all datasets were balanced before being presented to C5.0.

**4.3 Linear Modelling Algorithms.** It is also of interest to compare AURORA to inducers of linear binary classifiers. To this end, we choose for comparison RankOpt [5], and logistic regression. Comparison with RankOpt is of interest because RankOpt operates on a similar principle to AURORA. Namely, it has the same objective function, and performs optimisation using gradient descent. Comparison with logistic regression is of interest because of logistic regression’s ubiquitous application to binary classification tasks such as those considered here.

**4.4 Experimental Procedure.** To lend statistical significance to our results, it is desirable to apply each inducer to a large number of independent training sets, and average the AUCs of the resulting classifiers on unseen test sets. This necessitates that the training sets for multiple runs of each algorithm be mutually exclusive of one another. Hence each dataset was split into  $N$  mutually exclusive subsets, and in each run of each classifier, training was performed using one subset, and the resulting classifier was tested on the remaining  $N-1$  subsets. The rationale behind such a split between training and test data is as follows. We are interested in measuring the performance of the algorithms under conditions where the amount of training data available is limited, so in our experimentation, the amount of data used for training is kept relatively small. Naturally, however, we want to measure the true AUC of the trained classifiers as precisely as possible. By measuring classifier AUC on an unseen test set, which we have made as large as possible, our estimator of the AUC will be as precise an indicator as possible of the classifier’s true (infinite population) AUC value.

With the exception of the digit dataset, which contains 60,000 observations, and the migrant dataset, which contains just over 100,000 observations, each of the datasets contains a minimum of 180,000 observations. So from each of these datasets 180,000 observations were randomly selected. These were split randomly into 30 mutually exclusive subsets ( $N = 30$ ), with 6000 observations each. 30 runs of each non-linear classification algorithm (AURORA, RankBoost, C5.0, and SVM-L2) were then performed, each using a different one of the 30 subsets for training, and the other 29 for testing. Each subset was used exactly once as the training set, which means that all training runs used com-

pletely independent data. This yielded 30 test results for each inducer for each dataset. This experimental procedure was then repeated using a far larger amount of training data – 30,000 observations – meaning that each dataset was split into 6 mutually exclusive subsets. The experimental procedure for the digit dataset was identical except that  $N$  was divided by 3 for each training set size. For the migrant dataset,  $N$  was set to 16 at 6,000 training observations per subset, and 3 at 30,000 per subset.

Tables 4 and 5 show the results of the experiments with the non-linear inducers with 6,000 and 30,000 training observations respectively. For each dataset, the best performing inducer has its AUC score shown in bold if the difference in performance over all other inducers is significant (i.e.  $> 2$  standard deviations higher). Each value quoted in these tables is  $100\times$  the mean of the AUC on the test sets across all training subsets, for the stated training data quantity, dataset, and inducer.

The amount of improvement of AURORA over the other non-linear algorithms is shown in Table 6. The pairwise differences (on a subset by subset basis) between AURORA and each of the other algorithms were measured. The values in Table 6 are then the number of standard deviations by which AURORA outperforms the other algorithms. For 5 of the 8 datasets, the amount by which AURORA outperformed all other algorithms was always highly statistically significant ( $> 4$  standard deviations), for both dataset sizes. Results are shown for the other 3 datasets.

Some experimentation was also performed using the linear classification algorithms, with 6,000 training observations per subset. Table 7 compares AURORA with these with 6,000 training observations, for a random selection of the datasets.

**4.5 AUC Metric Comparison.** In this section we report results of an extensive comparison of AURORA with C5.0, the SVM and RankBoost over the 8 datasets described in Sec. 4.1. We also compare AURORA’s performance to that of RankOpt and logistic regression.

**4.5.1 Comparison with Non-Linear Inducers.** AURORA significantly outperforms RankBoost in most cases at the 95% confidence level. For charity with 30,000 training observations and migrant with 6,000, there is no statistically significant difference between the two. The one case in which RankBoost significantly outperforms AURORA is for churn with 6,000 observations.

AURORA significantly outperforms both the SVM and C5.0 for both training set sizes on all datasets.

Table 4: AUC of non-linear algorithms with 6k training observations

dataset	C5.0	SVM	RB	AURORA
charity	52.92	56.11	57.17	<b>57.95</b>
churn	72.33	76.82	<b>77.39</b>	76.89
digit	91.15	92.86	92.73	<b>95.42</b>
forest	78.99	84.10	83.96	<b>84.72</b>
housing	94.98	97.73	97.68	<b>98.12</b>
married	79.50	83.37	83.63	<b>84.09</b>
migrant	65.14	68.42	68.75	68.77
weather	65.68	65.73	66.34	<b>70.36</b>

Table 5: AUC of non-linear algorithms with 30k training observations

dataset	C5.0	SVM	RB	AURORA
charity	53.85	58.57	59.56	59.65
churn	73.55	77.35	77.91	<b>78.17</b>
digit	94.33	93.04	93.28	<b>96.80</b>
forest	83.63	86.13	84.32	<b>86.70</b>
housing	96.25	97.81	97.80	<b>98.40</b>
married	80.75	83.74	84.06	<b>85.61</b>
migrant	66.27	69.14	69.33	<b>70.91</b>
weather	70.65	66.14	67.41	<b>73.54</b>

AURORA’s results for the digit and weather datasets are particularly impressive.

**4.5.2 Comparison with Linear Inducers.** It is natural to expect that AURORA (as well as any other non-linear inducer) will outperform linear inducers whenever there is a significant amount of non-linearity in the data. For certain datasets, either the logistic regression algorithm available or RankOpt failed to produce a result. For all other datasets results are shown in Table 7. In all cases, AURORA’s performance was significantly better than that of the linear inducers. The gap is particularly large for the weather and migrant datasets.

**4.6 Computational Efficiency.** As discussed in Sec. 2.3, AURORA achieves computational efficiency by virtue of being based on a piecewise linear function that can essentially be computed in linear time. The theory that AURORA executes in  $O(n)$  time is now put to the

Table 6: Improvement of AURORA over other non-linear inducers

dataset	training size = 6k			training size = 30k		
	C5.0	SVM	RB	C5.0	SVM	RB
charity	20.8	13.4	3.9	19.6	7.2	1.0
churn	21.2	2.5	-17.2	13.6	26.1	6.3
migrant	14.5	2.9	0.2	10.0	46.1	20.1

Table 7: Performance of AURORA and linear inducers

dataset	RankOpt	LogRes	AURORA
forest	83.53	83.63	84.72
housing	96.94	96.97	98.12
married	83.41	83.33	84.09
migrant	65.31	64.83	68.77
weather	62.25	61.78	70.36

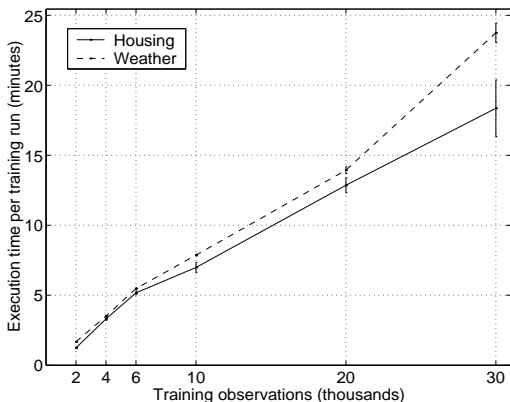


Figure 3: AURORA scalability with increasing number of observations

test by computing actual execution times per training run as the number of training observations increases. Execution times are shown for two datasets, namely weather and housing, at 6 dataset training sizes, in Figure 3. This bears out the theory that the actual complexity is close to  $O(n)$  over a wide range of dataset sizes.

Another aspect of scalability which is of interest, is scalability with increasing numbers of raw predictor features. To measure AURORA’s scalability in this dimension, we take the churn dataset, which is the widest of the 8 datasets used for experimentation, and progressively prune out more and more of the least informative predictors. Execution times per training run can then be measured for different numbers of raw predictors, to create problems of 5 different widths. Figure 4 shows that execution time increases only very gradually as the number of predictors increases. This is because AURORA short lists the candidate predictors using a fast heuristic (the APG), and the length of the short list is constant, regardless of the number of predictors. In both Figures 3 and 4 the error bars represent 95% confidence limits.

It has thus been demonstrated that AURORA is highly scalable both as the number of training observations increases, and as the number of dimensions increases.

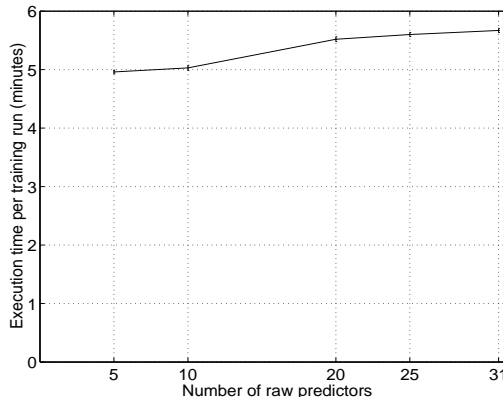


Figure 4: AURORA scalability with increasing number of predictors

## 5 Conclusion and Future Work

We have introduced AURORA, a non-linear binary classifier which optimises AUC. AURORA has been compared to several other non-linear binary classifiers, and in a clear majority of cases was found to significantly outperform them.

It has been shown that AURORA is, over a wide range of amount of training data, effectively an  $O(n)$  algorithm, and is hence scalable to very large amounts of training data.

Currently, classifier selection is done via a hold-out set. This is widely regarded as a sub-optimal use of the training data [7]. Future work will explore the use of alternate methods, such as cross-validation.

## Acknowledgments

The permission of the Managing Director, Telstra Research Laboratories (TRL), to publish this paper is gratefully acknowledged. The technical advice of Herman Ferra, TRL, is gratefully acknowledged.

## References

- [1] D. Bamber. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *J. Math. Psych.*, 12:387 – 415, 1975.
- [2] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.
- [3] C. Cortes and M. Mohri. AUC Optimization vs. Error Rate Minimization. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge MA, 2004.
- [4] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Prefer-

- ences. *Journal of Machine Learning Research*, 4:933–969, 2004.
- [5] A. Herschtal and B. Raskutti. Optimising Area Under the ROC Curve Using Gradient Descent. In *Proceedings of the Twenty-first International Conference on Machine Learning ICML04*, 2004.
- [6] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the Tenth European Conference on Machine Learning ECML98*, 1998.
- [7] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, 1995.
- [8] H. B. Mann and D. R. Whitney. On a test whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18:50 – 60, 1947.
- [9] M. J. McPhaden. <http://www.pmel.noaa.gov/tao>, 2005.
- [10] C. Perlich, F. Provost, and J. S. Simonoff. Tree Induction vs. Logistic Regression: A Learning Curve Analysis. *Journal of Machine Learning Research*, 4:211–255, 2003.
- [11] J. Platt. Fast training of support vector machines using sequential minimal optimization, 1998.
- [12] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.
- [13] J. R. Quinlan. <http://www.rulequest.com/see5-unix.html>, 2004.
- [14] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.
- [15] G. M. Weiss and F. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19(2003):315 – 354, 2003.
- [16] L. Yan, R. Dodier, M. C. Mozer, and R. Wolniewicz. Optimizing classifier performance via approximation to the wilcoxon-mann-whitney statistic. In *Proceedings of the Twentieth Intl. Conf. on Machine Learning*, pages 848 – 855. AAAI Press, Menlo Park, CA, 2003.