Discover Computing

Research

Learning bivariate scoring functions for ranking

Franco Maria Nardini[1] · **Roberto Trani**[1] · **Rossano Venturini**[2]

Received: 29 July 2022 / Accepted: 24 May 2024
Published online: 27 September 2024

Abstract
State-of-the-art Learning-to-Rank algorithms, e.g., λ MART, rely on univariate scoring functions to score a list of items. Univariate scoring functions score each item independently, i.e., without considering the other available items in the list. Nevertheless, ranking deals with producing an effective ordering of the items and comparisons between items are helpful to achieve this task. Bivariate scoring functions allow the model to exploit dependencies between the items in the list as they work by scoring pairs of items. In this paper, we exploit item dependencies in a novel framework—we call it the *Lambda Bivariate* (LB) framework—that allows to learn effective bivariate scoring functions for ranking using gradient boosting trees. We discuss the three main ingredients of LB: (*i*) the invariance to permutations property, (*ii*) the function aggregating the scores of all pairs into the per-item scores, and (*iii*) the optimization process to learn bivariate scoring functions for ranking using any differentiable loss functions. We apply LB to the λ RANK loss and we show that it results in learning a bivariate version of λ MART—we call it BI- λ MART—that significantly outperforms all neural-network-based and tree-based state-of-the-art algorithms for Learning-to-Rank. To show the generality of LB with respect to other loss functions, we also discuss its application to the SOFTMAX loss.

Keywords Learning-to-Rank · Bivariate scoring functions · Pairwise scoring

1 Introduction

Ranking is a challenging machine learning task that aims to produce an ordering of a list of items that maximize a listwise utility metric. Differently from classification and regression, ranking focuses on the ordering of the items rather than on a specific class or value predicted for each item. In the last years, the ranking problem has been addressed using machine learning, in the field known as Learning-to-Rank (LtR), and several approaches for solving this task have been devised [1]. Most of the existing LtR approaches rely on univariate scoring functions that estimate the ranking score of one item at a time in isolation, i.e., without considering the other available items in the list. Although this approach is proven to be effective, it is intrinsically limited as univariate scoring functions cannot compare the available items in the list to exploit their dependencies, which may be very helpful to rank them. For this reason, multivariate scoring functions that estimate the ranking score of each item as a function of multiple items of the list can effectively take into account the items dependencies when scoring.

Recently, several works employ neural networks to learn multivariate scoring functions to effectively exploit the item dependencies for ranking. Ai et al. and Bello et al. use recurrent neural networks for re-ranking to exploit contextual information available in the list of results [2, 3]. Similarly, Ai et al. propose Groupwise Scoring Functions (GSF), a new

✉ Roberto Trani, roberto.trani@isti.cnr.it; Franco Maria Nardini, francomaria.nardini@isti.cnr.it; Rossano Venturini, rossano.venturini@unipi.it | [1]ISTI-CNR, Pisa, Italy. [2]University of Pisa, Pisa, Italy.

framework to learn multivariate scoring functions for ranking using deep neural networks [4]. More recently, Pasumarthi et al. and Pang *et al.* propose the application of attention [5] to learn neural networks for ranking. The two methods, called ATTN-DIN [6] and SETRANK [7], respectively, are able to effectively capture contextual information and cross-item interactions from the list of items to rank. However, all the proposed contributions towards multivariate scoring functions are limited to neural network models and cannot be directly exploited to learn gradient boosting trees [8]. This is an important limitation as gradient boosting trees models, e.g., λ MART [9, 10], achieve state-of-the-art performance on Learning-to-Rank reference datasets.

Novel contributions To overcome the limitations discussed above, we propose the *Lambda*[1] *Bivariate* (LB) framework, a novel framework for learning bivariate scoring functions for ranking using gradient boosting trees. LB works by scoring all pairs of items in the list. Differently from existing solutions based on multivariate scoring functions, LB can be used with any loss function. Moreover, LB defines three requirements to learn effective bivariate scoring functions: (*i*) the invariance to permutations property, (*ii*) the function aggregating the scores of all pairs into the per-item scores, (*iii*) the optimization process to learn the bivariate scoring function that optimizes a target utility function. First, the invariance to permutations guarantees the independence of the scores of the items from their input order. We formally introduce it in LB because, since we use tree-based models, we cannot rely on a specific model architecture, e.g., symmetric neural networks, to learn such property [6, 7]. Second, the aggregation function is required to produce a final score for an item given the outcomes of the bivariate scoring function providing scores for pairs of items. Third, given an arbitrary loss function, we discuss how to optimize it in a pairwise fashion by exploiting gradient boosting trees.

We apply LB to the λ RANK loss [9, 11] and we show that it results in learning a bivariate version of λ MART—we call it BI- λ MART—that significantly outperforms all neural-network-based and tree-based state-of-the-art algorithms for Learning-to-Rank. To show the generality of LB to other loss functions, we also discuss its application to the SOFTMAX loss [12], which is a popular and effective listwise loss for neural networks [4, 6, 13].

To summarize, the novel and unpublished contributions presented in this paper are:

- We introduce LB, a novel framework for learning bivariate scoring functions for ranking using gradient boosting trees. We first discuss the main requirements that allow LB to effectively learn bivariate scoring functions. We then show that it can be easily employed with any loss function;
- We apply LB to the λ RANK loss [9, 11] and we define the resulting model as BI- λ MART, which represents a bivariate version of the state-of-the-art λ MART algorithm [10];
- We present a comprehensive experimental evaluation of BI- λ MART on three public Learning-to-Rank datasets. Results show that BI- λ MART outperforms all state-of-the-art neural-network-based models by a large margin and also outperforms λ MART with a relative improvement in terms of NDCG@5 ranging from 0.5% to 1.2%. To show the generality of the proposed framework, we also assess LB with the SOFTMAX loss;
- To allow the reproducibility of the results, we release our implementation of LB as open source upon acceptance of the paper.

Paper structure Section 2 discusses related work. Section 3 formalizes the problem while Sect. 4 introduces our proposed LB framework for learning bivariate scoring functions. We then discuss the experimental setup in Sect. 5 and we propose a comprehensive experimental evaluation of LB against state-of-the-art competitors in Sect. 6. Finally, Sect. 7 concludes the work and discusses future directions.

2 Related work

In the last years, the ranking problem has been extensively studied [1, 14] and several effective machine learning algorithms have been introduced [15, 16]. Learning-to-Rank algorithms are usually categorized by the loss function they employ, i.e., pointwise, pairwise, and listwise approaches.

Pointwise approaches aim to directly estimate the ground truth label of a single item, i.e., its relevance score, by employing a loss function defined on a per-item basis [17–20]. Therefore, pointwise approaches do not focus on directly optimizing the order of the items in the list.

[1] "Lambda" here origins from the lambda notation in mathematics and computer programming.

Pairwise approaches, instead, employ a loss function defined on pairs of items, exploiting the information of two items at a time, to learn the relative ordering of the items in the list [21–25]. Pairwise approaches can be further divided into approaches that employ bivariate scoring functions and approaches that employ univariate scoring functions. The approaches in the former category learn preferences, i.e., order relationships between pairs of items [26]. A recent work in this direction is the one by Dehghani et al. [27], where authors employ a bivariate neural network model, RankProb, that estimates the probability that one item is more relevant than the other. RankProb estimates the probability that one item is more likely being relevant than the other one. This problem is addressed as a binary classification task and the ranking score of an item is computed by averaging its pairwise preference scores. However, RankProb does not directly deal with the ranking task as it aims to learn a pairwise classifier separating the pairs, instead of optimizing the induced ranking. Moreover, the approaches falling in this category may lead to conflicts during the final sorting of the items as the learned preferences could not be transitive, e.g., a circular dependency among three or more items may exist and any possible ranking would break at least one preference. This problem can be addressed by finding an ordering of the items that minimizes the number of broken preferences, which is known to be NP-Hard [28], or that approximates it [29]. Nevertheless, well-known pairwise approaches employ a univariate scoring function. This means that the algorithm learns a univariate function that scores one item at a time and the function is learned by optimizing a pairwise loss function defined over pairs of items, e.g., RankBoost [24] and RankNet [23]. Pairwise approaches solve some of the issues of pointwise approaches. However, learning the relative ordering of the items is a more complex task than ranking the items. Indeed, the optimization of the relative order of the majority of the pairs of items of the list does not guarantee that the most relevant items would be ranked higher, i.e,. we may improve the ranking by focusing more on the most relevant items.

Listwise approaches overcome these limitations by employing a loss function defined on the list of items to directly optimize a given ranking metric [10–12, 30–32]. Since the rank is not a continuous and differentiable function, it cannot be optimized using classical gradient-based machine learning algorithms. To overcome this limitation, several continuous and differentiable approximations of listwise ranking metrics have been proposed [11, 33, 34]. The state-of-the-art listwise algorithm is λ MART [9], which won the "Yahoo! Learning to Rank Challenge" [35]. λ MART exploits a combination of λ Rank loss [11, 36] and gradient boosting trees [8].

Neural networks for ranking Most of the existing Learning-to-Rank algorithms share a common limitation: they learn univariate scoring functions that score each item independently from the other items of the list. Some recent approaches overcome this limitation. A first contribution in this line is by Ai et al., where authors propose to learn a Deep Listwise Context Model (DLCM) to be used for ranking a list of candidate items [2]. Authors employ a recurrent neural network (RNN) to sequentially encode a list of items and learn a local context model representing the list, then use it to re-rank the items. Authors show that DLCM can effectively capture the local ranking context based on the complex interactions between the items. The application of a sequence-to-sequence recurrent neural network for ranking is also proposed by Bello et al. [3]. The authors employ a RNN to predict the next "best" item to select, given the items already selected. Recently, Ai et al. propose GSF, a new framework for learning groupwise scoring functions [4]. The framework relies on neural networks to jointly learn the relevance scores of groups of items at a time, thus exploiting cross-item dependencies when scoring the groups. The GSF framework exploits the ability of neural networks to model multivariate scoring functions.

More recent approaches exploit attention [5] to learn effective neural networks for ranking. In this line, Pasumarthi et al. propose attn-DIN, a new approach that exploits self-attention item interaction networks for ranking under the multivariate scoring paradigm [6]. Authors show that attn-DIN can automatically learn permutation-equivariant representations, i.e., the scores it produces do not depend from the position of the items in the input, to capture item interactions without any auxiliary information. A second contribution exploiting neural networks and attention is SetRank by Pang et al. [7]. SetRank is a self-attention network that satisfies the permutation-equivariant requirement. Authors show that the self-attention mechanism allows SetRank to capture both the local context information from the cross-item interactions and to learn permutation-equivariant representations for the items.

The methods above exploit neural networks with "complex" architectures to learn multivariate scoring functions. This peculiarity limits the applicability of these techniques to gradient boosting trees, which still achieve state-of-the-art performance on public LtR datasets. We overcome this limitation by proposing LB, a new framework to learn bivariate scoring functions with gradient boosting trees and by showing that LB can be applied to learn effective bivariate scoring functions.

3 Problem formulation

This section provides a formulation of the ranking problem in the Learning-to-Rank (LtR) setting. In LtR, a list of items is associated to a query and each item, represented through a vector of m features, is labeled with its relevance score for the query. Let Q be the set of queries and n_q be the number of items of the query $q \in Q$. We can associate each query q to a matrix $\boldsymbol{X}^{(q)} \in \mathbb{R}^{n_q,m}$ and a vector $\boldsymbol{y}^{(q)} \in \mathbb{R}^{n_q}$ composed of all feature vectors and relevance labels of the items of q, respectively.

A Learning-to-Rank algorithm learns a ranking model $\Phi : \mathbb{R}^{n,m} \to \mathbb{R}^n$ that produces a score for each item of a query q, i.e., $\boldsymbol{s}^{(q)} = \Phi(\boldsymbol{X}^{(q)})$. The ranking model is learned by optimizing a given loss function $\mathcal{L}$, which is defined in terms of the loss function $\ell(\cdot)$ of the relevance labels and the estimated scores on a per-query basis

$$\mathcal{L}(\Phi) = \frac{1}{|Q|} \sum_{q \in Q} \ell\left(\boldsymbol{y}^{(q)}, \Phi(\boldsymbol{X}^{(q)})\right). \tag{1}$$

Learning-to-Rank algorithms are characterized mainly by the ranking model Φ learned and the loss function ℓ exploited. For instance, RankingSVM employs a linear model and a hinge loss [22], RankNet applies a neural network model and a cross entropy loss [23], while λ MART uses a gradient boosting trees model with a weighted cross entropy loss [10].

Most of the existing LtR algorithms score all items independently, i.e., the algorithm finds a ranking model Φ_U that applies the same *univariate scoring function* $\phi_U : \mathbb{R}^m \to \mathbb{R}$ to each feature vector representing a single item:

$$\Phi_U(\boldsymbol{X}^{(q)}) = \left\langle \phi_U(\boldsymbol{X}_1^{(q)}), \ldots, \phi_U(\boldsymbol{X}_{n_q}^{(q)}) \right\rangle. \tag{2}$$

A consequence of the application of univariate scoring functions is that an item will always get the same score/rank regardless of the other items available for the query. Instead, a ranking model Φ_M that relies on a *multivariate scoring function* $\phi_M : \mathbb{R}^{n,m} \to \mathbb{R}^n$ can produce more precise scores as it can ideally take into account all the items of the same query when scoring, i.e., $\Phi_M(\boldsymbol{X}^{(q)}) = \phi_M(\boldsymbol{X}^{(q)})$. For this reason, multivariate scoring functions are theoretically more effective than univariate ones as they can compare items and capture cross-item dependencies when scoring. Ideally, a multivariate scoring function should be: *i*) invariant to permutations of the items, i.e., the score assigned by ϕ_M to an item should not depend on its position in the vector $\boldsymbol{X}^{(q)}$, and *ii*) able to score a variable number of items.

In this paper, we explore ranking models based on *bivariate scoring functions*. Formally, a ranking model Φ_B based on a bivariate scoring function $\phi_B : \mathbb{R}^{2,m} \to \mathbb{R}^2$ applies ϕ_B to all possible pair of items, then it aggregates all pairwise scores using an aggregation function $f^{(q)}$:

$$\Phi_B(\boldsymbol{X}^{(q)}) = f^{(q)}\left(\phi_B(\boldsymbol{X}_1^{(q)}, \boldsymbol{X}_2^{(q)}), \ldots, \phi_B(\boldsymbol{X}_{n_q-1}^{(q)}, \boldsymbol{X}_{n_q}^{(q)})\right). \tag{3}$$

The definition of bivariate scoring functions requires to answer three questions. The *first* question regards how to learn a bivariate ranking model that is invariant to permutations. Indeed, this is a strong requirement as the model must be invariant to permutations to avoid being dependent on the order of the items given as input. The *second* question regards how to rank the items starting from the pairwise scores. For instance, if the bivariate ranking model assesses who wins each comparison, we can either build a rank of the items that minimizes the number of mis-ordered pairs [29], or we can rank the items according to the number of comparisons won by each item [37]. The *third* question regards how to learn bivariate scoring functions that optimize the ranking. For example, a pairwise model optimizing a classification loss function, e.g., cross entropy, is very accurate in deciding the outcomes of the comparisons but inaccurate in assigning higher scores to the most relevant items. Indeed, classification loss functions focus on all pairs equally, while not all pairs affect the final ranking in the same way, e.g., pairs of relevant items need more effort than pairs of non-relevant items.

In the following section, we answer the three questions above by proposing a novel LtR framework for learning bivariate scoring functions for ranking.

4 The lambda bivariate framework

We now present the Lambda Bivariate (LB) framework, a new Learning-to-Rank framework for learning bivariate scoring functions using gradient boosting trees. As previously introduced, a bivariate scoring function $\phi_B : \mathbb{R}^{2,\cdot} \to \mathbb{R}^2$ score pairs of items and produce pairs of scores. Let $\boldsymbol{x}_{i,j}^{(q)}$ be the vectorial representation, of size m_{pair}, of the pair of items $\langle i, j \rangle$ of the query

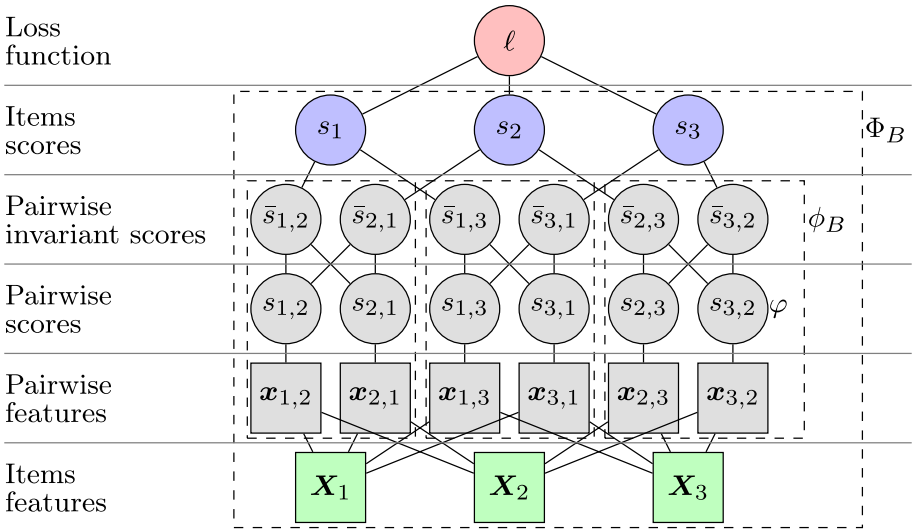

Fig. 1 An instantiation of the Lambda Bivariate framework on a list of 3 items. Bottom-up, all 6 permutations of 2 items are (*i*) fed to the gradient boosting trees model φ, (*ii*) paired and made invariant to permutations [output of the bivariate scoring function ϕ_B, Eq. (4)], and (*iii*) aggregated to form the per-item scores [output of the ranking model Φ_B, Eq. (5)]. The model φ is thus optimized using the derivatives of the loss function ℓ, defined in terms of the per-item scores, with respect to all pairwise scores $s_{i,j}$ [Eqs. (6) and (7)]. Symbols φ, ϕ_B and Φ_B are in correspondence of the outputs

q. Without loss of generality, in the following discussion we drop the superscript q from the notation. In addition, we do not assume any specific loss function as the LB framework can be instantiated with any loss function. Our interest in gradient boosting trees stems from the fact that they achieve state-of-the-art performance on public Learning-to-Rank datasets [9, 35]. Therefore, we want to rely on generic yet powerful machine learning algorithms to learn bivariate scoring functions. Nevertheless, the LB framework could be employed also with generic gradient-based models.

To formally define our ranking model, we need to: (*i*) learn a bivariate scoring function ϕ_B, invariant to permutations, using a gradient boosting trees model φ, (*ii*) define an aggregation function f aggregating all pairwise scores into per-item scores, and (*iii*) describe the optimization process to learn the model φ. Figure 1 shows all components of the LB framework and their interactions on a list of three items. The components are described in the next subsections.

4.1 Invariance to permutations

The invariance to permutations property aims to guarantee the independence of the scores of the items from their input order, i.e., the order in which the items are presented does not affect the final scores and, thus, the final ranking. While learning bivariate scoring functions respecting this property can actually be achieved using neural networks with a "symmetric" architecture, e.g., attention layers [5], we need to define how to learn them with a gradient boosting trees model.

Let $\varphi : \mathbb{R}^{m_{pair}} \rightarrow \mathbb{R}$ be the gradient boosting trees model used to implement the bivariate scoring function ϕ_B. φ assigns a single score $s_{i,j} = \varphi(\boldsymbol{x}_{i,j})$ to any pair of items $\langle i, j \rangle$. In general, $s_{i,j}$ cannot be expressed as a function of $s_{j,i}$, thus we cannot infer one score from the other. For instance, if we consider a binary classifier over pairs of items estimating whether the first item is more relevant than the second one, the output $s_{i,j}$ is generally different from $1 - s_{j,i}$. By using only one of the outputs to infer the other one, e.g., $s_{0,1}$ to infer $s_{1,0} \approx 1 - s_{0,1}$, we may assign different scores to the two pairs $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ if the input order of the two items is different. As a consequence, we exploit both $s_{i,j}$ and $s_{j,i}$ at the same time to guarantee the invariance to permutations property, which in turn implies that we must score all pairs $\langle i, j \rangle$ and $\langle j, i \rangle$ to implement the independence from the input order of the two items. In detail, we associate to each pair of items $\langle i, j \rangle$ the pairwise invariant score $\bar{s}_{i,j} = s_{i,j} - s_{j,i}$ to guarantee the property and we define ϕ_B as follows:

$$\phi_B(\boldsymbol{X}_i, \boldsymbol{X}_j) = \begin{bmatrix} \bar{s}_{i,j} \\ \bar{s}_{j,i} \end{bmatrix} = \begin{bmatrix} s_{i,j} - s_{j,i} \\ s_{j,i} - s_{i,j} \end{bmatrix} \tag{4}$$

Note that this is not the only possible solution that guarantees the invariance property. However, this is the one we selected to model also the symmetry of the scores $\bar{s}_{i,j}$. Thus, $\bar{s}_{i,j} = -\bar{s}_{j,i}$ always holds independently of the algorithm used to learn the model φ.

Figure 1 depicts the above dependence relation between the outcomes of the bivariate scoring function ϕ_B and the estimations of the gradient boosting trees model φ given the pair representations $\boldsymbol{x}_{i,j}$. In detail, the pair representations $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}_{j,i}$ (gray squares) are built from the item representations $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ (green squares). The pair representations are fed, separately, to the gradient boosting tree model φ, which produces two scores $s_{i,j}$ and $s_{j,i}$ for the two pairs (gray circles). Then, the bivariate scoring function ϕ_B (dashed rectangles) makes these pairwise scores invariant to permutations and produces the pairwise invariant scores $\bar{s}_{i,j}$ and $\bar{s}_{j,i}$.

4.2 Aggregation function

The aggregation function aims to produce a final score for any item aggregating all pairwise invariant scores provided by the bivariate scoring function ϕ_B. Therefore, to outline the ranking model Φ_B, which assigns a ranking score to all items starting from the pairwise invariant scores provided by the bivariate scoring function ϕ_B, we need to define the aggregation function f. In fact, the aggregation function is in charge of producing the vector of the item scores (violet circles in Fig. 1) by aggregating the pairwise invariant scores $\bar{s}_{i,j}$ of all items.

As discussed in the previous section, several alternative definitions are possible [29, 37, 38]. However, not all alternatives fit our needs as we aim to learn a gradient boosting trees model φ that optimizes the ranking of the items by looking at the scores $\boldsymbol{s}$ of all items. As a consequence, to learn the model φ, the aggregation function must be differentiable as we need to derive the error with respect to the model predictions. Moreover, as the number of items often varies across queries, we also want that the scores $\boldsymbol{s}$ to be independent of the number of pairs of items in a query so to make it uniform across different queries.

For this purpose, we define an aggregation function f that defines the score of each item i to be the average pairwise invariant score of all pairs $\langle i, j \rangle$ of the same query:

$$\boldsymbol{s}_i = \frac{1}{n-1} \sum_{j \in q/i} \bar{s}_{i,j} = \frac{1}{n-1} \sum_{j \in q/i} s_{i,j} - s_{j,i}. \tag{5}$$

4.3 Optimization process

The optimization process aims to learn the bivariate scoring function ϕ_B – through gradient-boosting trees working on pair of items. The target of the optimization is to maximize the quality of the final ranking. To this end, we now show that, given an arbitrary loss function ℓ, we can optimize it in a pairwise fashion by exploiting a gradient boosting trees model φ. As most of the existing LtR algorithms exploit univariate scoring functions, in the following we assume to know the derivatives of the loss function ℓ with respect to the per-item scores $\boldsymbol{s}_i$. We can thus employ Gradient Descent [23] or Expectation Maximization [39] techniques to learn the model φ.

To optimize the model φ, we need to compute the derivative of the loss function ℓ with respect to the outputs $s_{i,j}$ of the model φ. We can express these derivatives in terms of the previous known derivatives, i.e., $\partial \ell / \partial \boldsymbol{s}_i$:

$$\frac{\partial \ell}{\partial s_{i,j}} = \sum_{k \in q} \frac{\partial \ell}{\partial \boldsymbol{s}_k} \frac{\partial \boldsymbol{s}_k}{\partial s_{i,j}} = \sum_{k \in \{i,j\}} \frac{\partial \ell}{\partial \boldsymbol{s}_k} \frac{\partial \boldsymbol{s}_k}{\partial s_{i,j}} = \frac{1}{n-1} \left(\frac{\partial \ell}{\partial \boldsymbol{s}_i} - \frac{\partial \ell}{\partial \boldsymbol{s}_j} \right), \tag{6}$$

where the second equality follows from the fact that the score $s_{i,j}$ affects only the pairwise invariant scores $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ of the items i and j. The equation above shows that the optimization of the pairwise score $s_{i,j}$ in the LB framework naturally follows from the optimization of the pairwise invariant scores $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$.

As many recent implementations of gradient boosting trees, e.g., LightGBM [40], rely on quasi newton methods for optimizing non-linear functions [41], we also need to compute the second order derivatives of the loss function, i.e.,

$$\begin{aligned}\frac{\partial^2 \ell}{\partial s_{i,j}^2} &= \sum_{k \in q} \frac{\partial}{\partial \boldsymbol{s}_k}\left(\frac{\partial \ell}{\partial s_{i,j}}\right)\frac{\partial \boldsymbol{s}_k}{\partial s_{i,j}} = \sum_{k \in \{i,j\}} \frac{\partial}{\partial \boldsymbol{s}_k}\left(\frac{\partial \ell}{\partial s_{i,j}}\right)\frac{\partial \boldsymbol{s}_k}{\partial s_{i,j}} \\ &= \frac{1}{n-1}\sum_{k \in \{i,j\}} \frac{\partial}{\partial \boldsymbol{s}_k}\left(\frac{\partial \ell}{\partial \boldsymbol{s}_i} - \frac{\partial \ell}{\partial \boldsymbol{s}_j}\right)\frac{\partial \boldsymbol{s}_k}{\partial s_{i,j}} \\ &= \frac{1}{n-1}\left(\frac{\partial^2 \ell}{\partial \boldsymbol{s}_i^2}\frac{\partial \boldsymbol{s}_i}{\partial s_{i,j}} + \frac{\partial^2 \ell}{\partial \boldsymbol{s}_i \partial \boldsymbol{s}_j}\left(\frac{\partial \boldsymbol{s}_j}{\partial s_{i,j}} - \frac{\partial \boldsymbol{s}_i}{\partial s_{i,j}}\right) - \frac{\partial^2 \ell}{\partial \boldsymbol{s}_j^2}\frac{\partial \boldsymbol{s}_j}{\partial s_{i,j}}\right) \\ &= \frac{1}{(n-1)^2}\left(\frac{\partial^2 \ell}{\partial \boldsymbol{s}_i^2} - 2\frac{\partial^2 \ell}{\partial \boldsymbol{s}_i \partial \boldsymbol{s}_j} - \frac{\partial^2 \ell}{\partial \boldsymbol{s}_j^2}\right).\end{aligned} \tag{7}$$

Thanks to previous derivatives we can thus optimize existing loss functions using gradient boosting trees to solve ranking problems in a pairwise fashion.

4.4 Bivariate λ MART

We now describe the application of the LB framework to the λ Rank loss, which results in a bivariate version of λ MART [9] that we call Bi- λ MART.

λ MART combines the gradient boosting trees model [8] and the λ Rank loss [9, 11] to optimize the NDCG ranking metric [42]. The output of a gradient boosting trees model is a linear combination of the outputs of a set of regression trees. It is a boosting algorithm optimizing a general loss function and it can be seen as an algorithm performing gradient descent using regression trees. Trees are built incrementally, i.e., one by one, using gradient descent so that each new tree fits the derivatives of the loss function with respect to the scores achieved by the previous trees. By doing so, each new regression tree models a descent step decreasing the loss achieved by the previous trees. The second component of λ MART is the λ Rank loss [11, 36]. λ Rank is a listwise loss function defined as a weighted logistic function over the pairs of items of a query so to improve the NDCG metric. NDCG is a widely used performance metric which exploits multi-graded relevances [42]. It is defined as the ratio between DCG and Ideal DCG, where DCG is the Discounted Cumulative Gain of a ranked list of results, and Ideal DCG is the DCG of the same list sorted by relevance label. In detail, the loss function aims to separate all pair of items having different relevance by weighting each pair according to the delta of NDCG caused by a swap of the two items in the ranking:

$$\ell_{\lambda\text{RANK}}(y, s) = -\sum_{y_i > y_j} \frac{\Delta\text{DCG}(i,j)}{\text{Ideal DCG}} \log\left(\frac{1}{1 + e^{-(s_i - s_j)}}\right). \tag{8}$$

Since gradient boosting trees algorithms exploit derivatives to train tree-based models and λ Rank works by specifying the derivatives at any point during training, the combination of the two algorithms results in λ MART [9]. Similarly, the application of LB to the λ Rank loss exploits the two derivatives introduced in previous subsection, i.e., eqs. (6) and (7). This leads to the definition of Bi- λ MART, a pairwise version of λ MART. Note that the above derivatives now reflect the application of λ Rank in a pairwise scenario, as illustrated in Fig. 1. Such derivatives are the result of the three main components introduced in the previous section, i.e., the invariance to permutations property, the aggregation function, and the optimization process. Note that, while Bi- λ MART is the result of the application of LB to λ Rank, the framework is general and it can be applied also to other loss functions, as shown in Sect. 6.2 where we apply LB to the Softmax loss.

5 Experimental setup

In this section, we provide a detailed description of our experimental setup. In details, we describe the Learning-to-Rank datasets employed, the ranking architecture used for the assessment, the state-of-the-art competitors used for comparisons, the hyper-parameter tuning, and the feature sets used to represent the items.

Table 1 Statistics of all datasets for each ranking stage

Dataset		Queries	Stage 1		Stage 2		Stage 2 pairwise	
			Docs	Feat.	Docs	Feat.	Pairs	Feat.
Web10K	Train	6000	723,412	136	117,686	685	2,220,174	1507
	Valid	2000	235,259		39,273		740,838	
	Test	2000	241,521		39,205		739,312	
Web30K	Train	18,919	2,270,296	136	369,804	685	6,970,110	1507
	Valid	6306	747,218		123,415		2,328,250	
	Test	6306	753,611		123,519		2,329,648	
Yahoo!	Train	19,944	473,134	699	294,365	3500	4,754,252	7700
	Valid	2944	71,083		43,802		704,534	
	Test	6983	165,660		103,395		1,672,484	

5.1 Datasets

We conduct experiments on three public datasets for Learning-to-Rank, i.e., the two LETOR datasets [43] and the "Yahoo! Learning to Rank Challenge" [35] dataset. The three datasets have been created for assessing LtR algorithms in a Web search scenario. The datasets consist of query-document pairs. Each query-document pair is labeled by humans with a graded relevance ranging from 0 to 4, where larger values of the label indicate a higher relevance of the document with respect to the query. The two LETOR datasets, i.e., Web10K and Web30K, have been released by Microsoft [43] and contain 10,000 and 30,000 queries, respectively. They both provide an average of 120 documents per query and each query-document pair is represented by 136 features. Moreover, they both come divided in five folds, where each fold is divided in three partitions for training, validation, and test. In our experiments, we report the results achieved on the first fold of the two datasets. The third dataset we employ is the Yahoo! LTRC [35]. It contains 30,000 queries, with an average of 24 documents per query and each query-document pair is represented by 699 features. The Yahoo! dataset comes divided in two sets, each one splitted in three partitions for training, validation, and test, respectively. In our experiments, we report the results achieved on the first set of this dataset.

5.2 Two-stage ranking architecture

We evaluate the performance of LB and state-of-the-art competitors in a standard two-stage architecture of a Web search engine query processor [44]. The pipeline consists of a first stage that is in charge of producing a list of candidate results that are then re-ranked by a second stage that produces the final list of results to return to the user. Similarly to previous work [44], we experiment our proposals in the second-stage of the query processor by re-ranking the top-20 documents provided by a first-stage ranker. We measure the performance of the methods tested by reporting the mean Normalized Discounted Cumulative Gain (NDCG) [42] at rank k, i.e., NDCG@k.

The performance of the second stage strictly depends on the model used in the first stage to select the top-20 results. To this end, we define a robust methodology based on cross validation to build the second-stage datasets so to break any dependency due to stacking the first and second stage rankers. A similar approach has been used by previous work using public LtR datasets [2, 7]. For each dataset, we perform bayesian hyper-parameter optimization to find the best set of hyper-parameters of the first stage ranker that maximize NDCG@20 on the validation set[2]. In detail, we randomly split the queries of the training, validation and test partitions into 10 folds to train and validate 10 λ MART models using the same set of hyper-parameters. During the i-th iteration of the cross validation, we train a λ MART model on all except the i-th fold of the training partition and by using all except the i-th fold of the validation partition for early stopping. We then evaluate the learned model by using all queries of the i-th fold of the training, validation and test partitions. Therefore, each λ MART model act as first-stage ranker for a separate fold of the dataset, e.g., the model trained on all except the i-th fold of the training partition rank the queries of the i-th fold of the three partitions, i.e., training, validation and test. For each query, we use the top-20 documents provided by the first-stage ranker as second-stage dataset, which is then used to evaluate LB and all state-of-the-art competitors. Table 1 reports the main statistics of the three datasets for

[2] We employ the Python library `hyperopt` [45], https://github.com/hyperopt/hyperopt.

the different ranking stages. Note that the cross validation is used only to create the second stage datasets. Then, the second stage rankers are trained using the new training, validation, and test partitions (which contains the same queries of the first stage partitions, but with fewer results per query) for training, model selection, and evaluation, respectively.

5.3 Competitors

We perform a comparison of the performance of LB against several state-of-the-art competitors for Learning-to-Rank.

- **DNN** [46]: A deep feed-forward neural network model to learn univariate scoring functions, which employs batch normalization and dropout after each layer.
- **GSF** [4]: A groupwise scoring model based on neural networks that ranks documents based on group comparisons. In our experiments, we report the results for groups of size 2 and 20, i.e., GSF(2) and GSF(20), which are equivalent to bivariate scoring functions and multivariate scoring functions (scoring all results of a query together), respectively.
- **ATTN-DIN** [6]: A neural model using self-attention to encode the entire list followed by multiple feed-forward layers to jointly score all results.
- **SETRANK** [7]: A neural models based on self-attention to jointly score the entire list of results. Differently from ATTN-DIN, SETRANK uses a variant of the attention layers and does not employ feed-forward layers on top of that. The authors of SETRANK also proposed a version of the model for re-ranking, which extends the set of features of each result with the positional encoding of the first stage rank. We did not observe any real advantage in using it, mainly because, in our setting, the score and the rank in the first stage are already in the feature set (see Sect. 5.5), so we do not report the results of this version of SETRANK.
- **GBT** [8]: A gradient boosting trees model learning univariate scoring functions, which employs first and second order derivatives to build the trees.

We evaluate LB and the competitors above by using the λ Rank and the Softmax loss. In particular, GSF , attn-DIN , and SetRank model multivariate scoring functions using neural networks and already exploit the three components that we are using in our framework for gradient boosting trees. For what regards GBT, its instantiation with the λ Rank loss is known as λ MART [9, 10]. Moreover, we call GBT_Softmax its instantiation with the Softmax loss. We also experiment a variation of the λ MART algorithm – we call it λ MART* – that takes into account the bias introduced by working in a two-stage ranking architecture. The bias origins from the fact that, when λ MART is trained on the second stage of the architecture, it only exploits the training data of the current stage, i.e., a subset of the available data for each query. Therefore, the learning algorithm may ignore some of the relevant results available for some queries when computing the Ideal DCG used to normalize the weights associated to each query in equation (8). λ MART* fixes this behavior using the Ideal DCG of the first stage which considers all results.

We employed the original implementations of DNN, GSF, and ATTN-DIN available in the TFRanking[3] library [46], while for SETRANK we used the original implementation[4] released by the authors, which is based on the TensorFlow library [47] as the previous ones. All gradient boosting trees models, including LB, are implemented in the LightGBM [40] library.[5] Similarly to λ MART*, our implementation of BI- λ MART uses the Ideal DCG of the first stage during training as λ MART*.

5.4 Hyper-parameter tuning

The hyper-parameters of all models are tuned to optimal on the validation set according to the NDCG@5 metric. The neural network models use up to 150,000 epochs of training and employ early stopping to stop the training when there is no improvement on the validation set for 10,000 consecutive epochs. We used a batch size of 32 and the Adagrad [45] optimizer. The tuning included learning rate, dropout, number of layers, their size, and number of attention layers/heads/size. The gradient boosting trees models exploit up to 2000 trees and employ early stopping to stop the training when there is no improvement on the validation set for 30 consecutive iterations. The tuning included learning rate, minimum

[3] https://github.com/tensorflow/ranking.
[4] https://github.com/pl8787/SetRank.
[5] To ease the reproducibility of the results we will release our implementations of λ MART* and BI- λ MART upon acceptance of the paper.

Table 2 NDCG@k of tree-based models optimizing the λ RANK loss on the three datasets

Dataset	Stage - Model	NDCG				
		@1	@3	@5	@10	@20
WEB10K	1 - λ MART	0.4848	0.4805	0.4874	0.5082	0.5383
	2 - λ MART	0.4985	0.4901^{a}	0.4968^{a}	0.5143^{a}	0.5417
	2 - λ MART*	0.5061^{a}	0.4927^{a}	0.4975^{a}	0.5155^{a}	0.5428^{a}
	2 - BI- λ MART	**0.5202**^{abc}	**0.5014**^{abc}	**0.5029**^{abc}	**0.5195**^{abc}	**0.5465**^{abc}
WEB30K	1 - λ MART	0.5253	0.5058	0.5110	0.5289	0.5596
	2 - λ MART	0.5340	0.5194^{a}	0.5244^{a}	0.5409 ^{a}	0.5661^{a}
	2 - λ MART*	0.5369^{a}	0.5217^{a}	0.5248^{a}	0.5414^{a}	0.5666^{a}
	2 - BI- λ MART	**0.5386**^{a}	**0.5272**^{abc}	**0.5291**^{abc}	**0.5449**^{abc}	**0.5690**^{abc}
YAHOO!	1 - λ MART	0.7227	0.7350	0.7549	0.7957	0.8342
	2 - λ MART	0.8451^{a}	0.8390^{a}	0.8421^{a}	0.8546^{a}	0.8834^{a}
	2 - λ MART*	0.8449^{a}	0.8382^{a}	0.8409^{a}	0.8541^{a}	0.8831^{a}
	2 - BI- λ MART	**0.8494**^{a}	**0.8430**^{abc}	**0.8466**^{abc}	**0.8593**^{abc}	**0.8857**^{abc}

Best results are highlighted in bold

Superscript letters a,b,c denote statistically significant improvements using a paired t-test ($p < 0.01$) with respect to the first, second, and third row of each dataset, respectively

data in leaf, number of leaves and minimum hessian. We use the python library hyperopt to perform Bayesian Hyper-parameter Optimization [45] to find the best set of hyper-parameters of each model.

5.5 Feature expansion

To allow the fairest possible evaluation of the algorithms that learn univariate scoring functions, i.e., DNN, λ MART, and λ MART*, we expand the feature representation of the items in the second-stage datasets with listwise features. These additional features, built by considering the whole list of results on a per-query basis, allow the three competitors to exploit query-level information. By doing so, we partially overcome their limitations of considering a single item at a time. Indeed, several work show that by adding features that captures contextual information of the entire list of results turns out to improve the ranking performance [44, 48]. We perform this expansion by adding to the representation of each item in the second stage its score achieved in the first stage. This score is computed by using the λ MART model learned with the methodology described in Sect. 5.2. Then, for each feature f (including the first stage score) and query q, we add four new listwise features: (*i*) the mean and (*ii*) the standard deviation of the values assumed by f in the list of q, (*iii*) the rank of each item when ranking the items of q according to f, and (*iv*) the standardization of f according to the values assumed in q. In our experiments, all second-stage models are learnt by using this extended set of features.

For BI- λ MART, we represent each pair of items $\langle i, j \rangle$ with the concatenation of the extended set of listwise features of i, j, and their feature-wise difference. Note that, while the extended set of listwise features of an item can be effectively exploited by univariate scoring functions, the latter set, i.e., the concatenation of the features of the two items of the pair and their feature-wise difference, can be exploited only by bivariate scoring functions as they work on pair of items. The reason behind explicitly adding features modeling feature-wise differences relies in the fact that tree-based approaches do not easily capture the difference between two features. This is due to the learning algorithm that, for each split node, greedily chooses one feature and threshold at a time, thus missing a multi-feature view when deciding the split nodes. The total number of features used by the different approaches on the three datasets is reported in Table 1. In Sect. 6.1, we experimentally show the benefits obtained by using these features by performing an ablation study.

6 Experimental results

We now report the results of our experimental evaluation when using the λ RANK loss [11, 36]. We performed an extensive hyper-parameter tuning process for each model and dataset. Details of hyper-parameters used can be found in Appendix A. We show in Tables 2 and 3 the performance of tree-based and neural-network-based algorithms, respectively,

Table 3 NDCG@*k* of neural network models optimizing the λ RANK loss when applied as second stage rankers to the three datasets

Dataset	Model	NDCG				
		@1	@3	@5	@10	@20
WEB10K	DNN	0.4670	0.4537	0.4590	0.4772	0.5056
	GSF(2)	0.4680	0.4546	0.4587	0.4759	0.5053
	GSF(20)	0.4694	0.4528	0.4569	0.4724	0.5039
	ATTN-DIN	0.4639	0.4537	0.4607	0.4765	0.5053
	SETRANK	0.4786	0.4616	0.4653	0.4810	0.5091
WEB30K	DNN	0.5018	0.4840	0.4880	0.5049	0.5330
	GSF(2)	0.4993	0.4842	0.4881	0.5047	0.5328
	GSF(20)	0.5007	0.4846	0.4895	0.5050	0.5329
	ATTN-DIN	0.5030	0.4898	0.4929	0.5083	0.5356
	SETRANK	0.5037	0.4895	0.4938	0.5095	0.5362
YAHOO!	DNN	0.7965	0.7912	0.7960	0.8119	0.8425
	GSF(2)	0.8010	0.7938	0.7987	0.8137	0.8438
	GSF(20)	0.8032	0.7946	0.7993	0.8138	0.8445
	ATTN-DIN	0.8073	0.7990	0.8026	0.8167	0.8464
	SETRANK	0.8061	0.7975	0.8017	0.8159	0.8459

where the evaluation is done by computing the NDCG@*k* on the test queries of the three datasets, taking into account also the relevant results discarded between the first and second stage of the ranking pipeline.

The first row of each dataset in Table 2 reports the performance of the λ MART algorithm when employed as a single stage ranking pipeline. The effectiveness of the λ MART baseline on the three datasets is comparable with results reported in early work employing LightGBM [49] and better than those employing the RankLib library, e.g., [4, 7]. The reason is that LightGBM provides a much more effective implementation of λ MART than RankLib. The remaining rows of each dataset in Table 2 show the performance of the tested Learning-to-Rank models when employed as a second-stage rankers, in a ranking architecture where the first stage employs a λ MART model providing the list of the first 20 result of each query, as described in Sect. 5.2. First of all, experimental results show that a two-stage ranking pipeline exploiting only λ MART models (second row of each dataset in Table 2) always achieve a higher NDGG@*k* than a single-stage pipeline (first row of each dataset in Table 2). Indeed, the improvement given by the two-stage architecture is statistically significant with respect to the single-stage one most of the times. Our proposed variant λ MART*, which fixes the computation of the Ideal DCG of the λ MART algorithm when employed in the second stage of the ranking pipeline, always outperforms the single-stage λ MART counterpart. Indeed, it outperforms the single-stage λ MART model on all datasets with statistically significant improvements for all tested NDCG cutoffs.

Table 2 also shows that BI- λ MART achieves the best results with statistically significant improvements with respect to all tested competitors in almost all configurations. When considering NDCG@5, i.e., the metric used for training and tuning all models, BI- λ MART always outperform the competitors with statistically significant improvements. In particular, BI- λ MART achieves a NDCG@5 from 3.2% to 12.2% higher than λ MART in the first stage, from 0.5% to 1.2% higher than λ MART in the second stage, and from 0.7% to 1.1% higher than our variant λ MART*. Since all tested models optimize the same λ RANK loss using gradient boosting trees, the experimental results confirm that the greater expressive power of BI- λ MART comes from the use of bivariate scoring functions.

Table 3 reports the performance of the several neural-network-based competitors introduced in the previous section when optimizing the λ RANK loss. Experimental results show that NNs learning multivariate scoring functions are more effective than DNN, which is the only univariate one, on WEB30K and YAHOO!. In details, SETRANK and ATTN-DIN achieve the best results on all datasets among the neural network models for all tested NDCG cutoffs. The improved effectiveness achieved by SETRANK and ATTN-DIN, with respect to all other neural network models, relies in the fact that the attention layers [5] allow them to exploit the interactions among all documents of the list. Indeed, they effectively exploit the information available in the list of documents to achieve a better ranking. Nevertheless, experimental results show that tree-based models outperform neural network models by a large margin.

To conclude, results show that BI- λ MART, which instantiate LB using the λ RANK loss, outperforms all state-of-the-art tree-based and neural-network-based competitors optimizing the same loss function, with statistically significant improvements for almost all tested NDCG cutoffs and datasets.

Table 4 NDCG@k of bivariate models optimizing the λ RANK loss when trained without or with the additional set of feature-wise difference features and applied as second stage rankers to WEB30K

Model	NDCG				
	@1	@3	@5	@10	@20
GSF(2) without	0.4993	0.4842	0.4881	0.5047	0.5328
GSF(2) with	0.5007	0.4846	0.4895	0.5050	0.5329
BI- λ MART without	0.5390	0.5237	0.5256	0.5409	0.5669
BI- λ MART with	0.5386	0.5272	0.5291	0.5449	0.5690

6.1 Role of feature-wise difference features

In Sect. 5.5, we described the set of expanded features employed to allow the fairest evaluation possible against the models learning univariate scoring functions, i.e., λ MART, λ MART*, and DNN. We also detailed the additional set of features used by BI- λ MART. This latter model exploits the extended sets of features of items i and j plus their feature-wise difference. While univariate scoring functions and attention-based multivariate scoring functions can access only the features of one item at a time when scoring, these additional features can be exploited by bivariate scoring functions, i.e., GSF(2) and BI- λ MART, as they score a pair of items at a time. We now assess the importance of the feature-wise difference features in the performance of BI- λ MART. We aim to show that, the feature-wise differences are marginally relevant for neural-network-based models and essential for tree-based models. The reason is that the greedy learning algorithm used to build the trees lacks a multi-feature view when building the split-nodes as each split node is decided greedily by looking at one feature and threshold at a time.

Table 4 reports the results of this investigation, where we compare, on the WEB30K dataset, the performance of both GSF and BI- λ MART when trained without or with the additional set of feature-wise difference features. Experiments show that GSF(2) slightly benefits from the feature-wise difference features as its performance are always very close to its counterpart that does not exploit them (+0.29% in terms of NDCG@5). The small difference between the two versions of GSF(2) is more evident at small cutoffs of the NDCG metric while it thins for larger values of the cutoff.

We also perform the same investigation for BI- λ MART. Experiments show that it achieves much better results in terms of NDCG@k than its counterpart that does not employ the feature-wise difference features (+0.67% in terms of NDCG@5). Moreover, without these features, the performance of BI- λ MART are very close to those of λ MART*, which exploits univariate scoring functions. In general, results show that the set of feature-wise difference features are crucial to the effectiveness of BI- λ MART.

Experimental results confirm that deep neural network models are already able to capture this type of multi-feature aspects, while tree-based models are not. To conclude, experiments support our initial hypothesis that feature-wise difference features provide an essential representation of the pairs of items for tree-based models.

6.2 Assessing LB with the SOFTMAX loss

To show the generality of the LB framework, here we present an analysis of the performance of neural networks and tree-based models when using the SOFTMAX loss [12]. This is a popular and effective listwise loss for neural networks [4, 6, 13] and is defined as $\ell_{\text{SOFTMAX}}(y, s) = -\sum_i y_i \log \frac{e^{s_i}}{\sum_j e^{s_j}}$. We perform this analysis on the second stage of the WEB30K dataset. The hyper-parameters of all models have been tuned using this new loss.

We report the results of the comparison in Table 5. First of all, we note that all neural network models achieve a better performance using the SOFTMAX loss than using the λ RANK loss (Table 3), with improvements up to 1.27% in terms of NDCG@5. As in the previous analysis, SETRANK and ATTN-DIN achieve the best results among the neural networks for all tested NDCG cutoffs thus confirming the effectiveness of the attention mechanism for ranking.

We implemented the SOFTMAX loss in LightGBM to assess the performance of the tree-based competitor based on univariate scoring functions. The GBT model optimizing the SOFTMAX loss – we call it $\text{GBT}_{\text{SOFTMAX}}$– outperforms all neural network models by a large margin with statistically significant improvements. However, we highlight that, GBT achieves a lower performance when optimizing SOFTMAX than when optimizing the λ RANK loss, i.e., –0.27% in terms of NDCG@5 w.r.t. λ MART (see Table 2).

Table 5 NDCG@*k* of models optimizing the SOFTMAX loss when applied as second stage rankers to WEB30K

Model	NDCG				
	@1	@3	@5	@10	@20
DNN	0.5021	0.4868	0.4910	0.5078	0.5339
GSF(2)	0.4989	0.4873	0.4916	0.5082	0.5341
GSF(20)	0.5016	0.4883	0.4928	0.5092	0.5348
ATTN-DIN	0.5049	0.4933	0.4991	0.5127	0.5377
SETRANK	0.5055	0.4963	0.4998	0.5145	0.5385
$\text{GBT}_{\text{SOFTMAX}}$	0.5320†	0.5181†	0.5230†	0.5391†	0.5649†
$\text{BI-GBT}_{\text{SOFTMAX}}$	**0.5389**$^{\dagger *}$	**0.5267**$^{\dagger *}$	**0.5299**$^{\dagger *}$	**0.5451**$^{\dagger *}$	**0.5682**$^{\dagger *}$

Superscripts $\dagger$ and $*$ denote statistically significant improvements using a paired *t*-test ($p < 0.01$) with respect to neural network competitors (DNN, GSF(2), GSF(20), ATTN-DIN, SetRank) and tree-based competitors ($\text{GBT}_{\text{SOFTMAX}}$), respectively

In this experimental evaluation, we call $\text{BI-GBT}_{\text{SOFTMAX}}$ the application of the LB framework to the SOFTMAX loss. $\text{BI-GBT}_{\text{SOFTMAX}}$ achieves better results than $\text{GBT}_{\text{SOFTMAX}}$ for all tested NDCG cutoffs, showing the effectiveness of the bivariate scoring functions for ranking. In particular, it improves over $\text{GBT}_{\text{SOFTMAX}}$ by 1.3% of NDCG@5 and all the improvements reported are statistically significant. Moreover, $\text{BI-GBT}_{\text{SOFTMAX}}$ achieves a slightly better performance than BI- λ MART, showing that it is more robust to changes of the loss function with respect to the univariate counterpart GBT, which performs worse with SOFTMAX than with λ RANK. Therefore, λ RANK is a strong loss function for ranking when using univariate scoring functions based on gradient boosting trees. However, the SOFTMAX loss can further improve the effectiveness of the bivariate scoring functions counterpart. $\text{BI-GBT}_{\text{SOFTMAX}}$ outperforms all neural network models based on SOFTMAX by a large margin. In particular, it improves over the best-performing neural network, i.e., SETRANK, by 6.0% in terms of NDCG@5.

Results thus show that the proposed LB framework outperforms all state-of-the-art neural-network-based and tree-based competitors using different loss functions.

7 Conclusions and future work

Our work stems from the fact that most of the state-of-the-art Learning-to-Rank algorithms learn univariate scoring functions that score each item independently, i.e., without taking into account the other items in the list. To overcome this limitation, we presented a novel Learning-to-Rank framework – the Lambda Bivariate framework – to learn ranking models based on bivariate scoring functions, i.e., functions scoring pairs of items so to effectively capture the dependencies between items in the list. We showed that LB can be used with any loss function to train gradient boosting trees. Moreover, we discussed an application of LB to the λ RANK loss, which resulted in BI- λ MART, a pairwise version of λ MART.

We proposed a comprehensive experimental evaluation of BI- λ MART and several state-of-the-art tree-based and neural-network-based competitors over three public Learning-to-Rank datasets. Results show that BI- λ MART outperforms neural network models by a significant margin and also outperforms λ MART with a relative improvement in terms of NDCG@5 ranging from 0.5% to 1.2%. Moreover, we also experimented LB with the softmax loss function and showed that it outperforms in terms of NDCG@5 the tree-based competitor by 1.3% and the best-performing neural-network-based competitor by 6.0%. As future work, we aim to extend the LB framework to learn multivariate scoring functions for ranking to jointly score multiple items at a time.

Author contributions All authors contributed equally to the design of the *Lambda Bivariate* framework and to the experimental evaluation presented in this manuscript. All authors wrote the first draft of the manuscript and approved its final version.

Funding This work was supported by: MUR-PRIN 2022 "Algorithmic Problems and Machine Learning"; PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI" funded by the European Union (EU) under the NextGeneration EU programme; the EU's Horizon Europe research and innovation programme EFRA (Grant Agreement Number 101093026). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the EU or European Commission-EU. Neither the EU nor the granting authority can be held responsible for them.

Data availability The experimental evaluation has been conducted by employing three public datasets for Learning-to-Rank, i.e., the two LETOR datasets [43] and the "Yahoo! Learning to Rank Challenge" [35] dataset.

Declarations

Ethics approval and consent to participate Not applicable.

Competing interests The authors declare a potential Conflict of interest (COI) with Michael Bendersky and Nicola Tonellotto (previous co-authors).

Appendix A

A Experimental details

To ease the reproducibility of the results, tables below report the hyper-parameters used for all models of Tables 2 and 3. The methodology used to tune the hyper-parameters of all second-stage models is described in Sect. 5.4. Tables 6 and 7 reports the non-default parameters of neural network models, while Table 8 reports the ones of tree-based models.

Table 6 Hyper-parameters of TensorFlow neural network models

Dataset	Model	Learning Rate	Hidden layer Sizes	Dropout
Web10K	DNN	0.010	1024,256,64	0.5
	GSF(2)	0.005	1024,512,256,128	0.5
	GSF(20)	0.050	512,256,128,64	0.50
	attn-DIN	0.005	512,256,128,64	0.25
	SetRank	0.050	–	–
Web30K	DNN	0.050	512,256,128,64	0.5
	GSF(2)	0.010	4096,1024,256,64	0.5
	GSF(20)	0.050	1024,512,256,128	0.5
	attn-DIN	0.050	512,128,32	0.5
	SetRank	0.050	–	–
Yahoo!	DNN	0.050	1024,512,256	0.5
	GSF(2)	0.050	2048,1024,512,256	0.5
	GSF(20)	0.050	1024,512,256	0.25
	attn-DIN	0.050	2048,1024,512,256	0.25
	SetRank	0.050	–	–

Table 7 Hyper-parameters of the attention layers of attention-based neural network models

Dataset	Model	Num layers	Num heads	Size	Dropout
Web10K	attn-DIN	4	6	256	0.25
	SetRank	5	2	256	0.50
Web30K	attn-DIN	1	4	256	0.50
	SetRank	4	2	256	0.50
Yahoo!	attn-DIN	6	1	128	0.25
	SetRank	4	2	512	0.25

Table 8 Hyper-parameters of LightGBM tree-based models

Dataset	Stage - Model	Learning rate	Min data	Num leaves	Min hessian
WEB10K	1 - λ MART	0.050	200	115	0
	2 - λ MART	0.080	2680	230	20
	2 - λ MART*	0.045	1910	245	80
	2 - BI- λ MART	0.005	5640	300	0
WEB30K	1 - λ MART	0.080	190	365	195
	2 - λ MART	0.040	460	500	240
	2 - λ MART*	0.045	40	75	50
	2 - BI- λ MART	0.001	7150	1530	0
YAHOO!	1 - λ MART	0.050	500	425	10
	2 - λ MART	0.105	170	445	170
	2 - λ MART*	0.075	170	490	5
	2 - BI- λ MART	0.005	1790	1100	0

References

1. Liu T. Learning to rank for information retrieval. Found Trends Inf Retr. 2009;3(3):225–331. https://doi.org/10.1561/1500000016.
2. Ai Q, Bi K, Guo J, Croft WB. Learning a deep listwise context model for ranking refinement. In: Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2018. p. 135–44. https://doi.org/10.1145/3209978.3209985.
3. Bello I, Kulkarni S, Jain S, Boutilier C, Chi EH, Eban E, Luo X, Mackey A, Meshi O. Seq2Slate: re-ranking and slate optimization with RNNs. CoRR. abs/1810.02019. 2018
4. Ai Q, Wang X, Bruch S, Golbandi N, Bendersky M, Najork M. Learning groupwise multivariate scoring functions using deep neural networks. In: Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR), ACM; 2019. p. 85–92. https://doi.org/10.1145/3341981.3344218.
5. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Lu, Polosukhin I. Attention is all you need. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Curran Associates, Inc.; 2017;30:5998–6008.
6. Pasumarthi RK, Zhuang H, Wang X, Bendersky M, Najork M. Permutation equivariant document interaction network for neural learning to rank. In: Proceedings of the 2020 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR), ACM; 2020. p. 145–8.
7. Pang L, Xu J, Ai Q, Lan Y, Cheng X, Wen J. SetRank: Learning a permutation-invariant ranking model for information retrieval. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2020. p. 499–508. https://doi.org/10.1145/3397271.3401104.
8. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;29:1189–232.
9. Burges CJ. From RankNet to LambdaRank to LambdaMART: an overview. Learning. 2010;11(23–581):81.
10. Wu Q, Burges CJC, Svore KM, Gao J. Adapting boosting for information retrieval measures. Inf Retr. 2010;13(3):254–70. https://doi.org/10.1007/s10791-009-9112-1.
11. Burges CJC, Ragno R, Le QV. Learning to rank with nonsmooth cost functions. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), MIT Press; 2006. p. 193–200.
12. Cao Z, Qin T, Liu T, Tsai M, Li H. Learning to rank: from pairwise approach to listwise approach. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML). ACM; 2007. https://doi.org/10.1145/1273496.1273513.
13. Bruch S, Wang X, Bendersky M, Najork M. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In: Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR), ACM; 2019. p. 75–8. https://doi.org/10.1145/3341981.3344221.
14. Capannini G, Lucchese C, Nardini FM, Orlando S, Perego R, Tonellotto N. Quality versus efficiency in document scoring with learning-to-rank models. Inf Process Manag. 2016;52(6):1161–77. https://doi.org/10.1016/j.ipm.2016.05.004.
15. Tax N, Bockting S, Hiemstra D. A cross-benchmark comparison of 87 learning to rank methods. Inf Process Manag. 2015;51(6):757–72.
16. Macdonald C, Santos RL, Ounis I. The whens and hows of learning to rank for web search. Inf Retr. 2013;16(5):584–628.
17. Li P, Burges CJC, Wu Q. McRank: learning to rank using multiple classification and gradient boosting. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), 2007. p. 897–904.
18. Cossock D, Zhang T. Subset ranking using regression. In: Proceedings of the 19th Annual Conference on Learning Theory (COLT), vol. 4005. Springer; 2006. p. 605–19. https://doi.org/10.1007/11776420_44.
19. Shashua A, Levin A. Ranking with large margin principle: two approaches. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), MIT Press; 2002. 937–44.
20. Crammer K, Singer Y. Pranking with ranking. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), MIT Press; 2001. p. 641–7.
21. Zheng Z, Zha H, Zhang T, Chapelle O, Chen K, Sun G. A general boosting method and its application to learning ranking functions for web search. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Curran Associates, Inc.; 2007. p. 1697–704.

22. Cao Y, Xu J, Liu T, Li H, Huang Y, Hon H. Adapting ranking SVM to document retrieval. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2006. p. 186–93. https://doi.org/10.1145/1148170.1148205.
23. Burges CJC, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender GN. Learning to rank using gradient descent. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML). ACM; 2005. https://doi.org/10.1145/1102351.1102363.
24. Freund Y, Iyer RD, Schapire RE, Singer Y. An efficient boosting algorithm for combining preferences. J Mach Learn Res. 2003;4:933–69.
25. Herbrich R, Graepel T, Obermayer K. Large margin rank boundaries for ordinal regression. In: Smola AJ, Bartlett P, Schölkopf B, Schuurmans D, editors. Advances in large-margin classifiers. Cambridge: The MIT Press; 2000. https://doi.org/10.7551/mitpress/1113.003.0010.
26. Fürnkranz J, Hüllermeier E. Preference learning and ranking by pairwise comparison. In: Fürnkranz J, Hüllermeier E, editors. Preference learning. Berlin: Springer; 2010. p. 65–82. https://doi.org/10.1007/978-3-642-14125-6_4.
27. Dehghani M, Zamani H, Severyn A, Kamps J, Croft WB. Neural ranking models with weak supervision. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2017. p. 65–74. https://doi.org/10.1145/3077136.3080832.
28. Cohen WW, Schapire RE, Singer Y. Learning to order things. J Artif Intell Res. 1999;10:243–70. https://doi.org/10.1613/jair.587.
29. Ailon N, Mohri M. Preference-based learning to rank. Mach Learn. 2010;80(2–3):189–211. https://doi.org/10.1007/s10994-010-5176-9.
30. Bruch S, Han S, Bendersky M, Najork M. A stochastic treatment of learning to rank scoring functions. In: Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM), ACM; 2020. p. 61–9. https://doi.org/10.1145/3336191.3371844.
31. Xia F, Liu T, Wang J, Zhang W, Li H. Listwise approach to learning to rank: theory and algorithm. In: Proceedings of the Twenty-Fifth International Conference Machine Learning (ICML), vol. 307. ACM; 2008. p. 1192–9. https://doi.org/10.1145/1390156.1390306.
32. Xu J, Li H. AdaRank: a boosting algorithm for information retrieval. In: Roceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2007. p. 391–8. https://doi.org/10.1145/1277741.1277809.
33. Bruch S, Zoghi M, Bendersky M, Najork M. Revisiting approximate metric optimization in the age of Deep Neural Networks. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2019. p. 1241–4. https://doi.org/10.1145/3331184.3331347.
34. Pobrotyn P, Bialobrzeski R. NeuralNdcg: direct optimisation of a ranking metric via differentiable relaxation of sorting. CoRR. abs/2102.07831. 2021.
35. Chapelle O, Chang Y. Yahoo! learning to rank challenge overview. In: Proceedings of the Yahoo! Learning to Rank Challenge, Held at ICML, vol. 14. JMLR.org; 2011. p. 1–24.
36. Donmez P, Svore KM, Burges CJC. On the local optimality of LambdaRank. In: Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2009. p. 460–7. https://doi.org/10.1145/1571941.1572021.
37. Beretta L, Nardini FM, Trani R, Venturini R. An optimal algorithm to find champions of tournament graphs. In: Proceedings of the 26th International Symposium String Processing and Information Retrieval—(SPIRE), Springer; 2019. p. 267–73. https://doi.org/10.1007/978-3-030-32686-9_19.
38. Nogueira R, Yang W, Cho K, Lin J. Multi-stage document ranking with BERT. CoRR. abs/1910.14424. 2019.
39. Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc: Ser B. 1977;39(1):1–22.
40. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T. LightGBM: a highly efficient gradient boosting decision tree. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS). 2017. p. 3146–54.
41. Leong WJ, Hassan MA, Yusuf MW. A matrix-free quasi-newton method for solving large-scale nonlinear systems. Comput Math Appl. 2011;62(5):2354–63. https://doi.org/10.1016/j.camwa.2011.07.023.
42. Järvelin K, Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans Inf Syst. 2002;20(4):422–46. https://doi.org/10.1145/582415.582418.
43. Qin T, Liu T. Introducing LETOR 4.0 datasets. CoRR. abs/1306.2597. 2013.
44. Yin D, Hu Y, Tang J Jr, Daly T, Zhou M, Ouyang H, Chen J, Kang C, Deng H, Nobata C, Langlois J, Chang Y. Ranking relevance in Yahoo Search. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. p. 323–32. ACM. https://doi.org/10.1145/2939672.2939677.
45. Bergstra J, Yamins D, Cox DD. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the 30th International Conference on Machine Learning, (ICML). JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org; 2013. p. 115–23.
46. Pasumarthi RK, Bruch S, Wang X, Li C, Bendersky M, Najork M, Pfeifer J, Golbandi N, Anil R, Wolf S. Tf-Ranking: Scalable tensorflow library for learning-to-rank. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), ACM. 2019. p. 2970–8. https://doi.org/10.1145/3292500.3330677.
47. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M. TensorFlow: a system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation, ({OSDI}). USENIX Association. 2016. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.
48. Lucchese C, Nardini FM, Orlando S, Perego R, Tonellotto N. Speeding-up document ranking with rank-based features. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), ACM; 2015. p. 895–8. https://doi.org/10.1145/2766462.2767776.
49. Wang X, Li C, Golbandi N, Bendersky M, Najork M. The LambdaLoss framework for ranking metric optimization. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM), ACM; 2018. p. 1313–22. https://doi.org/10.1145/3269206.3271784.