Ranking Instances by Maximizing the Area under ROC Curve

H. Altay Güvenir, *Member*, *IEEE*, and Murat Kurtcephe

Abstract—In recent years, the problem of learning a real-valued function that induces a ranking over an instance space has gained importance in machine learning literature. Here, we propose a supervised algorithm that learns a ranking function, called ranking instances by maximizing the area under the ROC curve (RIMARC). Since the area under the ROC curve (AUC) is a widely accepted performance measure for evaluating the quality of ranking, the algorithm aims to maximize the AUC value directly. For a single categorical feature, we show the necessary and sufficient condition that any ranking function must satisfy to achieve the maximum AUC. We also sketch a method to discretize a continuous feature in a way to reach the maximum AUC as well. RIMARC uses a heuristic to extend this maximization to all features of a data set. The ranking function learned by the RIMARC algorithm is in a human-readable form; therefore, it provides valuable information to domain experts for decision making. Performance of RIMARC is evaluated on many real-life data sets by using different state-of-the-art algorithms. Evaluations of the AUC metric show that RIMARC achieves significantly better performance compared to other similar methods.

Index Terms—Ranking, data mining, machine learning, decision support

1 Introduction

In this paper, we propose a binary classification methodology that ranks instances based upon how likely they are to have a positive label. Our method is based on receiver operating characteristic (ROC) analysis, and attempts to maximize the area under the ROC curve (AUC); hence, the algorithm is called ranking instances by maximizing the area under ROC curve (RIMARC). The RIMARC algorithm learns a ranking function which is a linear combination of nonlinear score functions constructed for each feature separately. Each of these nonlinear score functions aims to maximize the AUC by considering only the corresponding feature in ranking. All continuous features are first discretized into categorical ones in a way that optimizes the AUC. Given a single categorical feature, it is possible to derive a scoring function that achieves the maximum AUC. We show the necessary and sufficient condition that such a scoring function for a single feature has to satisfy for achieving the maximum AUC. Computing the score function for a categorical feature requires only one pass over the training data set. Missing feature values are simply ignored. The AUC value, obtained on the training data, for a single feature, reflects the effect of that feature in the correct ranking. Using these AUC values as weights, the RIMARC algorithm combines these score functions, learned for each feature, into a single ranking function.

The main characteristics of the RIMARC algorithm can be summarized as follows: It achieves comparably high AUC values. Its time complexity for both learning and applying the ranking function is relatively low. Being a nonparametric method, it does not require tuning of parameters to achieve the best performance. It is robust to missing feature values. Finally, the ranking function learned is in a human readable form that can be easily interpreted by domain experts, listing the effects (weight) of features and how their particular values affect the ranking. The RIMARC algorithm is simple and easy to implement. In cases where a data set is collected from experiments for research purposes, the researchers may be more interested in the effects of the features and their particular values on ranking than the particular ranking function.

In the next section, the ranking problem is revisited. Section 3 covers ROC, AUC and research on AUC maximization. In Section 4, the RIMARC method and implementation details are given. Section 5 presents the empirical evaluation of RIMARC on real-world data sets. Section 6 discusses the related work. Finally, Section 7 concludes with some suggestions for future work.

2 Ranking

The ranking problem can be viewed as a binary classification problem with additional ordinal information. In the binary classification problem, the learner is given a finite sequence of labeled training examples $z = ((x_1, y_1), \ldots, (x_n, y_n))$, where the x_i are instances in some instance space X and the y_i are labels in $Y = \{\mathbf{p}, \mathbf{n}\}$, and the goal is to learn a binary-valued function $h : X \rightarrow Y$ that predicts accurately labels of future instances.

The problem of finding a function that ranks positive instances higher than the negative ones is referred as the *bipartite* ranking problem. Here, a training data set D, from

• *The authors are with the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey.*
E-mail: guvenir@cs.bilkent.edu.tr, kurtcephe@gmail.com.

Manuscript received 26 Feb. 2012; revised 15 Oct. 2012; accepted 17 Oct. 2012; published online 23 Oct. 2012.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2012-02-0128.
Digital Object Identifier no. 10.1109/TKDE.2012.214.

an instance space X, is given, where the instances come from a set of two categories, positive and negative, represented as {**p**, **n**}. Using D, the goal is to learn a ranking function $r : X \rightarrow \mathbb{R}$ that ranks future positive instances higher than negative ones. In other words, the function r is expected to assign higher values to positive instances than to negative ones. Then, the instances can be ordered using the values provided by the ranking function.

3 Introduction to ROC Analysis

The ROC graph is a tool that can be used to visualize, organize, and select classifiers based on their performance [22]. It has become a popular performance measure in the machine learning community after it was realized that accuracy is often a poor metric to evaluate classifier performance [33], [41], [42].

The literature on ROC is more established to deal with binary classification problems than multiclass ones. At the end of the classification phase, some classifiers simply map each instance to a class label (discrete output). Some other classifiers, such as naïve Bayes or neural networks are able to estimate the probability of an instance belonging to a specific class (continuous valued output).

Binary classifiers produce a discrete output represented by only one point in the ROC space, since only one confusion matrix is produced from their classification output. Continuous-output-producing classifiers can have more than one confusion matrix by applying different thresholds to predict class membership. All instances with a score greater than the threshold are predicted to be **p** class and all others are predicted to be **n** class. Therefore, for each threshold value, a separate confusion matrix is obtained. The number of confusion matrices is equal to the number of ROC points on an ROC graph. With the method proposed by Domingos [15], it is possible to obtain ROC curves even for algorithms that are unable to produce scores.

Let a set of instances, labeled as **p** or **n**, be ranked by some scoring function. Given a threshold value τ, instances whose score is below τ are predicted as **n**, and those with score higher than τ are predicted as **p**. For a given threshold value, *TP* is equal to the number of positive instances that have been classified correctly and *FP* is equal to the number of negative instances that have been misclassified.

The ROC graph can be plotted as the fraction of true positives out of the positives (*TPR* = true positive rate) versus the fraction of false positives out of the negatives (*FPR* = false positive rate). The values of *TPR* and *FPR* are calculated by using (1). In this equation, N is the number of total negative instances and P is the number of total positive instances

$$TPR = \frac{TP}{P}, \quad FPR = \frac{FP}{N}. \tag{1}$$

For each possible distinct τ value, a distinct (*FPR*, *TPR*) value is computed. The ROC space is a 2D space with a range of [0, 1] on both axes. In ROC space the vertical axis represents the true positive rate of a classification output, while the horizontal axis represents the false positive rate. That is, each (*FPR*, *TPR*) pair corresponds to a point on the ROC space. In a data set with s distinct classifier scores, there are $s + 1$ ROC points, including the trivial (0, 0) and (1, 1) points.

Although ROC graphs are useful for visualizing the performance of a classifier, a scalar value is needed to compare classifiers. Bradley [5] proposes the area under the ROC curve as a performance measure.

The ROC graph space is a one-unit square. The highest possible AUC value is 1.0, which represents the perfect ordering. In ROC graphs, a 0.5 AUC value means random guessing has occurred and values below 0.5 are not realistic as they can be negated by changing the decision criteria of the classifier.

The AUC value of a classifier is equal to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [22].

In an empirical ROC curve, the AUC is usually estimated by the trapezoidal rule. According to this rule, trapezoids are formed using the observed points as corners, the areas of these trapezoids are computed and then they are added up. Fawcett [22] proposed efficient algorithms for generating the ROC points and computing the AUC.

It can be shown that the area under the ROC curve is closely related to the Mann-Whitney U, which tests whether positives are ranked higher than negatives. AUC is also equivalent to the Wilcoxon test of ranks [30].

4 RIMARC

RIMARC is a simple, yet powerful, ranking algorithm designed to maximize the AUC metric directly. The RIMARC algorithm reduces the problem of finding a ranking function for the whole set of features into finding a ranking function for a single categorical feature, and then combines these functions to form one covering all features. We will show that it is possible to determine a ranking function that achieves the maximum possible AUC for a single categorical feature. During the training phase, the RIMARC algorithm first discretizes the continuous features by a method called MAD2C, proposed by Kurtcephe and Guvenir [35]. The MAD2C method discretizes a continuous feature in such a way that results in a set of categorical values so that the AUC of the new categorical feature is the maximum. At this point, all the features are categorical. Then, the score for a value v of a feature f is assigned to be the probability of a training instance that has the value v for the feature f to have the class label **p**. During the computation of the score values, instances whose value for the feature f is missing are simply ignored. For each feature, the values are sorted in the increasing order of their score, and the AUC is computed. Finally, the weight of a feature f is computed as, $w_f = 2(AUC(f) - 0.5)$ where *AUC*(f) is the AUC obtained on the feature f. That is, the weight of a feature is a linear function of its AUC value calculated using the training instances whose value for that feature is known. A higher value of AUC for a feature is an indication of its higher relevance in determining the class label. For example, if the AUC computed for a feature f is 1, than it means that all instances in the training set can be ranked by using only the values of f. Hence, we can expect that new query instances can also be ranked correctly by

using f only. The training method of the RIMARC algorithm is given in Algorithm 2.

For a given query, q, the ranking function, $r()$, of the RIMARC algorithm returns a real value $r(q)$ in the range of [0, 1]. This value $r(q)$ is roughly the probability that the instance q has the class label **p**. It is only a rough estimate of the probability, since it is very likely that no other instance with exactly the same feature values has been observed in the training set. The ranking function of the RIMARC algorithm determines this estimated probability by computing the weighted average of probabilities computed on single features, as shown in Algorithm 1.

Algorithm 1. The ranking function.

```
Input: q, score[F][V], weight[F]
// q: query instance,
// F: number of features, V: number of values
Output: r(q): the ranking value of q.
totalScore ← 0;
totalWeight ← 0;
for f ← 1 to F do
    if q_f is known then
        totalScore ← totalScore + weight[f]*score[f][ q_f];
        totalWeight ← totalWeight + weight[f]
end
return totalScore / totalWeight
```

Algorithm 2. Training in RIMARC.

```
Input: trainSet[F][T]
// F: number of features, T: number of instances
Output: score[F][V], weight[F]
//score: score for each value v of each feature f
//weight: weight for each feature
for f ← 1 to F do
    if numerical(f) then
        cutPoints[]← MAD2C(trainSet[f]);
        convNumValuesToCatValues(cutPoints[], trainSet[f]);
    end
    // V is the set of categoric values of f
    for v ← 1 to V do
        score[f][v] ← Pr[p | p_f=v]; // p ∈ trainSet
    //sort elements of V by their increasing score[f] values
    sort(V, score[f]);
    AUC[f] ← computeAUC(score[f]);
    weight[f] ← 2(AUC[f] – 0.5)
end
```

In the following sections, we will show how the ranking function and the score values can be defined for a single categorical feature.

4.1 Single Categorical Feature Case

A categorical feature has a finite set of choices as its values. Let $V = \{v_1, \ldots v_k\}$ be a the set of categorical values for a given feature. In that case, the training data set D is a set of instances represented by a vector of feature value and class label as $<v_i, c>$, where $v_i \in V$ and $c \in \{\mathbf{p}, \mathbf{n}\}$. A ranking function $r : V \rightarrow$ [0, 1] can be defined to rank the values in V. According to this ranking function, a value v_j comes after a value v_i if and only if $r(v_i) < r(v_j)$; hence, r defines a total ordering on the set V, that is $v_i \leq v_j$. A pair of consecutive values v_i and v_{i+1} defines an ROC point R_i on the ROC space. The coordinates of the point R_i are (FPR_i, TPR_i). The instances of D can then be ranked according to the values of $r(v_i)$ corresponding to their feature values, v_i.

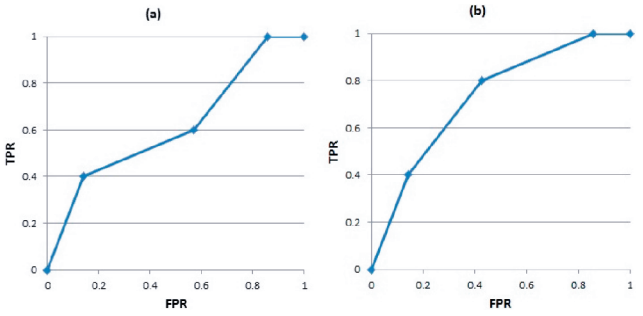

Fig. 1. Effect of swapping the values of the ranking function of two feature values.

Let D be a data set with a single categorical feature whose value set is $V = \{v_1, \ldots, v_k\}$, and $r : V \rightarrow$ [0, 1] be the ranking function that orders the values of V, such that $v_i \leq v_{i+1}$ if $r(v_i) < r(v_{i+1})$, for all values of $1 \leq i \leq k$. Since there is only one feature, this function $r()$, ranks the instances, directly.

It is interesting to note that, if the values of the ranking function for two consecutive values v_i and v_{i+1} are swapped, then the only change in the ROC curve is that the ROC point corresponding to the v_i and v_{i+1} values moves to a new location so that the slopes of the line segments adjacent to that ROC point are swapped.

For example, consider a data set with a single categorical feature, given as $D = \{(\mathrm{a}, \mathbf{n}), (\mathrm{b}, \mathbf{p}), (\mathrm{b}, \mathbf{n}), (\mathrm{b}, \mathbf{n}), (\mathrm{b}, \mathbf{n}), (\mathrm{c}, \mathbf{p}), (\mathrm{c}, \mathbf{p}), (\mathrm{c}, \mathbf{n}), (\mathrm{c}, \mathbf{n}), (\mathrm{d}, \mathbf{p}), (\mathrm{d}, \mathbf{p}), (\mathrm{d}, \mathbf{n})\}$, where $V = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$. If a ranking function r orders the values of V as $\mathrm{a} \leq \mathrm{c} \leq \mathrm{b} \leq \mathrm{d}$, with $r(\mathrm{a}) < r(\mathrm{c}) < r(\mathrm{b}) < r(\mathrm{d})$, the ROC curve shown in Fig. 1a will be obtained. If the ranking function is modified so that values of $r(\mathrm{b})$ and $r(\mathrm{c})$ are swapped, the ROC curve shown in Fig. 1b will be obtained. A similar technique was used earlier by earlier by Flach and Wu [24] to create better prediction models for classifiers.

This property of the ROC spaces helps to remove the concavities in an ROC curve, resulting in a larger AUC. To obtain the maximum AUC value, the ROC curve has to be convex. Hence, we will show how to construct the ranking function so that the resulting ROC curve is guaranteed to be convex.

Note that the slope of the line segment between two consecutive ROC points R_i and R_{i+1} is

$$s_i = \frac{TPR_i - TPR_{i+1}}{FPR_i - FPR_{i+1}}. \tag{2}$$

In order for the ROC curve to be convex, the slopes of all line segments connecting consecutive ROC points starting from the trivial ROC point (1, 1) must be nondecreasing, as shown in Fig. 2. Therefore, the condition for a convex ROC curve is

$$\underset{1 \leq i < k}{\forall i} \quad s_i \geq s_{i+1}. \tag{3}$$

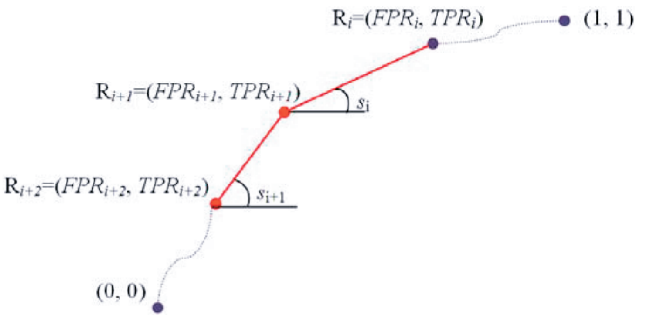

Fig. 2. Relation between the slopes of two consecutive line segments in a convex ROC curve.

Using (2),

$$\underset{1\le i<k}{\forall i} \quad \frac{TPR_{i+1}-TPR_{i+2}}{FPR_{i+1}-FPR_{i+2}} \ge \frac{TPR_i-TPR_{i+1}}{FPR_i-FPR_{i+1}}.$$

By definition, $TPR_i = \frac{TP_i}{P}$, where TP_i is the number of true positives with value v_i. Further, due to the ordering of values, $TP_i = P_i + TP_{i+1}$, where P_i is the number of **p**-labeled instances with value v_i. Hence,

$$\begin{aligned} TPR_i - TPR_{i+1} &= \frac{TPR_i}{P} - \frac{TPR_{i+1}}{P} \\ &= \frac{1}{P}\left(P_i + TP_{i+1} - TP_{i+1}\right) = \frac{P_i}{P}. \end{aligned}$$

Similarly,

$$FPR_i - FPR_{i+1} = \frac{N_i}{N}.$$

$TPR_{i+1} - TPR_{i+2} = \frac{P_{i+1}}{P}$ and

$$FPR_{i+1} - FPR_{i+2} = \frac{N_{i+1}}{N}.$$

Therefore, the condition in (3) can be rewritten as

$$\underset{1\le i<k}{\forall i} \quad \frac{P_{i+1}/P}{N_{i+1}/N} \ge \frac{P_i/P}{N_i/N}.$$

Finally,

$$\underset{1\le i<k}{\forall i} \quad \frac{P_{i+1}}{N_{i+1}} \ge \frac{P_i}{N_i}. \tag{4}$$

That is, to obtain a convex ROC curve, the condition in (4) must be satisfied for all ROC points. In other words, to achieve the maximum AUC, the ranking function has to satisfy the following condition:

$$\underset{1\le i<k}{\forall i} \; r(v_{i+1}) > r(v_i) \text{ iff } \frac{P_{i+1}}{N_{i+1}} \ge \frac{P_i}{N_i}. \tag{5}$$

Further, any ranking function that satisfies (5) will result in the same ROC curve and achieve the maximum AUC. For example, the ranking function defined as $r(v_i) = \frac{P_i}{N_i}$ will result in a convex ROC curve.

It is also important to note that, given a data set with a single categorical feature, there exists exactly one convex ROC curve, and it corresponds to the best ranking.

The general assumptions for ranking problems are given below:

$$\underset{1\le i\le k}{\forall i} \; P_i \ge 0, \qquad \underset{1\le i\le k}{\forall i} \; N_i \ge 0,$$
$$P = \sum_1^k P_i > 0, \quad N = \sum_1^k N_i > 0.$$

Although the data set is guaranteed to have at least one instance with class label **p** and one instance with label **n**, it is possible that for some values of i, N_i may be 0. In such cases the ranking function defined as $r(v_i) = P_i/N_i$ will have an undefined value. To avoid such problems, the RIMARC algorithm defines the ranking function as

$$r(v_i) = \frac{P_i}{P_i + N_i}. \tag{6}$$

Note that, since $\forall i_{1\le i\le k}\; P_i + N_i > 0$, $r(v_i)$ is defined for all values of i. To see that the ranking function defined in (6) satisfies the condition in (5), note that if

$$\underset{0\le i<n}{\forall i} \quad \frac{P_{i+1}}{P_{i+1}+N_{i+1}} \ge \frac{P_i}{P_i+N_i},$$

then

$$\underset{0\le i<n}{\forall i} \quad P_{i+1}(P_i + N_i) \ge P_i(P_{i+1} + N_{i+1}),$$

and

$$\underset{0\le i<n}{\forall i} \quad \frac{P_{i+1}}{N_{i+1}} \ge \frac{P_i}{N_i}.$$

The ranking function given in (6) has another added benefit, in that it is simply the probability of the **p** label among all instances with value v_i. Such a probability value is easily interpretable by humans.

However, note that the commonly used Laplace estimate, defined as $\frac{P_i+1}{P_i+N_i+2}$ does not satisfy the condition in (5).

4.1.1 The Effect of the Choice of the Class Labeling on the AUC

To calculate the P and N values, one of the classes should be labeled as **p** and the other class as **n**, but one can question the effect this choice has on the AUC value. It is possible to show that the AUC value of a categorical feature is independent from the choice of class labels by using the value from the Wilcoxon-Mann-Whitney statistics.

In (7), the AUC formula based on the Wilcoxon-Mann-Whitney statistics is given. The set D_p represents the **p**-labeled instances and D_n represents the **n**-labeled instances. D_{pi} is the ranking of the ith instance in the D_p set, similarly, D_{nj}, is the ranking of the ith instance in the D_p set

$$AUC = \frac{\sum_{i=1}^{P}\sum_{j=1}^{N} f(D_{pi}, D_{nj})}{PN},$$
$$f = \begin{bmatrix} 1 & \text{if} & D_{pi} > D_{nj} \\ 0 & \text{if} & D_{pi} < D_{nj} \\ 0.5 & \text{if} & D_{pi} = D_{nj} \end{bmatrix}. \tag{7}$$

The dividend part of the AUC formula in (7) counts the number of **p**-labeled instances for each element of the D_p set whose ranking is higher than any element of the D_n set. Then, AUC is calculated by dividing this summation by the multiplication of the **p**-labeled and **n**-labeled elements.

It is straightforward that the divisor part of the AUC formula is independent of the choice of the class labels.

TABLE 1
Toy Training Data Set with One Categorical Feature

Class label	**n**	**n**	**p**	**n**	**n**	**p**	**n**	**p**	**p**	**n**	**p**	**p**	**p**
Feature value	a	a	a	a	b	b	b	c	c	c	d	d	d

Assume that the ranking function given in (6) is used on the data set D, and D_p and D_n sets are formed. Let n_i be the number of **n**-labeled instances whose ranking is lower than the ith element of the D_p set and let r_i be the score assigned to this element. When the classes are swapped, the new ranking score r'_i is equal to $1 - r_i$. Using this property, all instance scores are subtracted from 1. However, this operation simply reverses the ranking of the instances. So the formula in (7), which calculates the value of AUC, using the ranking of the instances, is independent of the choice of class labeling.

4.1.2 An Example Toy Data Set

Consider a toy training data set with a single categorical feature given in Table 1. To calculate the AUC value of this particular feature, assume that the ranking scores are calculated by the function in (6). The ranking scores of the categorical values are as follows: $r(\mathrm{a}) = 0.25, r(\mathrm{b}) = 0.33, r(\mathrm{c}) = 0.67$, and $r(\mathrm{d}) = 1$. The version of the data set sorted by the ranking function is given in Table 2. In this example, the value of P is 7 and the value of N is 6. The AUC value of this feature is calculated by using (7), as $\frac{34.5}{7*6} = 0.82$. When the class labels are swapped (**n** labels are replaced by **p** labels and vice versa) the ranking scores are also swapped. The sorted version of the swapped toy data set is given in Table 3. Since the relative ranking of the instances does not change, the AUC value of the new ranking is also is 0.82.

4.2 Handling Continuous Features

Having determined the requirement for a ranking function for a categorical feature to achieve the maximum possible AUC, the next problem is to develop a mechanism for handling the continuous features. An obvious and trivial ranking function maps each real value seen in the training set with the class label **p** to 1 and each real value with the class label **n** to 0. This risk function will result in the maximum possible value for AUC, which is 1.0. However, such a risk function will over fit the training data, and will be undefined for unseen values of the feature, which are very likely to be seen in a query instance. The first requirement for a risk function for a continuous feature is that it must be defined for all possible values of that continuous feature. A straightforward solution to this requirement is to discretize the continuous feature by grouping all consecutive values with the same class value

TABLE 2
Toy Training Data Set with Score Values

Score	0.25	0.25	0.25	0.25	0.33	0.33	0.33	0.67	0.67	0.67	1.0	1.0	1.0
Class label	**n**	**n**	**p**	**n**	**n**	**p**	**n**	**p**	**p**	**n**	**p**	**p**	**p**
Feature value	a	a	a	a	b	b	b	c	c	c	d	d	d

The ranking scores of the instances are calculated and they are sorted in ascending order.

TABLE 3
Training Data Set with Class Labels Swapped

Score	0	0	0	0.33	0.33	0.33	0.67	0.67	0.67	0.75	0.75	0.75	0.75
Class label	**n**	**n**	**n**	**n**	**n**	**p**	**p**	**n**	**p**	**p**	**p**	**n**	**p**
Feature value	d	d	d	c	c	c	b	b	b	a	a	a	a

The ranking scores are recalculated and instances are sorted in ascending order.

to a single categorical value. The cut off points can be set to the middle point between feature values of differing class labels. The ranking function, then, can be defined using the function given in (6) for categorical features. Although this would result in a ranking function that is defined for all values of a continuous function, it would still suffer from the over fitting problem. To overcome this problem, the RIMARC algorithm makes the following assumption.

Assumption. *For a continuous feature, either the higher values are indicators of the* **p** *class and lower values are indicators of the of the* **n** *class, or vice versa.*

Although there exist some features in real-world domains that do not satisfy this assumption, in the data sets we examined this assumption is satisfied in general.

This assumption is also consistent with the interpretations of the values of continuous features in many real-world applications. For example, in a medical domain, a high value of fasting blood glucose is an indication for diabetes. On the other hand, low fasting blood glucose is an indication of another health problem, called hypoglycemia.

4.2.1 The MAD2C Method

As explained above, the RIMARC algorithm requires all features to be categorical. Therefore, the continuous features in a data set need to be converted into categorical ones, through discretization. The aim of a discretization method is to find the proper cut-points to categorize a given continuous feature.

The MAD algorithm given in [35] is defined for multiclass data sets. It is designed to maximize the AUC value by checking the ranking quality of values of a continuous feature. A special version of the MAD algorithm, called MAD2C, defined for two-class problems, is used in RIMARC.

The MAD2C algorithm first sorts the training instances in ascending order. Sorting is essential for all discretization methods to produce intervals that are as homogenous as possible. After the sorting operation, feature values are used as hypothetical ranking score values and the ROC graph of the ranking on that feature is constructed. The AUC of the ROC curve indicates the overall ranking quality of the continuous feature. To obtain the maximum AUC value, only the points on the convex hull must be selected. The minimum number of points that form the convex hull is found by eliminating the points that cause concavities on the ROC graph. In each pass, the MAD2C method compares the slopes in the order of the creation of the hypothetical lines, finds the junction points (cut-points) that cause concavities and eliminates them. This process is repeated until there is no concavity remaining on the graph. The points left on the graph are the cut-points, which will be used to discretize the feature.

TABLE 4
A Toy Data Set for Visualizing MAD2C

Class value	n	n	p	n	n	p	n	p	p	n	p	p	p
F1	1	2	3	4	5	6	6	7	8	9	10	11	12

It has been proven that the MAD2C method finds the cut-points that will yield the maximum AUC. It is also shown that the cut-points found by MAD2C never separate two consecutive instances of the same class. This is an important property, expected from a discretization method. The MAD2C method is preferred among other discretization methods since it has been shown in empirical evaluations that MAD2C has lower running time in two-class data sets and helps the classification algorithms to achieve a higher performance than other well-known discretization methods on AUC basis [35].

4.2.2 A Toy Data Set Discretization Example

To visualize the discretization process using the MAD2C method, a toy data set is given in Table 4. After the sorting operation, the ROC points are formed. The ROC graph for the example in Table 4 is given in Fig. 3. In this example, the higher F1 values are indicators of the **p** class. If the lower values of feature F1 were indicators of the **p** class, then the **p** and **n** class labels would be swapped and the ROC graph below the diagonal would have been obtained. Note that these two ROC graphs are symmetric about the diagonal.

The first pass of the MAD2C algorithm is shown in Fig. 4. All points below or on the diagonal are ignored since they have no positive effect on the maximization of AUC. Then the points causing concavities are eliminated. MAD2C converged to the convex hull in one pass for this example, as shown in Fig. 4. The set of points left on the graph are reported as the discretization cut-points.

4.3 The RIMARC Algorithm

The training phase of the RIMARC algorithm is given in Algorithm. 1. In the training phase, first, using the MAD2C algorithm, all continuous features are discretized and converted into categorical features. Ranking scores are calculated for each value of a given categorical feature, including discretized continuous features. In this step, the ranking function defined in (6) is used to obtain the optimal ranking for the features. Then, the training instances are sorted according to their ranking values computed in the previous step. Since the ranking function used by RIMARC always results in a convex ROC curve, the AUC is always equal to or greater than 0.5.

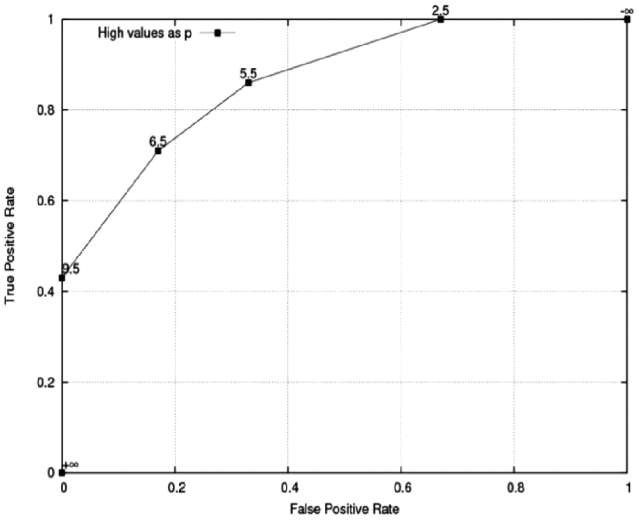

Fig. 4. Final cut-points after the first pass of convex hull algorithm.

The AUC of a feature indicates its relevance in the ranking of an instance. For example, if a feature has 1 as its AUC, than the ranking score of a new instance can be computed by considering only that feature. On the other hand, a feature with 0.5 as its AUC should be ignored. Therefore, the RIMARC algorithm uses a function of AUC to determine the weight of a feature in ranking. Since it is easier for human experts to interpret the weight values in the range of 0 to 1, the RIMARC algorithm uses a weight w_f for a feature f computed as

$$w_f = 2(AUC_f - 0.5). \tag{8}$$

The ROC curve of an irrelevant feature is simply a diagonal line from $(0, 0)$ to $(1, 1)$, with $AUC = 0.5$. The weight function in (8) assigns 0 to such irrelevant features to ignore them in computing the ranking function. The score values and weights of the features are stored for the querying phase.

The model learned for the single function for the toy data set in Table 4 is shown in Fig. 5. The model contains the weight of the feature and a nonlinear ranking function.

The testing phase of the RIMARC method is straightforward, as shown in Algorithm 2. For each feature, the ranking score corresponding to the value of the feature in

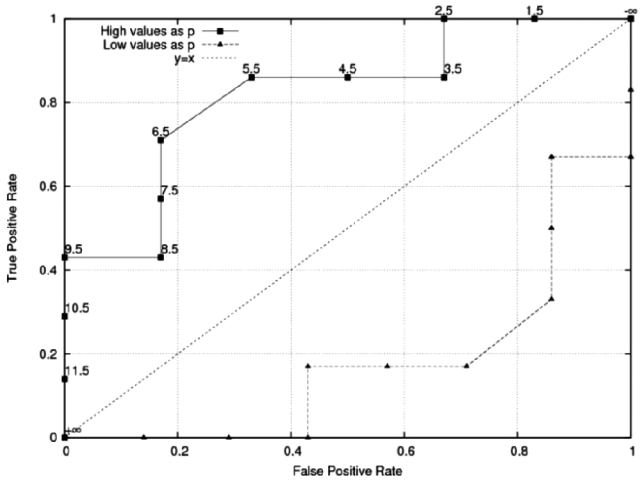

Fig. 3. Visualization of the ROC points in two-class discretization.

```
F1: weight= 0,738
If F1="9.5<",     score=1.0
If F1="6.5..9.5", score=0.667
If F1="5.5..6.5", score=0.5
If F1="2.5..5.5", score=0.333
If F1="<2.5",     score=0.0
```

Fig. 5. The model learned for the single feature in Table 5.

the query instance is used. The ranking score of this feature is weighted by its weight computed in the training phase. The computation of the ranking function for a query instance q is given in (9).

$$r(q) = \frac{\sum_f w_f.\ \Pr(\mathbf{p} \mid q_f)}{\sum_f w_f}, \qquad w_f = \begin{cases} 2(AUC_f - 0.5), & q_f \text{ is known}, \\ 0, & q_f \text{ is missing}. \end{cases} \tag{9}$$

The maximization of AUC for the whole feature set is a challenging problem. Cohen et al. [11] showed that the problem of finding the ordering that agrees best with a learned preference function is NP-Complete. As a solution, this weighting mechanism is used as a simple heuristic to extend this maximization over the whole feature set.

In (9), $\Pr(\mathbf{p} \mid q_f)$ is the probability that the query instance q being $\mathbf{p}$-labeled, given that the value of feature f in q is q_f and w_f is the weight of the feature f, computed by (8). Finally, to obtain the weighted average, the sum of all weighted score values is divided by the sum of the weights of all used (known) features. That is, $r(q)$ is the weighted probability of the instance q has the label $\mathbf{p}$.

4.4 Time Complexity

The training time complexity of the MAD2C algorithm is $O(n^2)$, in the worst case and $O(n \log n)$ in the best case, where n is the number of training instances. Since the distribution of the data is not known, calculation of the average-case is not possible theoretically. However, in empirical evaluations, the MAD2C method converged to the convex hull in very close to linear time [35] excluding sorting time. After discretizing the numerical features, the time complexity of the RIMARC algorithm is $O(m \cdot v \cdot logv + n)$, where m is the number of features and v is the average number of categorical values per feature. That is, the training time complexity of the RIMARC is bounded by that of the MAD2C algorithm. The time complexity of the querying step is simply $O(m)$.

4.5 Interpretation of the Predictive Model of RIMARC

The RIMARC method not only provides the ranking score as a single real value for a given query instance, but also reports the model used for ranking, which can provide useful information to domain experts. For example, a high feature weight value indicates that the corresponding feature is a highly effective factor in the given domain. On the other hand, domain experts may choose to ignore features with low weights, potentially reducing the cost of record keeping.

Some of the categorical features are formed by discretizing continuous features. Assume that the effect of age is investigated on a real-world domain, such as medicine. Although such a feature can be discretized into child, youth, adult, and elderly, by some experts, the intervals should be chosen carefully since they can affect the ranking performance of the system. Further, in some experimental domains this kind of prior knowledge may not even be available. The MAD2C method used in RIMARC learns the proper intervals to maximize the AUC during the training phase.

TABLE 5
Real Life Data Sets for Evaluation

Dataset	Abbr.	Inst.	Continuous features	Categorical features
Australian	A	690	6	8
Bupa	B	345	6	0
Crx	C	653	9	6
Heart (Statlog)	H1	270	7	6
Hypothyroid	H2	3164	7	18
Mammographic Masses	M	961	1	5
Pima-Diabetes	P	768	8	0
SickEuthyroid	S1	3163	7	18
SPECTF	S2	267	44	0
Wisconsin-Breast	W	569	30	0

Properties of the data sets used in the empirical evaluations.

5 Empirical Evaluations

To support the theoretical background of the RIMARC algorithm with empirical results, it is compared with 27 different machine learning algorithms on AUC basis. To experiment with real-life domains, we selected some of the two-class data sets from the UCI machine learning repository [25]. We chose 10 data sets with two class labels. The properties of the data sets are given in Table 5.

To perform the comparisons, 27 different classification algorithms are selected from the WEKA package [28]. Since the AUC values are used for the measure of the predictive performance, only the classification methods that provide prediction probability estimates are employed in experimental results. However, the SVM algorithms do not provide these probabilities directly. Therefore, the LibSVM library is used for the SVM algorithm [9], since it provides probability estimates using the second approach proposed by Wu et al. [50]. To compare RIMARC with ranking algorithms that try to maximize the AUC directly, the SVM-Perf is also chosen, since the source code is available from the author [34].

Although RIMARC is a nonparametric method, it would be unfair not to optimize the parametric classifiers used in the evaluation. Therefore, the parameters of the SVM-based classifiers (SVM-RBF and SVM-Perf-RBF), naïve Bayes and J48 (decision tree) algorithms are optimized. For all other classifiers, their default settings in the WEKA package are used.

5.1 Predictive Performance

Researchers have reported that some of the algorithms that aim to maximize AUC directly do not obtain significantly better AUC values than the ones designed to maximize accuracy [13], [29]. Therefore, it is important to show that RIMARC can outperform accuracy-maximizing algorithms statistically significantly, as well the ranking algorithms based on classifiers.

Stratified tenfold cross validation is used to calculate AUC values for each data set. As shown in Table 6, the RIMARC method outperformed all algorithms on the average AUC metric. Further, Wilcoxon signed-rank test is used to determine whether the differences in averages are significant [48]. This statistical method is chosen, since it is a nonparametric method and does not assume normal distribution as paired t-test does. According to the Wilcoxon

TABLE 6
Predictive Performance Comparison

Algorithms/Datasets	A	B	C	H1	H2	M	P	S1	S2	W	Average
RIMARC	0.924 (0.024)	0.677 (0.085)	0.933 (0.019)	0.909 (0.040)	0.986 (0.021)	0.903 (0.031)	0.825 (0.057)	0.942 (0.036)	0.870 (0.057)	0.989 (0.011)	**0.896 (0.032)**
AdaBoostM1+	0.927 (0.028)	0.738 (0.066)	0.928 (0.029)	0.888 (0.069)	0.991 (0.011)	0.898 (0.028)	0.803 (0.044)	0.967 (0.010)	0.821 (0.072)	0.986 (0.009)	0.895 (0.037)
ClassViaRegr.+	0.923 (0.034)	0.727 (0.065)	0.923 (0.030)	0.881 (0.054)	0.990 (0.017)	0.903 (0.028)	0.829 (0.056)	0.987 (0.011)	0.778 (0.084)	0.990 (0.010)	0.893 (0.039)
Bagging+	0.920 (0.029)	0.760 (0.077)	0.914 (0.031)	0.872 (0.074)	0.981 (0.025)	0.895 (0.031)	0.822 (0.052)	0.973 (0.016)	0.803 (0.067)	0.981 (0.018)	0.892 (0.042)
Logistic+	0.916 (0.042)	0.727 (0.076)	0.920 (0.041)	0.899 (0.055)	0.971 (0.027)	0.896 (0.028)	0.829 (0.061)	0.956 (0.024)	0.819 (0.047)	0.971 (0.019)	0.890 (0.042)
MultiC.Classifier+	0.916 (0.042)	0.727 (0.076)	0.920 (0.041)	0.899 (0.055)	0.971 (0.027)	0.896 (0.028)	0.829 (0.061)	0.956 (0.024)	0.819 (0.047)	0.971 (0.019)	0.890 (0.042)
ThresholdSelector+	0.916 (0.042)	0.727 (0.076)	0.920 (0.041)	0.899 (0.055)	0.971 (0.027)	0.896 (0.028)	0.829 (0.061)	0.956 (0.024)	0.819 (0.047)	0.968 (0.020)	0.890 (0.042)
ADTree+	0.922 (0.028)	0.704 (0.074)	0.928 (0.025)	0.877 (0.072)	0.990 (0.019)	0.893 (0.034)	0.801 (0.045)	0.981 (0.018)	0.800 (0.091)	0.984 (0.014)	0.888 (0.042)
SVM-RBF-Opt++	0.914 (0.031)	0.710 (0.070)	0.923 (0.023)	0.896 (0.075)	0.983 (0.023)	0.853 (0.034)	0.821 (0.068)	0.944 (0.026)	0.835 (0.061)	0.986 (0.016)	0.887 (0.043)
AODE+	0.929 (0.029)	0.530 (0.035)	0.932 (0.033)	0.905 (0.068)	0.992 (0.011)	0.905 (0.028)	0.820 (0.050)	0.964 (0.010)	0.821 (0.079)	0.988 (0.014)	0.879 (0.036)
NaiveBayes++	0.895 (0.032)	0.648 (0.090)	0.900 (0.046)	0.900 (0.066)	0.977 (0.028)	0.897 (0.030)	0.812 (0.066)	0.920 (0.037)	0.850 (0.052)	0.980 (0.020)	0.878 (0.047)
NaiveBayes-Opt+	0.922 (0.030)	0.530 (0.035)	0.929 (0.033)	0.900 (0.071)	0.990 (0.014)	0.902 (0.027)	0.817 (0.049)	0.959 (0.013)	0.823 (0.072)	0.984 (0.014)	0.876 (0.036)
BayesNet+	0.922 (0.031)	0.530 (0.035)	0.930 (0.033)	0.900 (0.071)	0.991 (0.011)	0.900 (0.029)	0.817 (0.049)	0.960 (0.013)	0.828 (0.068)	0.984 (0.014)	0.876 (0.035)
MultiBoostAB+	0.916 (0.036)	0.685 (0.082)	0.915 (0.032)	0.866 (0.067)	0.991 (0.011)	0.897 (0.028)	0.793 (0.034)	0.960 (0.012)	0.744 (0.091)	0.982 (0.014)	0.875 (0.041)
LWL++	0.926 (0.033)	0.680 (0.083)	0.921 (0.033)	0.860 (0.074)	0.973 (0.025)	0.899 (0.033)	0.796 (0.052)	0.955 (0.018)	0.750 (0.057)	0.959 (0.029)	0.872 (0.044)
FT++	0.897 (0.049)	0.731 (0.115)	0.854 (0.067)	0.831 (0.061)	0.943 (0.049)	0.885 (0.038)	0.784 (0.083)	0.909 (0.040)	0.753 (0.101)	0.991 (0.010)	0.858 (0.061)
DecisionTable++	0.922 (0.043)	0.557 (0.044)	0.906 (0.035)	0.890 (0.078)	0.991 (0.014)	0.882 (0.030)	0.802 (0.052)	0.968 (0.013)	0.668 (0.075)	0.973 (0.018)	0.856 (0.040)
FilteredClassifier++	0.904 (0.032)	0.530 (0.035)	0.904 (0.041)	0.850 (0.068)	0.962 (0.052)	0.882 (0.039)	0.798 (0.056)	0.958 (0.015)	0.667 (0.103)	0.937 (0.040)	0.839 (0.048)
PART++	0.872 (0.067)	0.631 (0.066)	0.855 (0.035)	0.787 (0.087)	0.962 (0.031)	0.883 (0.033)	0.774 (0.066)	0.956 (0.032)	0.641 (0.094)	0.937 (0.028)	0.830 (0.054)
REPTree++	0.875 (0.042)	0.652 (0.102)	0.863 (0.034)	0.807 (0.097)	0.968 (0.022)	0.858 (0.045)	0.768 (0.061)	0.962 (0.025)	0.597 (0.116)	0.925 (0.039)	0.828 (0.058)
J48-(C4.5)Opt++	0.886 (0.034)	0.647 (0.056)	0.887 (0.043)	0.779 (0.097)	0.949 (0.053)	0.870 (0.040)	0.756 (0.039)	0.952 (0.028)	0.601 (0.128)	0.941 (0.043)	0.827 (0.056)
Att.Sel.Classifier++	0.872 (0.050)	0.602 (0.074)	0.885 (0.045)	0.809 (0.102)	0.961 (0.031)	0.869 (0.040)	0.781 (0.054)	0.920 (0.032)	0.607 (0.138)	0.942 (0.028)	0.825 (0.059)
END++	0.872 (0.052)	0.636 (0.070)	0.887 (0.046)	0.779 (0.095)	0.949 (0.041)	0.869 (0.040)	0.756 (0.036)	0.945 (0.031)	0.599 (0.114)	0.941 (0.031)	0.823 (0.056)
OrdinalC.Classifier++	0.872 (0.052)	0.636 (0.070)	0.887 (0.046)	0.779 (0.095)	0.949 (0.041)	0.869 (0.040)	0.756 (0.036)	0.945 (0.031)	0.599 (0.114)	0.941 (0.031)	0.823 (0.056)
J48 (C4.5) ++	0.872 (0.052)	0.636 (0.070)	0.887 (0.046)	0.779 (0.095)	0.949 (0.041)	0.869 (0.040)	0.756 (0.036)	0.945 (0.031)	0.599 (0.114)	0.941 (0.031)	0.823 (0.056)
VFI++	0.914 (0.039)	0.570 (0.061)	0.911 (0.037)	0.869 (0.076)	0.770 (0.078)	0.841 (0.039)	0.570 (0.071)	0.760 (0.036)	0.855 (0.046)	0.947 (0.026)	0.801 (0.051)
DecisionStump++	0.862 (0.043)	0.579 (0.058)	0.870 (0.031)	0.722 (0.083)	0.963 (0.022)	0.812 (0.038)	0.693 (0.047)	0.944 (0.011)	0.609 (0.057)	0.866 (0.045)	0.792 (0.044)
SVM-Perf-RBF-Opt++	0.660 (0.050)	0.712 (0.101)	0.659 (0.066)	0.682 (0.106)	0.976 (0.018)	0.840 (0.037)	0.710 (0.056)	0.763 (0.026)	0.687 (0.080)	0.938 (0.038)	0.763 (0.058)
SVM-Perf-RBF++	0.652 (0.048)	0.712 (0.101)	0.653 (0.067)	0.682 (0.106)	0.976 (0.018)	0.768 (0.041)	0.710 (0.056)	0.763 (0.026)	0.687 (0.080)	0.938 (0.038)	0.754 (0.058)
IBk++	0.802 (0.034)	0.629 (0.103)	0.813 (0.053)	0.743 (0.057)	0.764 (0.069)	0.795 (0.045)	0.670 (0.081)	0.759 (0.050)	0.584 (0.100)	0.948 (0.025)	0.751 (0.062)
RBFNetwork++	0.716 (0.131)	0.504 (0.079)	0.792 (0.071)	0.858 (0.068)	0.693 (0.149)	0.823 (0.043)	0.640 (0.043)	0.696 (0.073)	0.751 (0.068)	0.919 (0.107)	0.739 (0.083)
SVM-RBF++	0.654 (0.056)	0.672 (0.107)	0.646 (0.063)	0.471 (0.156)	0.965 (0.018)	0.874 (0.037)	0.648 (0.076)	0.736 (0.044)	0.411 (0.119)	0.937 (0.038)	0.701 (0.071)

The comparison of the predictive performance of RIMARC algorithm with other algorithms on the AUC metric. Ten data sets are used during evaluation. Standard deviation values are given in parenthesis. Algorithms marked with ++ are outperformed by RIMARC on average, with a statistically significant difference. Algorithms marked with + are outperformed by RIMARC on average, with no statistically significant difference (higher is better).

signed-rank test on a 95 percent confidence level (the same level will be used for other statistical tests), RIMARC statistically significantly outperforms 17 of the 27 machine learning algorithms. These algorithms include naïve Bayes, decision trees (PART, C4.5) and SVM with an RBF kernel and SVM-Perf with an RBF kernel. The RIMARC algorithm outperformed the other 10 algorithms, as well, but the differences between the averages for these algorithms are not statistically significant.

One important point should be mentioned about the SVM algorithms. As seen in Table 6, SVM has the worst predictive performance among all the classification algorithms because of the absence of parameter tuning. Therefore, we optimized the SVM-RBF and SVM-Perf-RBF by using an inner cross validation to find the optimum gamma value for the RBF kernel. The optimized versions of SVM are called SVM-RBF-Opt and SVM-Perf-RBF-Opt. Gamma value ranged from 0.01 up to 0.1 in 10 steps. As seen in Table 6, optimization boosted the performance of SVM-RBF significantly (0.701 versus 0.887). However, SVM-Perf-RBF did not gain such a significant improvement by optimization (0.754 versus 0.763). As a result, it can be claimed that the nonparametric RIMARC algorithm outperforms SVM-based methods significantly, even when they are optimized on the gamma value.

Entropy-MDLP discretization method by Fayyad and Irani [19] is used on the naïve Bayes method to improve predictive performance. Since Dougherty et al. [16] showed that using discretization improves naïve Bayes predictive performance, an improvement was expected. However, as Table 6 shows, the discretization did not improve the performance of naïve Bayes at all. The C parameter is optimized by using inner cross validation for J48 algorithm (decision tree). The C-value is ranged from 0.1 up to 0.5 in 10 steps.

The classifier with the highest AUC after RIMARC was the Adaboost method. As an ensembling algorithm, Adaboost uses a base classifier, which is DecisionStump by default in the WEKA package.

The classification algorithms such as Logistic (multinomial logistic regression model) and ClassViaReg (classification via regression) achieve high AUC values. As mentioned above, these models are highly used in the domain of medicine, and in this work their predictive performance is validated.

Another point that deserves mentioning is that the average standard deviation among the individual AUC values of the RIMARC algorithm is the smallest among all the algorithms we tested.

Similar to the naïve Bayes classifier, the RIMARC algorithm assumes that the features are independent of each other. Holte [32] has pointed out that most of the data sets in the UCI repository are such that, for classification, their attributes can be considered independently of each other, which explains the success of the RIMARC algorithm on these data sets. Similar observation is also made by Güvenir and Şirin [27].

5.2 Running Time

The RIMARC method is designed to be simple, effective, and fast. It computes the scores for each value for a categorical

TABLE 7
Running Time Performance Comparison

Algorithms/Datasets	A	B	C	H1	H2	M	P	S1	S2	W	**Average**
VFI--	16	5	17	9	104	12	14	111	23	31	34
DecisionStump--	24	9	23	11	113	17	30	116	43	105	49
NaiveBayes--	30	11	29	18	164	30	37	154	61	102	64
AODE--	53	16	52	27	352	40	56	350	113	280	134
BayesNet--	49	17	52	28	425	54	60	393	81	244	140
REPTree--	77	33	80	33	395	105	136	569	124	185	174
FilteredClassifier--	119	16	107	48	462	86	116	858	155	265	223
J48 (C4.5)-	160	75	153	81	635	147	181	1283	256	378	335
Att.Sel.Classifier-	173	31	159	99	761	201	162	867	419	496	337
RIMARC	106	37	94	49	1131	128	164	1117	191	470	**349**
OrdinalC.Classifier+	162	79	150	83	681	177	197	1380	255	378	354
END+	236	119	218	126	880	219	254	1563	333	460	441
RBFNetwork++	404	136	348	148	1057	367	280	1149	712	620	522
PART++	416	102	511	193	865	230	248	2023	684	473	574
AdaBoostM1++	302	132	296	156	1584	302	394	1579	475	1135	635
MultiBoostAB++	318	133	297	164	1614	317	388	1622	477	1140	647
MultiC.Classifier++	991	79	1175	125	3698	239	226	3656	473	963	1163
Logistic++	1014	74	1215	124	3720	261	257	3667	472	935	1174
ADTree++	689	276	645	469	3149	579	973	3536	1562	2717	1459
IBk+	172	33	169	35	7365	233	153	7465	124	222	1597
Bagging++	740	295	727	295	4484	636	961	6904	1070	1718	1783
ThresholdSelector++	1695	137	2032	227	6442	429	387	6346	1118	2028	2084
DecisionTable++	1582	156	1699	434	10824	411	635	11498	1183	2526	3095
Class.ViaRegr. ++	5340	1355	5573	1143	9943	3912	3593	15598	2218	2662	5134
FT++	4705	847	4879	927	14834	2856	2230	30959	1796	2324	6636
LWL++	2094	376	1940	412	52652	2360	2652	52887	1414	6919	12370

The comparison of the average running time performance of RIMARC with other algorithms (in ms). Ten data sets are used during evaluation. Algorithms marked with ++ are outperformed by RIMARC on running time basis with a statistically significant difference. Algorithms marked with -- outperformed RIMARC on running time basis with a statistically significant difference. Algorithms marked with + are outperformed by RIMARC on average, and the ones marked with - outperform RIMARC on average, but with no significant difference. (Lower is better).

feature in close to linear time. MAD2C requires more time since it uses sorting. Although the time complexity of the RIMARC algorithm is shown to be low, empirical experiments were conducted to support this claim.

The overall running times of the training phase of 25 different algorithms on the data sets are compared with that of RIMARC. The training times of all algorithms are measured using Java Virtual Machine's CPU time and 100 results are averaged.

Since the outsourced libraries were used for the SVM-RBF and SVM-Perf-RBF algorithms, they are not included in the running time experiments. The results of the overall running time for the other algorithms are shown in Table 7.

The RIMARC algorithm, on the basis of running times, significantly outperformed 13 of the algorithms according to the Wilcoxon signed-rank test [49]. These methods outperformed by RIMARC are indicated by the ++ symbol in Table 7. Seven algorithms significantly outperformed RIMARC. These algorithms are shown with a $--$ symbol. The differences between the other five methods in the table and RIMARC are not significant. Note that all of the algorithms that significantly outperformed the RIMARC algorithm in terms of the running time are outperformed by RIMARC in terms of the AUC metric.

6 Related Work

The problem of learning a real-valued function that induces a ranking or ordering over an instance space has gained importance in machine learning literature. Information retrieval, credit-risk screening or estimation of risks associated with a surgery are some examples of the application domains. In this paper, we consider the ranking problem with binary classification data. It is known as the *bipartite ranking problem*, which refers to the problem of learning a ranking function from a training set of examples with binary labels [2], [10], [26]. Agarwal and Roth [2] studied the learnability of bipartite ranking functions and showed that learning linear ranking functions over Boolean domains is NP-hard. In the bipartite ranking problem, given a training set of instances with labels either positive or negative, one wants to learn a real-valued ranking function that can be used for an unseen case to associate a measure of being close to positive (or negative) class. For example, in a medical domain, a surgeon may be concerned with estimating the risk of a patient who is planned to undergo a serious operation. A successful ranking (or scoring) function is expected to return a high value if the operation carries high risks for that patient. Specific ranking functions have been developed for particular domains, such as information retrieval [18], [47], finance [6], medicine [12], [14], [44], fraud detection [20], and insurance [17]. Some of these methods are dependent on statistical models while some are based on machine learning algorithms. A binary classification algorithm that returns a confidence factor associated with the class label can be used for bipartite ranking, where the confidence factor associated with a positive label (or the complement associated with a negative label) can be taken as the value of the ranking function.

The area under the receiver operating characteristic curve (AUC) is a widely accepted performance measure for evaluating the quality of a ranking function [5], [22]. It has been shown that the AUC represents the probability that a randomly chosen positive instance is correctly assigned a higher rank value than a randomly selected negative instance. Further, this probability of correct ranking is equal to the value estimated by the nonparametric Wilcoxon statistic [30]. Also, AUC has important features such as insensitivity to class distribution and cost distributions [5], [22], [33]. Agarwal et al. [1] showed what kind of classification algorithms can be used for ranking problems and proved theorems about generalization properties of AUC.

Some approximation methods aiming at maximizing the global AUC value directly have been proposed by researchers [31], [39], [51]. For example, Ataman et al. [3] proposed a ranking algorithm by maximizing AUC with linear programming. Brefeld and Scheffer [7] presented an AUC maximizing support vector machine. Rakotomamonjy [43] proposed a quadratic programming-based algorithm for AUC maximization and showed that under certain conditions 2-norm soft margin support vector machines can also maximize AUC. Toh et al. [46] designed an algorithm to optimize the ROC performance directly according to the fusion classifier. Ferri et al. [23] proposed a method to locally optimize AUC in decision tree learning, and Cortes and Mohri [13] proposed boosted decision stumps. To maximize AUC in rule learning, several algorithms have been proposed [4], [21], [40]. A nonparametric linear classifier based on the local maximization of AUC was presented by Marrocco et al. [38]. A ROC-based genetic learning algorithm has been proposed by Sebag et al. [44]. Marrocco et al. [37] used linear

combinations of dichotomizers for the same purpose. Freund et al. [26] gave a boosting algorithm combining multiple rankings. Cortes and Mohri [13] showed that this approach also aims to maximize AUC. A method by Tax et al. [45] that weighs features linearly by optimizing AUC has been applied to the detection of interstitial lung disease. Ataman et al. [3] advocated an AUC-maximizing algorithm with linear programming. Joachims [34] proposed a binary classification algorithm by using SVM that can maximize AUC. Ling and Zhang [36] compared AUC-based tree-augmented naïve Bayes (TAN) and error-based TAN algorithms; the AUC-based algorithms are shown to produce more accurate rankings. More recently, Calders and Jaroszewicz [8] suggested a polynomial approximation of AUC to optimize it efficiently. Linear combinations of classifiers are also used to maximize AUC in biometric scores fusion [46]. Han and Zhao [29] proposed a linear classifier based on active learning that maximizes AUC.

7 Conclusions and Future Work

In this paper, we presented a supervised algorithm for learning a ranking function, called RIMARC.

We have shown that for a categorical feature, there is only one ordering that gives the maximum AUC. Then, we showed the necessary and sufficient condition that a ranking function for a single categorical feature has to satisfy to achieve this ordering. As a result, we proposed a ranking function that achieves the maximum possible AUC value on a single categorical feature. This ranking function is based on the probability of $\mathbf{p}$ class for each value of that feature. The MAD2C algorithm used by RIMARC discretizes continuous features in a way that yields the maximum AUC, as well. The RIMARC algorithm used AUC values of features as their weights in computing the ranking function. With this simple heuristic, we computed the weighted average of all feature value scores to achieve maximum AUC over the whole feature set. Since the RIMARC algorithm uses all available feature values and ignores the missing ones, it is robust to missing feature values.

We presented the characteristics of the ranking function learned by the RIMARC algorithms and how it can be interpreted. The ranking function is in a human readable form that can be easily interpreted by domain experts. The feature weights learned help the experts to determine how they affect the ranking.

We compared RIMARC with 27 different algorithms. According to our empirical evaluations, RIMARC significantly outperformed 17 algorithms on an AUC basis and 13 algorithms on a time basis. It also outperformed all algorithms on the average AUC and 16 of them on an average running time basis.

It is also worth noting that the RIMARC algorithm is a non-parametric machine learning algorithm. As such, it does not have any parameters that need to be tuned to achieve high performance on a given data set; hence, it can be used by domain experts who are not experienced in tuning machine learning algorithms.

To improve the performance of RIMARC, instead of using the weighted average, other approaches can be investigated. Another possible direction for future work would be to experiment with methods that ensemble RIMARC with other ranking algorithms.

References

[1] S. Agarwal, T. Graepel, R. Herbrich, S. Har-Peled, and D. Roth, "Generalization Bounds for the Area under the ROC Curve," *J. Machine Learning Research,* vol. 6, pp. 393-425, 2005.

[2] S. Agarwal and D. Roth, "Learnability of Bipartite Ranking Functions," *Proc. 18th Ann. Conf. Learning Theory,* 2005.

[3] K. Ataman, W.N. Street, and Y. Zhang, "Learning to Rank by Maximizing AUC with Linear Programming," *Proc. IEEE Int'l Joint Conf. Neural Networks (IJCNN),* pp. 123-129, 2006.

[4] H. Boström, "Maximizing the Area under the ROC Curve Using Incremental Reduced Error Pruning," *Proc. Int'l Conf. Machine Learning Workshop (ICML '05),* 2005.

[5] A.P. Bradley, "The Use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms," *Pattern Recognition,* vol. 30, no. 7, pp. 1145-1159, 1997.

[6] B.O. Bradley and M.S. Taqqu, "Handbook of Heavy-Tailed Distributions in Finance," *Financial Risk and Heavy Tails,* S.T. Rachev, ed., pp. 35-103, Elsevier, 2003.

[7] U. Brefeld and T. Scheffer, "AUC Maximizing Support Vector Learning," *Proc. ICML Workshop ROC Analysis in Machine Learning,* 2005.

[8] T. Calders and S. Jaroszewicz, "Efficient AUC Optimization for Classification," *Proc. 11th European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD '07),* pp. 42-53, 2007.

[9] C.C. Chang and C.C. Lin, "LIBSVM: A Library for Support Vector Machines," http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.

[10] S. Clemençon, G. Lugosi, and N. Vayatis, "Ranking and Scoring Using Empirical Risk Minimization," *Proc. 18th Ann. Conf. Learning Theory (COLT '05),* pp. 1-15, 2005.

[11] W.W. Cohen, R.E. Schapire, and Y. Singer, "Learning to Order Things," *J. Artificial Intelligence Research,* vol. 10, pp. 243-270, 1998.

[12] R.M. Conroy, K. Pyörälä, and A.P. Fitzgerald, "Estimation of Ten-Year Risk of Fatal Cardiovascular Disease in Europe: The SCORE Project," *European Heart J.,* vol. 11, pp. 987-1003, 2003.

[13] C. Cortes and M. Mohri, "AUC Optimization versus Error Rate Minimization," *Proc. Conf. Neural Information Processing Systems (NIPS '03),* vol. 16, pp. 313-320, 2003.

[14] R.B. D'Agostino, S.V. Ramachandran, and J. Pencina, "General Cardiovascular Risk Profile for Use in Primary Care: The Framingham Heart Study," *Circulation,* vol. 17, pp. 743-753, 2008.

[15] P. Domingos, "MetaCost: A General Method for Making Classifiers Cost-Sensitive," *Proc. Fifth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 155-164, 1999.

[16] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," *Proc. 12th Int'l Conf. Machine Learning,* pp. 194-202, 1995.

[17] K. Dowd and D. Blake, "After VaR: The Theory, Estimation, and Insurance Applications of Quantile-Based Risk Measures," *The J. Risk and Insurance,* vol. 73, no. 2, pp. 193-229, 2006.

[18] W. Fan, M.D. Gordon, and P. Pathak, "Discovery of Context-Specific Ranking Functions for Effective Information Retrieval Using Genetic Programming," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 4, pp. 523-27, Apr. 2004.

[19] U. Fayyad and K. Irani, "On the Handling of Continuous-Valued Attributes in Decision Tree Generation," *Machine Learning,* vol. 8, pp. 87-102, 1992.

[20] T. Fawcett and F. Provost, "Adaptive Fraud Detection," *Data Mining and Knowledge Discovery,* vol. 1, pp. 291-316, 1997.

[21] T. Fawcett, "Using Rule Sets to Maximize ROC Performance," *Proc. IEEE Int'l Conf. Data Mining (ICDM '01),* pp. 131-138, 2001.

[22] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters,* vol. 27, pp. 861-874, 2006.

[23] C. Ferri, P. Flach, and J. Hernandez, "Learning Decision Trees Using the Area under the ROC Curve," *Proc. 19th Int'l Conf. Machine Learning (ICML '02),* pp. 139-146, 2002.

[24] P. Flach and S. Wu, "Repairing Concavities in ROC Curves," *Proc. UK Workshop Computational Intelligence,* pp. 38-44, 2003.

[25] A. Frank and A. Asuncion, "UCI Machine Learning Repository," School of Information and Computer Science, Univ. of California, http://archive.ics.uci.edu/ml, 2010.

[26] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences," *J. Machine Learning Research,* vol. 4, pp. 933-969, 2003.

[27] H.A. Güvenir and İ. Şirin, "Classification by Feature Partitioning," *Machine Learning,* vol. 23, no. 1, pp. 47-67, 1996.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations,* vol. 11, no. 1, pp. 10-18, 2009.
[29] G. Han and C. Zhao, "AUC Maximization Linear Classifier Based on Active Learning and Its Application," *Neurocomputing,* vol. 73, nos. 7-9, pp. 1272-1280, 2010.
[30] J.A. Hanley and B.J. McNeil, "The Meaning and use of the Area under a Receiver Operating Characteristic (ROC) Curve," *Radiology,* vol. 143, pp. 29-36, 1982.
[31] A. Herschtal and B. Raskutti, "Optimising the Area under the ROC Curve Using Gradient Descent," *Proc. Int'l Conf. Machine Learning,* pp. 49-56, 2004.
[32] R.C. Holte, "Very Simple Classification Rules Perform Well on Most Commonly Used Data Sets," *Machine Learning,* vol. 11, pp. 63-91, 1993.
[33] J. Huang and C.X. Ling, "Using AUC and Accuracy in Evaluating Learning Algorithms," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 3, pp. 299-310, Mar. 2005.
[34] T. Joachims, "A Support Vector Method for Multivariate Performance Measures, " *Proc. Int'l Conf. Machine Learning (ICML),* 2005.
[35] M. Kurtcephe and H.A. Güvenir, "A Discretization Method Based on Maximizing the Area under ROC Curve," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 27, no. 1, article 1350002, 2013.
[36] C.L. Ling and H. Zhang, "Toward Bayesian Classifiers with Accurate Probabilities," *Proc. Sixth Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining,* pp. 123-134, 2002.
[37] C. Marrocco, M. Molinara, and F. Tortorella, "Exploiting AUC for Optimal Linear Combinations of Dichotomizers," *Pattern Recognition Letters,* vol. 27, no. 8, pp. 900-907, 2006.
[38] C. Marrocco, R.P.W. Duin, and F. Tortorella, "Maximizing the Area under the ROC Curve by Pairwise Feature Combination," *Pattern Recognition,* vol. 41, pp. 1961-1974, 2008.
[39] M.C. Mozer, R. Dodier, M.D. Colagrosso, C. Guerra-Salcedo, and R. Wolniewicz, "Prodding the ROC Curve: Constrained Optimization of Classifier Performance," *Proc. Conf. Advances in Neural Information Processing Systems,* vol. 14, pp. 1409-1415, 2002.
[40] R. Prati and P. Flach, "Roccer: A ROC Convex Hull Rule Learning Algorithm," *Proc. ECML/PKDD Workshop Advances in Inductive Rule Learning,* pp. 144-153, 2004.
[41] F. Provost and T. Fawcett, "Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions," *Proc. Third Int'l Conf. Knowledge Discovery and Data Mining,* pp. 43-48, 1997.
[42] F. Provost, T. Fawcett, and R. Kohavi, "The Case against Accuracy Estimation for Comparing Induction Algorithms," *Proc. 15th Int'l Conf. Machine Learning,* pp. 445-453, 1998.
[43] A. Rakotomamonjy, "Optimizing Area under ROC Curve with SVMS," *Proc. Workshop ROC Analysis in Artificial Intelligence,* pp. 71-80, 2004.
[44] M. Sebag, J. Aze, and N. Lucas, "ROC-Based Evolutionary Learning: Application to Medical Data Mining," *Artificial Evolution,* vol. 2936, pp. 384-396, 2004.
[45] D.J.M. Tax, R.P.W. Duin, and Y. Arzhaeva, "Linear Model Combining by Optimizing the Area under the ROC Curve," *Proc. IEEE 18th Int'l Conf. Pattern Recognition,* pp. 119-122, 2006.
[46] K.A. Toh, J. Kim, and S. Lee, "Maximizing Area under ROC Curve for Biometric Scores Fusion," *Pattern Recognition,* vol. 41, pp. 3373-3392, 2008.
[47] F. Wang and X. Chang, "Cost-Sensitive Support Vector Ranking for Information Retrieval," *J. Convergence Information Technology,* vol. 5, no. 10, pp. 109-116, 2010.
[48] M. Wasikowski and X. Chen, "Combating the Small Sample Class Imbalance Problem Using Feature Selection," *IEEE Trans. Knowledge Discovery and Data Eng.,* vol. 22, no. 10, pp. 1388-1400, Oct. 2010.
[49] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics,* vol. 1, pp. 80-83, 1945.
[50] T.-F. Wu, C.-J. Lin, and W.C. Wen, "Probability Estimates for Multi-Class Classification by Pairwise Coupling," *J. Machine Learning Research,* vol. 5, pp. 975-1005, 2004.
[51] L. Yan, R. Dodier, M.C. Mozer, and R. Wolniewicz, "Optimizing Classifier Performance via the Wilcoxon-Mann-Whitney Statistics," *Proc. 20th Int'l Conf. Machine Learning,* pp. 848-855, 2003.

H. Altay Güvenir received the BS and MS degrees in electronics and communications engineering from Istanbul Technical University, in 1979 and 1981, respectively, and the PhD degree in computer engineering and science from Case Western Reserve University in 1987. He joined the Department of Computer Engineering at Bilkent University in 1987. He has been a professor and serving as the chairman of the Department of Computer Engineering since 2001. His research interests include artificial intelligence, machine learning, data mining, and intelligent data analysis. He is a member of the IEEE and the ACM.

Murat Kurtcephe received the first MSc degree in computer science from Bilkent University, Ankara, Turkey, where he worked on machine learning and data mining, especially on discretization and risk estimation methods with Prof. H. Altay Güvenir. He received the second MSc degree from the Computer Science Department at Case Western University, where he worked with Professor Meral Ozsoyoglu on query visualization and pedigree data querying. Currently, he is working at NorthPoint Solutions, New York, focusing on regulatory compliance reporting applications.