Handling Class Imbalance with POISE: pAUC Optimization in Supervised Experiments

Nikitha Rao
t-nirao@microsoft.com
Microsoft Research, India

Sreangsu Acharyya
srach@microsoft.com
Microsoft Research, India

ABSTRACT

Recognizing the well known deficiencies of classification accuracy as a quality metric in class imbalanced scenarios, we reaffirm the use of partial AUC (pAUC), which is an improvement over the related metric of AUC. Optimizing pAUC is formulated as a two person zero-sum game between (i) an adversary that selects a fixed fraction of negative examples and (ii) a learner that needs to assign higher scores to the positive examples, no matter the choice of the adversary. The optimal scoring function is obtained as an equilibrium of this game. This optimization is combined with an efficient, task specific vector embedding that captures the geometry induced by decision trees, thereby extending the method to datasets that are not linearly separable. We evaluate our proposed solution by comparing its performance against state of the art alternatives (such as LambdaMART, RankSVM) as well as popular alternatives such as SMOTE and make note of the superior performance obtained.

KEYWORDS

partial auc; learning to rank; bipartitie ranking; class imbalance

ACM Reference Format:
Nikitha Rao and Sreangsu Acharyya. 2020. Handling Class Imbalance with POISE: pAUC Optimization in Supervised Experiments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Despite classification accuracy being a common measure of model performance, it can be misleading for imbalanced datasets. In such cases, a classifier can achieve high accuracy by predicting the majority class — making the ability to distinguish classes unnecessary for achieving good performance — defeating the purpose of the classifier and the performance metric. This has led to the use of other metrics like precision-recall, F1 score [4].

For classifiers obtained by thresholding a scoring function, the receiver operating characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at different thresholds, has been used in various fields where evaluation of discrimination performance is of importance. TPR is the proportion of positive labels that are correctly identified and FPR is the proportion of negative labels incorrectly identified. The area under the ROC curve (AUC), computed as the TPR averaged over the full range of possible FPRs, is a popular performance metric that is used in binary classification and bipartite ranking problems [1, 27, 38]. AUC is often preferred over Accuracy, particularly when misclassification costs are high or the classes are imbalanced [5, 11, 28].

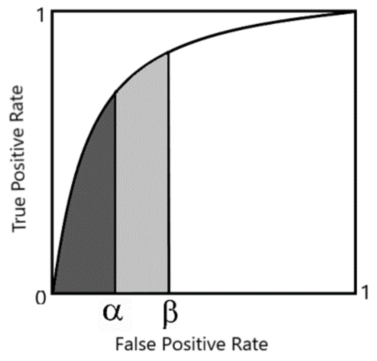

Figure 1: Partial AUC in FPR range $[\alpha, \beta]$

Most applications, however, cannot operate in the complete FPR range and need to achieve high TPR at low FPR range. AUC being an average over the entire FPR range allows a classifier to recover performance lost in low FPR regimes by performing well in high FPR regime, although, it will not be operating under such high FPR regimes in practice.

This deficiency of AUC can be mitigated by averaging over application relevant FPR ranges, thereby obtaining *partial* area under the ROC curve (pAUC) between two specified false positive rates (as shown in Figure 1). This is especially useful in medical diagnosis [18, 21, 42] where a high TPR is required at low FPR. In ranking problems [1, 38] where the accuracy at the top is critical [37] or in biometric screening where false positives are unacceptable, the leftmost part of the ROC curve is of utmost importance corresponding to maximizing the partial AUC in the false positive range $[0, \alpha]$ [1, 34, 39]. Note that the partial AUC is equal to the AUC when $\alpha = 1$.

In this work, we present a partial AUC maximization algorithm to handle class imbalance that uses a game theory formulation for the cost function. Minimizing pAUC loss is formulated as a two person zero-sum game between (i) an adversary that selects a fixed fraction of negative examples and (ii) a learner that needs to assign higher scores to the positive examples, no matter the choice of the adversary. The optimal scoring function is obtained as an equilibrium of this game. First, we recall a reduction of a linear scoring function based AUC maximization to a linear classification problem. The restriction to linear separators, an unfortunate limitation of this reduction, is removed by a novel use of learned decision tree paths. The efficient, task specific vector embeddings leverage the geometry induced by decision trees to capture any inherent non-linearity in the data, thereby allowing us to extend the proposed method to datasets that are not linearly separable. Note that the non-linearities exploited for class separation are not predetermined but learned from data. Finally, we evaluate our proposed method on a number of publicly available benchmark datasets and compare our performance with other state of the art models. We demonstrate empirically that our approach is both fast and scalable. We also show that it not only performs well for learning to rank tasks but can also generalize to other problems where accuracy at the top is of utmost importance.

Our contributions can be summarised as follows:

(1) We present a partial AUC maximization algorithm to handle class imbalance that uses a game theory formulation for the cost function (described in Section 5).
(2) We introduce the novel use of learned decision tree paths as vector embeddings to capture any inherent non-linearity in data (described in Section 4).
(3) We demonstrate empirically that our approach is both fast and scalable. We also show that our method not only performs well for learning to rank tasks but can also generalize to other problem spaces where partial AUC or accuracy at the top is crucial.

The remainder of this paper is organized as follows: We present an overview of the literature in Section 2. In Section 3 we review the AUC maximization formulation. In Section 4, we show introduce vector embeddings based on learned decision tree paths followed by the proposed method for pAUC maximization in Section 5. We present our experimental setup and results in Section 6 and Section 7 respectively. Finally, we present a discussion in Section 8 and conclude the paper in Section 9.

2 BACKGROUND AND RELATED WORK

Learning to rank methods have been used to solve different types of ranking problems in a variety of domains including information retrieval [31] (document retrieval, collaborative filtering, question answering and online advertising) [12, 30, 49, 57], recommendation systems [16, 40, 44], computational biology [48] and software engineering [36, 53]. Based on the input representation and loss function, learning to rank algorithms can be categorized into three types: pointwise [15, 26, 50], pairwise [6, 14, 23] and listwise [7, 33, 52].

In the pointwise approach, each query-document pair in the training data has a numerical score associated with it. Here, the learning to rank problem is approximated to a regression problem where the goal is to predict the relevance score for a given a query-document pair. Some examples include: MART, CRR, McRank etc. In contrast, the pairwise approach approximates the learning to rank to a binary classification problem where given a pair of documents, the goal is to identify the document that is of higher relevance. Some examples include RankSVM, RankNet, RankBoost, LambdaRank and so on. On the other hand, listwise approach aims to directly optimize the ranking metrics such as Normalized Discounted Cumulative Gain [20] or Expected Reciprocal Rank [8] for a set of ranked documents averaged over all queries. Optimizing the ranking metrics is not easy as several of the ranking measures are discontinuous functions. However, this problem has been intensively studied and there are a range of methods available in the literature for direct optimization of metrics [7, 33, 52, 58].

Algorithms falling into the pairwise category are able to handle class imbalance in ranking as they perform a comparison between every document pair to find the relative relevance, eliminating any imbalance that may be present. Though several algorithms have been developed for optimizing AUC in ranking [3, 14], they fail to perform well in scenarios where accuracy at the top is of importance. This is because when optimizing for AUC, the algorithms make up for the performance lost at low FPR by performing well at high FPRs. However, in practice we cannot afford to operate at such high FPR and therefore need to optimize for pAUC instead.

2.1 pAUC maximization

Despite AUC being commonly used to measure model performance, several applications such as bioinformatics [29, 48], medical diagnosis [21, 42] and computer vision [41] are now using pAUC as their choice of evaluation metric.

The problem of optimizing pAUC in a given FPR range has been primarily studied in the bioinformatics literature. For example, the greedy heuristics method [46] maximizes pAUC by means of linear combination of classifiers. However, several of the algorithms developed are based on heuristics that are specific to a given application and fail to generalise.

Several modifications have been made to the standard SVM algorithm [54] to maximize both AUC and pAUC in a given FPR range. While the structural SVM algorithm by Joachims et al. [22] optimizes multivariate non-linear performance measures to maximize AUC. Other variations of the SVM like the asymmetric SVM algorithm uses a variant of one-class SVM to optimize for pAUC in $[0, \alpha]$ by fine-grained parameter tuning. SVM_{pAUC} [34] extends the structural SVM framework of Joachims [22] to design convex surrogates that optimizes for pAUC using a cutting plane solver. Narasimhan et al. [35] further extends the SVM_{pAUC} to maximize the pAUC in the FPR range $[\alpha, \beta]$ by using a tighter approximation that directly optimizes the non-convex surrogate using well known non-convex optimization techniques based on difference-of-convex programming. Additionally, there have been several boosting based algorithms that optimizes AUC and pAUC. While algorithms like AdaBoost, RankBoost aim to maximize the entire area under the ROC curve, pAUCBooost and pU-AUCBoost optimize for pAUC in general false positive ranges of the form $[\alpha, \beta]$.

Our work is different from the existing work primarily in two aspects: a) This is the first work to use a game theory formulation for optimizing pAUC. We formulate the pAUC loss minimization as a zero-sum game between (i) an adversary that selects a fixed fraction of negative examples and (ii) a scoring function that needs to assign positive examples higher scores, irrespective of the choice of the adversary. An equilibrium of this game results in the optimal scoring function. b) We introduce a novel task specific vector embedding technique that captures the geometry induced by decision trees. The learned decision tree paths exploit any non-linearities that may be present in the data thereby extending the proposed method to datasets that are not linearly separable.

3 PAIRWISE FORMULATION FOR AUC

In this section we recall a standard reduction of the AUC maximization problem into a linear classification problem via the use of pairwise differences between the positive and negative examples [32]. Let $\mathcal{X}_+ = \{\mathbf{x}_1^+, \mathbf{x}_2^+, \ldots, \mathbf{x}_{N_+}^+\}$ be a set of N^+ positive examples and $\mathcal{X}_- = \{\mathbf{x}_1^-, \mathbf{x}_2^-, \ldots, \mathbf{x}_{N_-}^-\}$ be a negative example set of size N^- where $\mathbf{x} \in \mathbb{R}^n$ such that $N_+ \ll N_-$ and $y \in \{-1, 1\}$ is the class label. We use the scoring function $Q : \mathbb{R}^n \ni \mathbf{x} \mapsto Q(\mathbf{x}) = \mathbf{w}^\top \cdot \mathbf{x} \in \mathbb{R}$. The goal is to find a weight vector, $\mathbf{w}$, such that, $Q(\mathbf{x}_i) > Q(\mathbf{x}_j)$ where $y_i = +1,\ y_j = -1$. This implies $\mathbf{w}^\top \cdot \mathbf{x}_i > \mathbf{w}^\top \cdot \mathbf{x}_j$, or equivalently, $\mathbf{w}^\top \cdot (\mathbf{x}_i - \mathbf{x}_j) > 0$.

With subscript i indexing over the set $\mathcal{X}_+$ and subscript j indexing the set $\mathcal{X}_-$, let us define $\mathbf{z}_{ij} = \mathbf{x}_i - \mathbf{x}_j,\ y_{ij} = +1$ and $\mathbf{z}_{ji} = \mathbf{x}_j - \mathbf{x}_i,\ y_{ji} = -1$. The identities $\mathbf{w}^\top \cdot \mathbf{z}_{ij} > 0$ and $\mathbf{w}^\top \cdot \mathbf{z}_{ji} < 0$, confirms that this is a reduction to a linear classification problem, where vector $\mathbf{w}$ separates the classes $\bigcup \mathbf{z}_{ij}$ from $\bigcup \mathbf{z}_{ji}$.

Given sets $\mathcal{X}_+$ and $\mathcal{X}_-$, the AUC induced by the linear scoring function $Q(\mathbf{x}) = \mathbf{w}^\top \cdot \mathbf{x}$ can be computed as,

$$AUC(w) = \frac{1}{N_+ N_-} \sum_{i \in \mathcal{X}_+} \sum_{j \in \mathcal{X}_-} \mathbb{1}(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) > 0), \tag{1}$$

where $\mathbb{1}(\cdot)$ is the indicator function. Maximizing AUC over linear scoring functions is then equivalent to finding $\mathbf{w}$ that minimizes Equation 1, stated formally as

$$\min_{\mathbf{w}} \frac{1}{N_+ N_-} \sum_{i \in \mathcal{X}_+} \sum_{j \in \mathcal{X}_-} \mathbb{1}(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) > 0). \tag{2}$$

A standard practice in machine learning is to replace the intractable sum of indicator functions in an objective function by an upper bound obtained by replacing indicator functions by convex upper-bounding surrogate. Replacing the indicator functions in (1) by the Hinge function $h(x) = \max(0.x + 1)$ and adding a L_2^2 regularizer we obtain the surrogate objective, which readers will recognize as the objective function of an SVM (support vector machine) [54].

$$\min_{\mathbf{w}} \frac{1}{N_+ N_-} \sum_{i \in \mathcal{X}_+} \sum_{j \in \mathcal{X}_-} h(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+)) + c\|\mathbf{w}\|^2. \tag{3}$$

4 USING NON-LINEAR FEATURES

A limitation of the reduction discussed in Section 3 is that it applies to linear classifiers only. Datasets where classes are not linearly separable are common. This makes non-linearity essential to class separators if they are to be widely applicable. In this section we reconcile these two conflicting aspects of our approach.

Kernel methods are frequently adopted when linear methods are inadequate [19]. They typically use a *predetermined* kernel function $K(\cdot\,,\cdot)$ that maps a point $\mathbf{x}$, *implicitly* and non-linearly, to a point $\phi(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x})$ in the feature space where the classes are better separated linearly. In contrast, we use an *explicit non-linearity* learned from data to increase the linear separability of the classes in the feature space. In order to do so, we rely on two key properties, (i) a well known empirical observation that decision trees are one of the state of the art classifiers that can effectively separate classes that are not linearly separable and (ii) an interpretation of such decision trees as a vector embedding that facilitates linear separation.

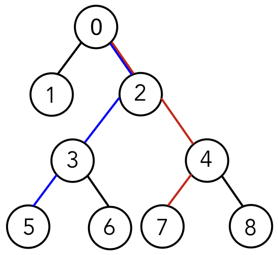

Figure 2: Feature generation using decision tree path

The remainder of the section is presented in two parts. The first describes how, given a decision tree, we obtain the non-linear vector embedding of items so that they are more amenable to linear separation. The second describes how we obtain the decision tree.

4.1 Vector embedding using decision tree

The root node of a decision tree represents the entire input domain that is partitioned into two by a node-specific predicate. Each such partition produced is represented by a child node that is partitioned, similarly and recursively, as many times as necessary to promote purity of label proportions in the leaf partitions. Leaf nodes of the tree are assigned the majority label of the corresponding partition. To predict the label of a data-point $\mathbf{x}$, one identifies the leaf node reached by traversing the tree by selecting the left or the right child as indicated by the node predicate evaluated on $\mathbf{x}$. The predicted label is the label of the leaf node reached. Formally, this can be expressed as

$$\sum_{p \in Paths[I]} y_i \mathbb{1}_{p_i(\mathbf{x})} \tag{4}$$

where $\mathbb{1}_{p_i(\mathbf{x})}$ is the indicator function that takes a value of 1 if $\mathbf{x}$ reaches the i^{th} leaf by traversing the tree (using the unique path p_i).

Using the formal similarity of expression (4) with the expression of a standard kernel classifier

$$\sum_i y_i \lambda_i K(\mathbf{x}_i, \mathbf{x})$$

we interpret the function $\mathbb{1}_{p_i(\mathbf{x})}$ as an approximate weighted kernel function $\lambda_i K(\mathbf{x}_i, \mathbf{x})$ defined by an unknown $\mathbf{x}_i$. Informally, a kernel function $K(\mathbf{x}_i, \mathbf{x})$ quantifies the similarity between $\mathbf{x}$ and $\mathbf{x}_i$ much as $\mathbb{1}_{p_i(\mathbf{x})}$ measures the similarity of $\mathbf{x}$ with path p_i, also represented by an embedded vector.

Given a decision tree with N nodes we map the data points to a $N-1$ dimensional space where each dimension corresponds to one of the $N-1$ edges of the tree. Dimensions that correspond to edges lying on p_i take (an absolute) value of 1.0, others are 0. These are scaled by 1.0 or -1.0 depending on the class label of the leaf node. Given that only one edge emerging from a node may be followed, there is no loss in generality in considering a frugal embedding of dimension of $(N-1)/2$ obtained by choosing one of the out edges of all non-leaf nodes.

Figure 2 is an example of a decision tree. Data sample $\mathbf{x}_1$ follows the red path while $\mathbf{x}_2$ follows the blue path. The decision tree path taken by $\mathbf{x}_1$ is $\mathbf{t}_1 = [01010010]$ and x_2 is $\mathbf{t}_2 = [01101000]$. Therefore, $K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{t}_1 \cdot \mathbf{t}_2 = 1$.

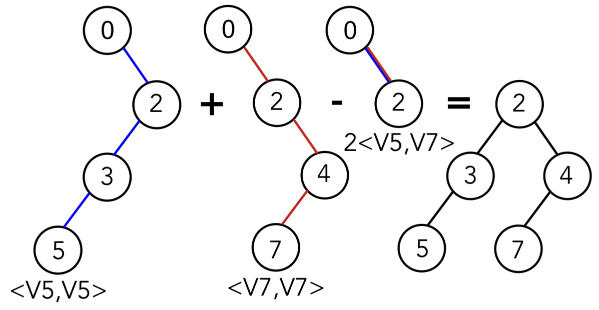

Figure 3: Common prefix as inner product

The non-linear mapping from data space to feature space is obtained by associating each leaf node of the decision tree with a vector such that all data points that reach the leaf node are mapped into the leaf specific vector. The embedding is chosen such that points with the same class label map to vectors close to each other if they reach leaves that are close in shortest path distance in the tree. On the other hand, since sibling leaves have opposite class labels they are kept apart by embedding them as antipodes – the sibling of a leaf embedded as $\mathbf{p}$ is embedded as $-\mathbf{p}$.

A property enjoyed by this embedding is that the L_2^2 distance induced by the inner product generates the graph shortest path metric, as shown in Figure 3. The depth of a node corresponds to squared norm of its vector embedding. Since nearby points in the data space are likely to belong to the same node if they have the same class label, the embedding scheme retains a degree of smoothness with respect to the data space.

In addition to the concatenated embedding $[\mathbf{x}, \mathbf{t}]$ where $\mathbf{x}$ is the original feature and $\mathbf{t}$ the decision tree path features, one can use other possibilities, for example, $[\mathbf{x}, \mathbf{t}, \mathbf{x} \otimes \mathbf{t}]$, and spectral embedding of the Gaussian kernel $K_{ij} = e^{-(\mathbf{t}_i - \mathbf{t}_j)^2/\sigma^2}$. Note that, in the limit of $\sigma \longrightarrow 0$, the kernel approaches the decision tree function $\mathbb{1}_{p_i(\mathbf{x})}$.

4.2 Learning the decision tree

It is well known that decision trees only learn axis parallel partitions. As a result, learning non-axis aligned separations causes the depth of the learned tree to be unnecessarily large and makes the learning process tedious as it requires a significant amount of additional resources. Therefore, when training the decision tree, we make sure that the focus is explicitly on learning the non-linearity by assigning the task of learning a linear separator to the linear AUC maximization algorithm from Section 3. This ensures that the decision tree does not have to relearn the linear separator that has already been learnt by the linear AUC maximizer.

The trained linear AUC maximizer is then used to compute the output scores for the data points after which the scores are appended to the original data as an additional feature. When training the decision tree, we observe that the first split typically occurs on the output scores generated by the linear AUC maximizer. After the initial split partitions the data, we upsample the two resulting data sets separately using SMOTE [9]. This is done to ensure that the positive and negative subtrees have sufficient samples from each class so that the learning process does not suffer from any imbalance that may be present in the original data. In the end, we use the learned decision tree paths to derive the vector embeddings.

5 PARTIAL AUC FORMULATION

Given a scoring function $Q : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbb{R}$, let $q_\alpha(\mathcal{X}_-, Q)$ denote the $(1-\alpha)$ quantile of the set of scores $\{Q(\mathbf{x}) | \mathbf{x} \in \mathcal{X}_-\}$. Thus, cardinality of the set $S_{\alpha,Q} \triangleq \{\mathbf{x} | Q(\mathbf{x}) \geq q_\alpha(\mathcal{X}_-, Q), \mathbf{x} \in \mathcal{X}_-\}$ satisfies the identity $|S_{\alpha,Q}| = \alpha|\mathcal{X}_-|$. Given these definitions, and making the dependence on $\mathbf{w}$ explicit, one can evaluate pAUC obtained by a linear scoring function $Q(\mathbf{x}) = \mathbf{w}^\top \cdot \mathbf{x}$ as

$$pAUC_\alpha(\mathbf{w}) = \frac{1}{N_+ N_- \alpha} \sum_{j \in S_{\alpha,\mathbf{w}}} \sum_{i \in \mathcal{X}_+} \mathbb{1}(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) > 0).$$

Given the difficulty of minimizing over indicator functions, we upper bound them by the hinge loss function obtaining

$$\min_{\mathbf{w}} \frac{1}{N_+ N_- \alpha} \sum_{j \in S_{\alpha,\mathbf{w}}} \sum_{i \in \mathcal{X}_+} h(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) < 0). \tag{5}$$

This optimization problem is significantly different from the Equation (2) presented in Section 3 because the domain of the summation over the set $\mathcal{X}_-$ (indexed by subscript j) is no longer independent of $\mathbf{w}$ and hence the reduction to the classification problem does not apply. To solve, we pose Equation (5) as an equivalent two person, zero-sum game between (i) an adversary(henceforth, $S-$player) that chooses S under the restrictions stated, to maximize the loss and (ii) the learner that tries to minimize the loss.

$$\min_{\mathbf{w}} \max_{\substack{S \subset \mathcal{X}_- \\ |S| = \alpha N_-}} \frac{1}{N_+ \alpha N_-} \sum_{i \in \mathcal{X}_+} \sum_{j \in S} h(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) < 0). \tag{6}$$

This equivalence holds because for any choice of $\mathbf{w}$ the set $S_{\alpha,\mathbf{w}}$ is indeed the most adversarial set that the $S-$player can choose under the stated restrictions. Marginalizing out S, we obtain the learner's loss function to be

$$D(\mathbf{w}) \triangleq \max_{\substack{S \subset \mathcal{X}_- \\ |S| = \alpha N_-}} \frac{1}{N_+ \alpha N_-} \sum_{i \in \mathcal{X}_+} \sum_{j \in S} h(\mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) < 0) \tag{7}$$

CLAIM 1. *$D(\mathbf{w})$ as defined in (7) is convex in $\mathbf{w}$.*

PROOF. For a fixed set S the RHS of (7) is convex because it is a sum of convex functions. The expression $D(\mathbf{w})$ is a maximum of a finite number of convex functions and is hence convex. □

Subgradient computation: The function $D(\mathbf{w})$ is not differentiable but is differentiable *almost everywhere*. Using subgradient calculus one can show [47] that its subgradient is obtained as

$$\partial D(\mathbf{w}) = \frac{1}{N_+ \alpha N_-} \sum_{i \in \mathcal{X}_+} \sum_{\substack{j \in S_{\alpha,\mathbf{w}} \\ \mathbf{w}^\top \cdot (\mathbf{x}_j^- - \mathbf{x}_i^+) < 1}} (\mathbf{x}_j^- - \mathbf{x}_i^+).$$

We minimize the loss function $D(\mathbf{w})$ by using the subgradient descent algorithm Pegasos [51].

5.1 Sampling

The subgradient steps discussed above require re-computation of $S_{\alpha,\mathbf{w}}$ at each $\mathbf{w}$ update. This would entail selecting the top α fraction of the re-scored negative examples incurring a time complexity of $O|\mathcal{X}_-|\log|\mathcal{X}_-|$. To make the the algorithm scalable, we stochastically estimate $q_\alpha(\mathcal{X}_-, Q)$ which corresponds to the $(1-\alpha)$ quantile of the scores induced on $\mathcal{X}_-$ by the updated weights.

This estimate is obtained by selecting a sample of size m from $\mathcal{X}_-$ uniformly at random by reservoir sampling [56] and is not repeated at every update of $\mathbf{w}$. The points in the sample are re-scored with updated weight at every update and the top k^{th} item, that is, the (k,m) order statistics is selected to yield our estimate $\hat{q}_\alpha(\mathcal{X}_-, Q)$. The values of k, m are chosen such that the expected value of $\mathbb{E}\mathbf{w}^\top \cdot \mathbf{x}_{k,m}$ equals $q_\alpha(\mathcal{X}_-, Q)$ with as low variance as desired.

In order to see how k and m may be chosen, consider $U_{k,m}$ to be the $(k,m)^{th}$ order statistics of a sample of size m drawn from a unit uniform distribution. Random variable $U_{k,m} \sim Beta(k, m-k+1)$, that is, it is distributed as a Beta distribution with parameters $k, m-k+1$ [2]. Thus,

$$\mathrm{E}[U_{k,m}] = \frac{k}{m+1}, \ \mathrm{Var}[U_{k,m}] = \frac{k(m-k+1)}{(k+m-k+1)^2(k+m-k+1+1)}$$

Imposing the constraints $\mathrm{E}[U_{k,m}] = \alpha$ and $\mathrm{Var}[U_{k,m}] = \sigma^2$ (where σ is the desired standard deviation of the estimate) one can easily solve for k and m as two unknowns in two equations. Note, in particular, that the solutions of k, m are independent of the size of $\mathcal{X}_-$. Our experiments are reported for a choice of $\sigma = \alpha/10$.

Algorithm 1: Partial AUC maximization algorithm

```
Input: $\mathcal{X}_+, \mathcal{X}_-, \alpha, n\_epochs$
Output: $\mathbf{w}$
1 $\mathbf{w}_0 = [\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_n]$ such that $\mathbf{w}_i \in U(-1.0, 1.0)$
2 for $t=1,...,n\_epochs$ do
3     for $\mathbf{x}_i^- \in \mathcal{X}_-$ do
          /* Uniformly sample $N_-$ */
4         $S = sample(\mathcal{X}_-, \alpha)$
          /* compute the score for subset of negative samples and sort them */
5         $Q(\mathbf{x}) = \mathbf{w}_t{}^\top \cdot \mathbf{x} \ \forall \mathbf{x} \in S$
          /* get the score at the $(1-\alpha)^{th}$ quantile */
6         $q_\alpha = (1-\alpha)$quantile of$Q(\mathbf{x})\ \mathbf{x} \in S$
7         if $\mathbf{w}_t{}^\top \cdot \mathbf{x}_i^- > q_\alpha$ then
              /* compute pairwise difference of $\mathbf{x}_i^-$ with all positive samples $\mathcal{X}_+$ */
8             $D(\mathbf{w}_t) = \frac{1}{N_+}\sum_{j \in \mathcal{X}_+} \mathbb{1}(\mathbf{w}_t{}^\top \cdot (\mathbf{x}_i^- - \mathbf{x}_j^+) > 0)$
              /* update weight vector by gradient descent */
9             $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\partial D(\mathbf{w}_t)$
```

Table 1: Summary of datasets

Data Source	Dataset	Number of instances	Number of features	imbalance ratio
LETOR 4.0	MQ2007	69,623	46	17:1
	MQ2008	15,211	46	15:1
UCI	a9a	48,842	123	3:1
	covtype	581,012	54	1:1
	ijcnn1	141,691	22	10:1
	letter_img	20,000	16	26:1

6 EXPERIMENTAL SETUP

In this section, we provide a description of the experimental setup and elaborate on the various benchmark datasets and baseline models. We broadly look at two real world scenarios: learning to rank task for extreme class imbalance and the more generic binary classification task for imbalanced datasets.

6.1 Datasets

Learning to Rank: We conduct our experiments on two benchmark datasets: MQ2007 and MQ2008 that are publicly available as part of the supervised ranking setting in LETOR 4.0 [43]. The MQ2007 has around 1,700 queries with an average of 41 labeled documents per query. This totals to 69,623 query-document pairs. The MQ2008 on the other hand has around 800 queries with an average of 19 labeled documents per query totalling to 15,211 query-document pairs. Each query-document pair is represented by a 46-dimensional feature vector in both the datasets and has a relevance score of 0, 1 or 2, where larger the relevance score, more relevant the query-document pair.

We transform the relevance scores into a binary label by setting a threshold such that, relevance score 2 indicates that the query-document pair is relevant (positive label) and relevance scores 0 and 1 indicate that the query-document pair is irrelevant (negative label). Table 1 summarizes the number of instances, features and imbalance ratio for the datasets. Both datasets contain a training, a validation, and a test set. We tune the hyperparameters on the validation set and perform the final model evaluation on the held-out test set. Both MQ2007 and MQ2008 can be downloaded from LETOR 4.0[1].

Binary Classification: To demonstrate that the proposed pAUC maximization algorithm generalises to other binary classification tasks having class imbalance, we additionally perform experiments on a number of benchmark datasets obtained from the UCI machine learning repository [13], where pAUC is a performance measure of interest. We look at problems in different domains:

- **a9a** - The goal is predict whether income exceeds \$50,000 per year based on census data.
- **covtype** - The goal is to predict the forest cover type. We use the binary class version of this dataset.
- **ijcnn1** - This is first problem (generalization ability challenge) of IJCNN challenge 2001 [10].

[1]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/

- **letter_img** - Here, the objective is to identify the capital letter 'Z' in the English alphabet from a large number of black-and-white rectangular pixel displays.

Table 1 summarizes the number of instances and features present in each dataset. In each case, we do a random 75 : 25 split (while maintaining class distribution) where 75% of the data is used for training and the remaining 25% of the data is used for testing. The processed versions for a9a, covtype and ijcnn1 was downloaded from LIBSVM[2] and letter_img was downloaded from the python imblearn.datasets package[3].

6.2 Models

Learning to Rank: We compare the performance of our proposed method (pAUC_{max}) for learning to rank with the AUC maximization model (AUC_{max}) and two other state of the art baseline models:

- **LambdaMART** [6]: LambdaMART, a combination of LambdaRank and MART (Multiple Additive Regression Trees), is a popular ranking algorithm used in commercial search engines. It makes use of gradient boosted decision trees using a cost function derived from LambdaRank for optimizing ranking metrics like NDCG, MRR, AUC and so on. We used the implementation from the python learning to rank toolkit (pyltr) [4] for building the LambdaMART model.
- **RankSVM** [23]: RankSVM, a variation of the support vector machine (SVM) algorithm [54], is a classic learning to rank model that employs pairwise ranking methods to automatically sort results based on the document relevancy for a given query. We used the implementations in SVM^{rank}[5].

In our learning to rank experiments, we tune the models to optimize for AUC during the training process. We tune other hyperparameters with guidance from prior work and further fine-tune them on the validation set. We evaluate the performance of the ranking models on imbalanced data by comparing both the AUC as well as the pAUC in the FPR range $[0.0, 0.1]$.

Binary Classification: We also conduct experiments to compare the performance of our proposed method (pAUC_{max}) for partial AUC maximization with the AUC maximization model (AUC_{max}) and five other state of the art baseline models:

- $\textbf{SVM}_{\textbf{AUC}}$ [22]: Structural SVM algorithm that optimizes multivariate non-linear performance measures like the F1 score. We use this model to optimise for AUC. The SVM_{AUC} algorithm is implemented using the publicly available API[55][6]
- $\textbf{SVM}_{\textbf{pAUC}}$ [34]: Builds on the structural SVM (SVM_{AUC}) framework [22] to design convex surrogates that optimizes for pAUC in FPR range $[\alpha, \beta]$ using a cutting plane solver. We used the implementation provided by [34][7].
- $\textbf{SVM}^{\textbf{dc}}_{\textbf{pAUC}}$ [35]: Building on SVM_{pAUC} [34], SVM^{dc}_{pAUC} maximizes the pAUC in FPR range $[\alpha, \beta]$ using a tighter approximation that directly optimizes the non-convex surrogate using the well known non-convex optimization technique based on difference-of-convex programming. We used the implementation provided by [35][8]
- **pAUCBooost** [25]: A boosting based algorithm that optimizes partial AUC in the general false positive ranges of the form $[\alpha, \beta]$.
- **GreedyHeuristic** [46]: An entension of the greedy heuristics method in [46] that maximizes pAUC by means of linear combination of classifiers.

For the binary classification experiments, we optimize for pAUC. We use 75% of the data for training and the remaining 25% for testing. We evaluate the performance of the various models by comparing the pAUC in the FPR range $[0.02, 0.05]$.

Table 2: Evaluating the usefulness of the vector embeddings from learned decision tree paths in pAUC maximization in FPR range [0.02, 0.05] on UCI datasets.

Dataset Type	pAUC(0.02, 0.05)			
	a9a	**covtype**	**ijcnn1**	**letter_img**
Concatenated	**0.3978**	**0.5751**	**0.7206**	**0.9276**
Raw	0.2828	0.4354	0.4742	0.9063
SMOTE	0.1980	0.1053	0.3026	0.5595

7 RESULTS

In this section, we present the results of our experiments. We start by looking at the usefulness of the vector embeddings from learned decision tree paths. We then evaluate the efficacy of the proposed method for optimizing partial AUC. In particular, we compare the performance of our method with other state-of-the art baseline models, firstly, for the task of learning to rank and then examine the generalizability of the proposed model by looking at binary classification tasks in several domains where class imbalance is a prominent issue. Additionally, we perform a run time analysis to compare the proposed method with other baseline models.

7.1 Usefulness of decision tree features

In Section 4, we introduced the novel use of learned decision tree paths as vector embeddings to capture any inherent non-linearity in data to overcome the limitation of the method only learning linear separators. We empirically examine those claims in this section. We consider the following variations of the dataset:

- **Raw data** - This is the original dataset as is.
- **Concatenated data** - For this dataset, we first derive the vector embeddings using the learned decision tree paths and append it to each of the raw data samples.
- **SMOTE data** - This dataset is formed by upsampling the minority (positive) class samples using SMOTE [9]. We use SMOTE to ensure that the positive samples are not simply duplicated and have some variations in the features.

[2]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html
[3]https://imbalanced-learn.readthedocs.io/en/stable/datasets/
[4]https://libraries.io/pypi/pyltr
[5]https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[6]http://svmlight.joachims.org/svm_struct.html
[7]http://clweb.csa.iisc.ac.in/harikrishna/Papers/SVMpAUC/
[8]http://clweb.csa.iisc.ac.in/harikrishna/Papers/SVMpAUC-tight/

Table 3: pAUC maximization in [0.0, 0.1] on LETOR datasets

Approach	MQ2007				MQ2008			
	pAUC[0,0.1]		pAUC[0,1] = AUC		pAUC[0,0.1]		pAUC[0,1] = AUC	
	pAUC	Δ pAUC	AUC	Δ AUC	pAUC	Δ pAUC	AUC	Δ AUC
pAUC_{max} on concatenated data	**0.1381**	0.0	**0.6562**	0.0	**0.2291**	0.0	**0.7826**	0.0
AUC_{max} on concatenated data	0.1017	+ 0.0364	0.6137	+ 0.0425	0.2079	0.0212	0.7349	+ 0.0477
AUC_{max} on raw data	0.0932	+ 0.0449	0.6121	+ 0.0441	0.1731	0.0560	0.7175	+ 0.0651
LambdaMART	0.0774	+ 0.0607	0.6085	+ 0.0477	0.1440	0.0851	0.6925	+ 0.0901
RankSVM*	-	-	-	-	0.0499	0.1792	0.4998	+ 0.2828

Table 4: Comparing the ranking performance using NDCG@k on LETOR datasets

Approach	MQ2007				MQ2008			
	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@1	NDCG@3	NDCG@5	NDCG@10
pAUC_{max} on concatenated data	**0.3461**	**0.4018**	**0.4519**	**0.4861**	**0.3141**	**0.3839**	**0.4415**	**0.4813**
LambdaMART	0.3397	0.3882	0.4422	0.4798	0.3075	0.3599	0.4087	0.4628

To compare the performance of the various dataset types, we train and test the AUC Maximization algorithm (described in Section 3) with each one of the three dataset types for all the UCI datasets. Table 2 summarizes the resulting pAUC in the FPR interval $[0.02, 0.05]$, computed over the test data, as obtained by the AUC Maximization algorithm for the different datasets types. It is evident from the numbers that the non-linearity introduced by the vector embeddings derived from the learned decision tree paths significantly helps improve the performance of the AUC maximization algorithm, even outperforming the SMOTE upsampled data, which is a common strategy employed by several people to overcome class imbalance.

7.2 Learning to Rank

In this section, we analyse the results of our experiments comparing the performance of our proposed model (pAUC_{max}) with other state of the art baseline models for maximising pAUC in the $[0, \alpha]$ range on the two supervised learning to rank datasets from LETOR 4.0, namely, MQ2007 and MQ2008. We report the pAUC scores in the FPR range $[0.0, 0.1]$ and $[0, 1]$ (corresponding to AUC) on the held-out test sets in Table 3. We also include the Δ pAUC and Δ AUC scores which corresponds to the difference in performance of our proposed model (pAUC_{max}) on the concatenated data with the baseline models. *Note that we do not report the score for RankSVM on MQ2007 as the model ran into a memory error and failed to train.

A comparison of AUC_{max} on raw data and the concatenated reemphasises the usefulness of the embeddings derived from the learned decision tree paths and show that the non-linearity introduced aids in maximizing the pAUC$[0, 0.1]$ and AUC scores. We did not find the results from RankSVM useful as it performs poorly on both the datasets. While LambdaMART outperforms RankSVM by a large margin, we observe that our proposed model (pAUC_{max}) with the concatenated data has a significant improvement from the baseline models for both pAUC$[0.0, 0.1]$ (improvement rate: +6% for MQ2007 and +8.5% for MQ2008) and the overall AUC (improvement rate: +4.7% for MQ2007 and +9% for MQ2008). This shows that our proposed method outperforms state of the art models for learning to rank with imbalanced classes.

Additionally, we compare the performance of the proposed model with LambdaMART on the two LETOR 4.0 datasets using NDCG@k, a common evaluation metric using in learning to ranking. We do not report the scores for RankSVM due to it's poor performance. Table 4 summarizes the NDCG@k for k values 1, 3, 5 and 10. We see a 0.6% to 4% improvement in scores, therefore the performance of the proposed model is comparable to that of LambdaMART.

7.3 Binary Classification

In our final set of experiments conducted, we look at the generalizabilty of the proposed method (pAUC_{max}) for binary classification problems having class imbalance where we maximize the pAUC in the more generic $[\alpha, \beta]$ range. We compare the performance of pAUC_{max} with other state of the art baseline models for optimizing pAUC on a number of benchmark UCI datasets and report the pAUC scores for the FPR range $[0.02, 0.05]$ on the held-out test sets in Table 5. We also include the Δ pAUC to highlight the difference in performance of pAUC_{max} on the concatenated data with each of the baseline models.

Upon analysing the numbers, we see that using the pAUC maximization algorithm on the concatenated data outperforms all of the state of the art baseline methods with significant improvements in results across all datasets. While the difference in performance on the a9a dataset, ranging from +0.36% to +16.35%, was not as prominent, we observe a huge leap in performance on the letter_img dataset, ranging from +44.40% to +67.60%. We also observe substantial increase in performance on covtype and ijcnn1 ranging from +15.80% to +36.79% and +8.43% to +64.40% respectively. This shows that our proposed method can also be applied to general binary classification problems having imbalanced classes and has significant improvement over existing state of the art baseline models.

Table 5: pAUC maximization in [0.02, 0.05] on UCI datasets

Approach	pAUC(0.02, 0.05)							
	a9a		covtype		ijcnn1		letter_img	
	pAUC	Δ pAUC	pAUC	Δ pAUC	pAUC	Δ pAUC	pAUC	Δ pAUC
pAUC_{max} on concatenated data	**0.4374**	0.0	**0.6065**	0.0	**0.7641**	0.0	**0.9648**	0.0
AUC_{max} on concatenated data	0.3978	+ 0.0396	0.5751	+ 0.0314	0.7206	+ 0.0435	0.9276	+ 0.0372
AUC_{max} on raw data	0.2828	+ 0.1546	0.4354	+ 0.1711	0.4742	+ 0.2899	0.9063	+ 0.0585
AUC_{max} on smote data	0.1980	+ 0.2394	0.1053	+ 0.5012	0.3026	+ 0.4615	0.5595	+ 0.4053
SVM_{AUC}	0.4338	+ 0.0036	0.2987	+ 0.3078	0.4750	+ 0.2891	0.4455	+ 0.5193
SVM_{pAUC}	0.2739	+ 0.1635	0.2467	+ 0.3598	0.6131	+ 0.1510	0.5208	+ 0.4440
SVM^{dc}_{pAUC}	0.3650	+ 0.0724	0.2410	+ 0.3655	0.6798	+ 0.0843	0.5182	+ 0.4466
pAUCBooost	0.4012	+ 0.0362	0.4485	+ 0.1580	0.4913	+ 0.2728	0.4954	+ 0.4694
GreedyHeuristic	0.3417	+ 0.0957	0.2386	+ 0.3679	0.1201	+ 0.6440	0.2888	+ 0.6760

Table 6: Training time in seconds (relative to pAUC_{max})

Approach	MQ2007	MQ2008
pAUC_{max} on concatenated data	907.52 (t)	561.56 (t)
LambdaMART	654.15 (0.72t)	397.81 (0.71t)
RankSVM*	-	5098.12 (9.08t)

7.4 Run time analysis

We compare the running time of our proposed method with the baseline models for the learning to rank problem. Table 6 summarizes the both the absolute time taken in seconds and the relative time taken by the baseline models as a factor of the time taken by the proposed model. We observe that the running time of our model is significantly better than RankSVM and is comparable to that of LambdaMART. *Note that we do not report the score for RankSVM on MQ2007 as the model ran into a memory error and failed to train.

8 DISCUSSION

There are a number of implications of this work. Firstly, the novel use of learned decision tree paths as vector embeddings to introduce non-linearity has been empirically proven to improve the performance of the AUC and pAUC maximization models. These task-specific embeddings can be combined with potentially any method that only learns a linear separator, thereby extending the method to datasets that are not linearly separable. Secondly, class imbalance is a promiment issue in a number of real world applications, our proposed method is generalizable and can be applied to problems in any domain where the goal is to optimize the pAUC in a given FPR range.

Although we have presented our work in terms of the hinge loss surrogate function, it is straightforward to extend the technique to other such functions, for example, logistic loss, exponential loss [45], etc. It is recommended to use surrogates functions that are proper scoring rules [45].

One current limitation of the approach is the restriction to the binary class problem. It would be fruitful to conside how to apply these techniques to the partial variants of generalizations of AUC to multiple classes [17, 24]

9 CONCLUSION

Given the increasing popularity in the usage of pAUC, as an improvement over AUC, for evaluating the performance of models in class imbalanced scenarios. We propose an algorithm for optimizing the pAUC in the FPR range $[0, \alpha]$ by formulating the minimization of pAUC loss as a two person zero-sum game between (i) an adversary that selects a fixed fraction of negative examples and (ii) a scoring function that needs to assign positive examples higher scores, no matter the choice of the adversary. The optimal scoring function is obtained as an equilibrium of this game. We eliminate the restriction to linear separators with the use of an efficient, task specific vector embedding technique that captures the geometry induced by decision trees, thereby extending the method to datasets that are not linearly separable.

We evaluate both the usefulness of the vector embeddings as well as the efficacy of the proposed method for optimizing partial AUC. More specifically, we compare the performance of our method with other state-of-the art baseline models, firstly, for the task of learning to rank and then examine the generalizability of the proposed model by looking at binary classification tasks in several domains where class imbalance is a prominent issue. The empirical results indicate that the proposed method does in fact optimize the partial AUC in the desired false positive range, outperforming the existing baseline techniques by a significant margin.

REFERENCES

[1] Shivani Agarwal, Thore Graepel, Ralf Herbrich, Sariel Har-Peled, and Dan Roth. 2005. Generalization bounds for the area under the ROC curve. *Journal of Machine Learning Research* 6 (21 July 2005).

[2] B.C. Arnold, N. Balakrishnan, and H.N. Nagaraja. 2008. *A First Course in Order Statistics.* Society for Industrial and Applied Mathematics.

[3] Kaan Ataman, W Nick Street, and Yi Zhang. 2006. Learning to rank by maximizing AUC with linear programming. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings.* IEEE, 123–129.

[4] Mohamed Bekkar, Hassiba Kheliouane Djemaa, and Taklit Akrouf Alitouche. 2013. Evaluation measures for models assessment over imbalanced data sets. *J Inf Eng Appl* 3, 10 (2013).

[5] Andrew P. Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30, 7 (1997), 1145 – 1159. https://doi.org/10.1016/S0031-3203(96)00142-2

[6] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning.* 129–136.

[8] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 621–630.

[9] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[10] Chih chung Chang and Chih-Jen Lin. 2001. IJCNN 2001 Challenge: Generalization Ability and Text Decoding. In *In Proceedings of IJCNN. IEEE*. 1031–1036.

[11] Corinna Cortes and Mehryar Mohri. 2004. AUC optimization vs. Error rate minimization. In *Advances in Neural Information Processing Systems 16 - Proceedings of the 2003 Conference, NIPS 2003 (Advances in Neural Information Processing Systems)*. Neural information processing systems foundation.

[12] Dotan Di Castro, Zohar Karnin, Liane Lewin-Eytan, and Yoelle Maarek. 2016. You've Got Mail, and Here is What You Could Do With It! Analyzing and Predicting Actions on Email Messages. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. 307–316.

[13] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[14] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of machine learning research* 4, Nov (2003), 933–969.

[15] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[16] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. 2010. Social media recommendation based on people and tags. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 194–201.

[17] D.J. Hand and R.J. Till. 2001. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning* 45 (2001), 171–186.

[18] Irit Hochberg, Deeb Daoud, Naim Shehadeh, and Elad Yom-Tov. 2019. Can internet search engine queries be used to diagnose diabetes? Analysis of archival search data. *Acta diabetologica* 56, 10 (2019), 1149–1154.

[19] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. 2008. Kernel methods in machine learning. *The annals of statistics* (2008), 1171–1220.

[20] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[21] Jonathan L Jesneck, Loren W Nolte, Jay A Baker, Carey E Floyd, and Joseph Y Lo. 2006. Optimized approach to decision fusion of heterogeneous data for breast cancer diagnosis. *Medical physics* 33, 8 (2006), 2945–2954.

[22] Thorsten Joachims. 2005. A Support Vector Method for Multivariate Performance Measures. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*. Association for Computing Machinery, New York, NY, USA, 377–384.

[23] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 217–226.

[24] Ross Kleiman and David Page. 2019. AUCμ: A Performance Metric for Multi-Class Machine Learning Models. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 97. 3439–3447.

[25] O. Komori and S. Eguchi. 2010. A boosting method for maximizing the partial area under the ROC curve. *BMC bioinformatics* (2010). https://doi.org/10.1186/1471-2105-11-314

[26] Ping Li, Qiang Wu, and Christopher J Burges. 2008. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*. 897–904.

[27] Charles X Ling, Jin Huang, and Harry Zhang. 2003. AUC: a better measure than accuracy in comparing learning algorithms. In *Conference of the canadian society for computational studies of intelligence*. Springer, 329–341.

[28] Charles X. Ling, Jin Huang, and Harry Zhang. 2003. AUC: A Statistically Consistent and More Discriminating Measure than Accuracy. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 519–524.

[29] Bin Liu and Yulin Zhu. 2019. ProtDec-LTR3. 0: protein remote homology detection by incorporating profile-based features into Learning to Rank. *Ieee Access* 7 (2019), 102499–102507.

[30] Qiaoling Liu and Eugene Agichtein. 2011. Modeling answerer behavior in collaborative question answering systems. In *European Conference on Information Retrieval*. Springer, 67–79.

[31] Tie-Yan Liu. 2011. *Learning to rank for information retrieval*. Springer Science & Business Media.

[32] Claudio Marrocco, Robert PW Duin, and Francesco Tortorella. 2008. Maximizing the area under the ROC curve by pairwise feature combination. *Pattern Recognition* 41, 6 (2008), 1961–1974.

[33] Taesup Moon, Alex Smola, Yi Chang, and Zhaohui Zheng. 2010. IntervalRank: isotonic regression with listwise and pairwise constraints. In *Proceedings of the third ACM international conference on Web search and data mining*. 151–160.

[34] Harikrishna Narasimhan and Shivani Agarwal. 2013. A Structural SVM Based Approach for Optimizing Partial AUC. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013 (JMLR Workshop and Conference Proceedings)*, Vol. 28. JMLR.org, 516–524.

[35] H. Narasimhan and S. Agarwal. 2016. Support Vector Algorithms for Optimizing the Partial Area Under the ROC Curve. *Neural Computation* 29 (2016).

[36] Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Learning to rank code examples for code search engines. *Empirical Software Engineering* 22, 1 (2017), 259–291.

[37] Shuzi Niu, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2012. Top-k Learning to Rank: Labeling, Ranking and Evaluation.

[38] Matthew Norton and Stan Uryasev. 2019. Maximization of AUC and Buffered AUC in Binary Classification. *Math. Program.* 174, 1–2 (March 2019), 575–612. https://doi.org/10.1007/s10107-018-1312-2

[39] Komori O and Eguchi S. 2010. A boosting method for maximizing the partial area under the ROC curve. *BMC Bioinformatics* (2010). https://doi.org/10.1186/1471-2105-11-314

[40] Daan Odijk and Anne Schuth. 2017. Online learning to rank for recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 348–348.

[41] S. Paisitkriangkrai, C. Shen, and A. Van Den Hengel. 2013. Efficient pedestrian detection by directly optimizing the partial area under the ROC curve. In *Proceedings of the IEEE International Conference on Computer Vision*. 1057–1064.

[42] Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. 2006. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins: Structure, Function, and Bioinformatics* 63, 3 (2006), 490–500. https://doi.org/10.1002/prot.20865

[43] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. arXiv:cs.IR/1306.2597

[44] Dimitrios Rafailidis and Fabio Crestani. 2017. Learning to rank with trust and distrust in recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 5–13.

[45] Mark D. Reid and Robert C. Williamson. 2011. Information, Divergence and Risk for Binary Experiments. *Journal of Machine Learning Research* 12, 22 (2011), 731–817.

[46] Maria Teresa Ricamato and Francesco Tortorella. 2011. Partial AUC Maximization in a Linear Combination of Dichotomizers. *Pattern Recogn.* 44, 10–11 (Oct. 2011), 2669–2677. https://doi.org/10.1016/j.patcog.2011.03.022

[47] R. Tyrrell Rockafellar. 1970. *Convex analysis*. Princeton University Press, Princeton, N. J.

[48] Xiaoqing Ru, Lida Wang, Lihong Li, Hui Ding, Xiucai Ye, and Quan Zou. 2020. Exploration of the correlation between GPCRs and drugs based on a learning to rank algorithm. *Computers in Biology and Medicine* 119 (2020), 103660.

[49] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.

[50] David Sculley. 2010. Combined regression and ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 979–988.

[51] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming* 127, 1 (2011), 3–30.

[52] Yue Shi, Martha Larson, and Alan Hanjalic. 2010. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*. 269–272.

[53] Zhendong Shi, Jacky Keung, Kwabena Ebo Bennin, and Xingjun Zhang. 2018. Comparing learning to rank techniques in hybrid bug localization. *Applied Soft Computing* 62 (2018), 636–648.

[54] Johan AK Suykens and Joos Vandewalle. 1999. Least squares support vector machine classifiers. *Neural processing letters* 9, 3 (1999), 293–300.

[55] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2005. Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research* 6 (Sept. 2005), 1453–1484.

[56] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.

[57] X Wang, M Bendersky, D Metzler, and M Najork. 2016. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 115–124.

[58] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.