Pairwise learning to rank by neural networks revisited

Köppel, Marius; Segner, Alexander; Wagener, Martin; Pensel, Lukas; Karwath, Andreas; Kramer, Stefan

Pairwise Learning to Rank by Neural Networks Revisited: Reconstruction, Theoretical Analysis and Practical Performance

Marius Köppel[1⋆](✉), Alexander Segner[1⋆], Martin Wagener[1⋆], Lukas Pensel[1], Andreas Karwath[2], and Stefan Kramer[1]

[1] Johannes Gutenberg-Universität Mainz
Saarstraße 21, 55122 Mainz, Germany
makoeppe@students.uni-mainz.de
[2] University of Birmingham,
Haworth Building (Y2), B15 2TT, United Kingdom
a.karwath@bham.ac.uk

Abstract. We present a pairwise learning to rank approach based on a neural net, called DirectRanker, that generalizes the RankNet architecture. We show mathematically that our model is reflexive, antisymmetric, and transitive allowing for simplified training and improved performance. Experimental results on the LETOR MSLR-WEB10K, MQ2007 and MQ2008 datasets show that our model outperforms numerous state-of-the-art methods, while being inherently simpler in structure and using a pairwise approach only.

Keywords: Information Retrieval · Machine learning · Learning to rank.

1 Introduction

Information retrieval has become one of the most important applications of machine learning techniques in the last years. The vast amount of data in every day life, research and economy makes it necessary to retrieve only relevant data. One of the main problems in information retrieval is the *learning to rank* problem [6, 17]. Given a query q and a set of documents $d_1, \ldots, d_n$ one wants to find a *ranking* that gives a (partial) order of the documents according to their relevance relative to q. Documents can in fact be instances from arbitrary sets and do not necessarily need to correspond to queries.

Web search is one of the most obvious applications, however, product recommendation or question answering can be dealt with in a similar fashion. Most common machine learning methods have been used in the past to tackle the learning to rank problem [2, 7, 10, 14]. In this paper we use an artificial neural net which, in a pair of documents, finds the more relevant one. This is known as the pairwise ranking approach, which can then be used to sort lists of documents. The chosen architecture of the neural net gives rise to certain properties

⋆ These authors contributed equally.

which significantly enhance the performance compared to other approaches. We note that the structure of our neural net is essentially the same as the one of RankNet [2]. However, we relax some constraints which are used there and use a more modern optimization algorithm. This leads to a significantly enhanced performance and puts our approach head-to-head with state-of-the-art methods. This is especially remarkable given the relatively simple structure of the model and the consequently small training and testing times. Furthermore, we use a different formulation to describe the properties of our model and find that it is inherently reflexive, antisymmetric and transitive. In summary, the contributions of this paper are:

1. We propose a simple and effective scheme for neural network structures for pairwise ranking, called DirectRanker, which is a generalization of RankNet.
2. Theoretical analysis shows which of the components of such network structures give rise to their properties and what the requirements on the training data are to make them work.
3. Keeping the essence of RankNet and optimizing it with modern methods, experiments show that, contrary to general belief, pairwise methods can still be competitive with the more recent and much more complex listwise methods.

The paper is organized as follows: We discuss different models related to our approach in sec. 2. The model itself and certain theoretical properties are discussed in sec. 3 before describing the setup for experimental tests in sec. 4 and their results in sec. 5. Finally, we conclude our findings in sec. 6.

2 Related Work

There are a few fundamentally different approaches to the learning to rank problem that have been applied in the past. They mainly differ in the underlying machine learning model and in the number of documents that are combined in the cost during training. Common examples for machine learning models used in ranking are: decision trees [8], support vector machines [4], artificial neural nets [5], boosting [22], and evolutionary methods [12]. During training, a model must rank a list of n documents which can then be used to compute a suitable cost function by comparing it to the ideally sorted list. If $n = 1$ the approach is called *pointwise*. A machine learning model assigns a numerical value to a single document and compares it to the desired relevance label to compute a cost function. This is analogous to a classification of each document. If $n = 2$ the approach is called *pairwise*. A model takes two documents and determines the more relevant one. We implement this concept in our model, the DirectRanker. If $n > 2$ the approach is called *listwise* and the cost is computed on a whole list of sorted documents. Examples for these different approaches are [6, 9, 16] for pointwise, [2, 4, 8] for pairwise and [5, 12, 23] for listwise models.

Beside our own model the focus of this paper lies mainly on the pairwise approach RankNet [2] and the listwise approach LambdaMART [22]. RankNet

is a neural net defining a single output for a pair of documents. For training purposes, a cross entropy cost function is defined on this output.

LambdaMART on the other hand is a boosted tree version of LambdaRank [3] which itself is based on RankNet. Here, listwise evaluation metrics M are optimized by avoiding cost functions and directly defining λ-gradients

$$\lambda_i = \sum_j S_{ij} \left| \Delta M \frac{\partial C_{ij}}{\partial o_{ij}} \right|$$

where ΔM is the difference in the listwise metric when exchanging documents i and j in a query, C is a pairwise cost function, and o_{ij} is a pairwise output of the ranking model. $S_{ij} = \pm 1$ depending on whether document i or j is more relevant.

The main advantages of RankNet and LambdaMART are training time and performance: While RankNet performs well on learning to rank tasks it is usually outperformed by LambdaMART considering listwise metrics which is usually the goal of learning to rank. On the other hand, since for the training of LambdaMART it is necessary to compute a contribution to λ_i for every combination of two documents of a query for all queries of a training set, it is computationally more demanding to train this model compared to the pairwise optimization of RankNet (cf Table 2).

In general, multiple publications (most prominently [5]) suggest that listwise approaches are fundamentally superior to pairwise ones. As the results of the experiments discussed in sec. 5.1 show, this is not necessarily the case.

Important properties of rankers are their reflexivity, antisymmetry and transitivity. To implement a reasonable order on the documents these properties must be fulfilled. In [20] the need for antisymmetry and a simple method to achieve it in neural nets are discussed. Also [2] touches on the aspect of transitivity. However, to the best of our knowledge, a rigorous proof of these characteristics for a ranking model has not been presented so far. A theoretical analysis along those lines is presented in sec. 3 of the paper.

3 DirectRanker Approach

Our approach to ranking is of the pairwise kind, i.e. it takes two documents and decides which one is more relevant than the other. This approach comes with some difficulties, as, to achieve a consistent and unique ranking, the model has to define an order. In our approach, we implement a quasiorder $\succeq$ on the feature space $\mathcal{F}$ such that the ranking is unique except for equivalent documents, i.e. documents with the same relevance label. This quasiorder satisfies the following conditions for all $x, y, z \in \mathcal{F}$:

(A) Reflexivity: $x \succeq x$
(B) Antisymmetry: $x \not\succeq y \Rightarrow y \succeq x$
(C) Transitivity: $(x \succeq y \wedge y \succeq z) \Rightarrow x \succeq z$

We implement such an order using a ranking function $r : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ by defining

$$x \succeq y :\Leftrightarrow r(x, y) \geq 0\,. \tag{1}$$

The conditions (A)-(C) for $\succeq$ can be imposed in form of the following conditions for r:

(I) Reflexivity: $r(x, x) = 0$
(II) Antisymmetry: $r(x, y) = -r(y, x)$
(III) Transitivity: $(r(x, y) \geq 0 \wedge r(y, z) \geq 0) \Rightarrow r(x, z) \geq 0$

In our case, r is the output of a neural network with specific structure to fulfill the above requirements. As shown by [20], the antisymmetry can easily be guaranteed in neural network approaches by removing the biases of the neurons and choosing antisymmetric activation functions. Of course, the result will only be antisymmetric, if the features fed into the network are antisymmetric functions of the documents themselves, i.e., if two documents A and B are to be compared by the network, the extracted features of the document pair have to be antisymmetric under exchange of A and B.

This leads to the first difficulty since it is not at all trivial to extract such features containing enough information about the documents. Our model avoids this issue by taking features extracted from each of the documents and optimizing suitable antisymmetric features as a part of the net itself during the training process. This is done by using the structure depicted in Fig. 1.

The corresponding features of two documents are fed into the two subnets nn_1 and nn_2, respectively. These networks can be of arbitrary structure, yet they have to be identical, i.e. share the same structure and parameters like weights, biases, activation, etc. The difference of the subnets' outputs is fed into a third subnet, which further consists only of one ouptut neuron with antisymmetric activation and without a bias, representing the above defined function r. With the following theorem we show that this network satisfies conditions (I) through (III):

Theorem 1. *Let f be the output of an arbitrary neural network taking as input feature vectors $x \in \mathcal{F}$ and returning values $f(x) \in \mathbb{R}^n$. Let o_1 be a single neuron with antisymmetric and sign conserving activation function and without bias taking $\mathbb{R}^n$-valued inputs. The network returning $o_1(f(x) - f(y))$ for $x, y \in \mathcal{F}$ then satisfies (I) through (III).*

Proof. Let the activation function of the output neuron be $\tau : \mathbb{R} \to \mathbb{R}$ with $\tau(-x) = -\tau(x)$ and $\operatorname{sign}(\tau(x)) = \operatorname{sign}(x)$ as required.

(I) If (II) is fulfilled, then (I) is trivially so because

$$r(x, x) = -r(x, x) \forall x \in \mathcal{F} \Rightarrow r(x, x) \equiv 0\,.$$

(II) The two networks nn_1 and nn_2 are identical (as they share the same parameters). Hence, they implement the same function $f : \mathcal{F} \to \mathbb{R}^n$. The

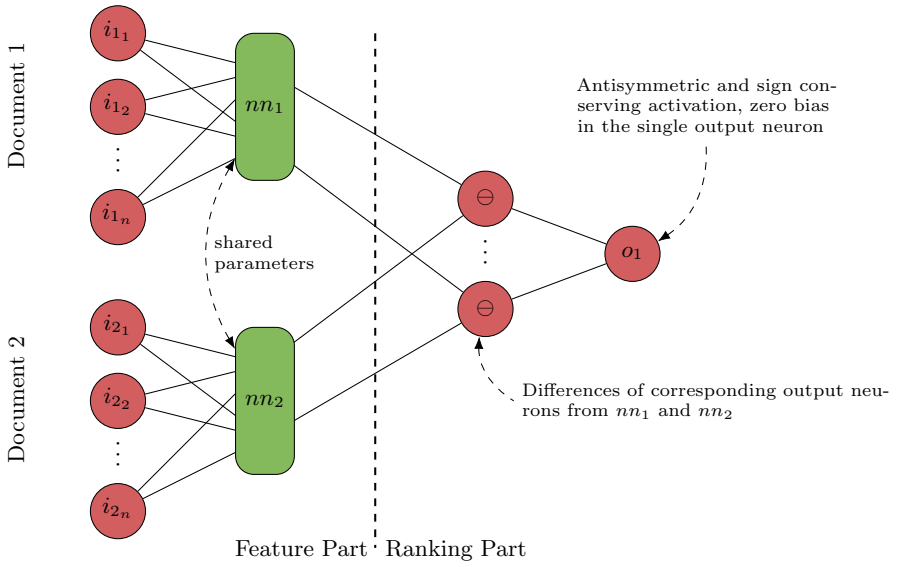

Fig. 1. Schema of the DirectRanker. nn_1 and nn_2 can be arbitrary networks (or other function approximators) as long as they give the same output for the same inputs i_j. The bias of the output neuron o_1 has to be zero and the activation antisymmetric and sign conserving.

output of the complete network for the two input vectors $x, y \in \mathcal{F}$ is then given by:

$$r(x,y) = \tau[w(f(x) - f(y))] = \tau[wf(x) - wf(y)] =: \tau[g(x) - g(y)]\,, \quad (2)$$

where w is a weight vector for the output neuron and $g : \mathcal{F} \to \mathbb{R}$. This is antisymmetric for x and y, thus satisfying the second condition (II).

(III) Let $x, y, z \in \mathcal{F}$, $r(x,y) \geq 0, r(y,z) \geq 0$, and let g be defined as in 2. Since τ is required to retain the sign of the input, i.e. $\tau(x) \geq 0 \Leftrightarrow x \geq 0$, $g(x) \geq g(y)$ and $g(y) \geq g(z)$, one finds

$$r(x,z) = \tau[g(x) - g(z)] = \tau\big[\underbrace{g(x) - g(y)}_{\geq 0} + \underbrace{g(y) - g(z)}_{\geq 0}\big] \geq 0\,.$$

Thus, r is transitive and (III) is fulfilled. □

These properties offer some advantages during the training phase of the networks for the distinction of different relevance classes:

(i) Due to antisymmetry, it is sufficient to train the network by always feeding instances with higher relevance in one and instances with lower relevance in the other input, i.e. higher relevance always in i_1 and lower relevance always in i_2 or vice versa.

(ii) Due to transitivity, it is not necessary to compare very relevant and very irrelevant documents directly during training. Provided that every document is trained at least with documents from the corresponding neighbouring relevance classes, the model can implicitly be trained for all combinations, given that all classes are represented in the training data.
(iii) Although it might seem to be sensible to train the model such that it is able to predict the equality of two different documents of the same relevance class, the model is actually restricted when doing so: If the ranker is used to sort a list of documents according to their relevance, there is no natural order of documents within the same class. Hence, the result of the ranker is not relevant for equivalent documents. Furthermore, when only documents of different classes are paired in training, the optimizer employed in the training phase has more freedom to find an optimal solution for ranking relevant cases, potentially boosting the overall performance.

Note that the DirectRanker is a generalization of the RankNet model [2], which is equivalent to the DirectRanker if the activation of o_1 is $\tau(x) = \tanh\left(\frac{x}{2}\right)$, and if a cross entropy cost, and a gradient descent optimizer are chosen, which are free parameters in our model.

For simplicity, we will from now on choose the activation function to be $\tau \equiv \mathrm{id}$. This can be done without loss of generality, since the activation function does not change the order, if τ is sign conserving.

In the following, we try to put the DirectRanker on a more sound basis by analyzing some cases in which the DirectRanker is able to approximate an order (given enough complexity and training samples). More precisely, we present some cases in which the following conditions are guaranteed to be met:

(i) There exists an order $\succeq$ satisfying (A)-(C) on the feature space $\mathcal{F}$.
(ii) A given order $\succeq$ on $\mathcal{F}$ can be represented by the DirectRanker, i.e. there is a continuous function $r : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ implementing the axioms (I)-(III) and which can be written as $r(x, y) = g(x) - g(y)$ on the whole feature space.

By the universal approximation theorem [11], the second condition implies that r can be approximated to arbitrary precision by the DirectRanker. In the following, we will discuss interesting cases, in which these assumptions are valid:

Theorem 2. *For every countable feature space $\mathcal{F}$ there exists an order $\succeq$ that is reflexive, antisymmetric, and transitive.*

Proof. By definition, for a countable set $\mathcal{F}$, there exists an injective function $g : \mathcal{F} \to \mathbb{N}$. Therefore, choose $x \succeq y :\Leftrightarrow g(x) \geq g(y)$ for $x, y \in \mathcal{F}$. □

In fact, every sorting of the elements of countable sets satisfies (A)-(C), and as we show in the next theorem, it can be approximated by the direct ranker, if the set is uniformly dense:

Theorem 3. *Let $\succeq$ implement (A)-(C) on an uniformly discrete feature space $\mathcal{F}$. Then, the DirectRanker can approximate a function that represents $\succeq$.*

Proof. First, consider $\mathcal{F}$ to be an infinite set. We will use the same notation as above to describe the ranking function r in terms of a continuous function g such that $r(x, y) = g(x) - g(y)$. Since we use neural networks to approximate g, referring to the universal approximation theorem [11], it is sufficient to show that a continuous function $g : \mathbb{R}^n \to \mathbb{R}$ exists such that r has the required properties. We will show that such a function exists by explicit construction. We can iterate through $\mathcal{F}$, since it is discrete and, therefore, countable. Now assign each element $x \in \mathcal{F}$ a value $g(x) \in \mathbb{R}$. Map the first value x_0 to 0, and then iteratively do the following with the i-th element of $\mathcal{F}$:

p1.1 If $\exists j : j < i \wedge x_i \succeq x_j \wedge x_j \succeq x_i$, set $g(x_i) := g(x_j)$ and continue with the next element.

p1.2 If $\forall j$ with $j < i : x_i \succeq x_j$, set $g(x_i) := \max_{j<i} g(x_j) + 1$ and continue with the next element.

p1.3 If $\forall j$ with $j < i : x_j \succeq x_i$, set $g(x_i) := \min_{j<i} g(x_j) - 1$ and continue with the next element. If there are $j, k < i$ with $x_k \succeq x_i \succeq x_j$, choose an arbitrary "largest element smaller than x_i", i.e. an element $x_l \in \mathcal{F}, l < i$ satisfying $x_i \succeq x_l \succeq x \forall x \in \{x_j \in \mathcal{F} | j < i, x_j \not\succeq x_i\}$, and an arbitrary "smallest element larger than x_i", i.e. an element $x_g \in \mathcal{F}, g < i$ such that $x \succeq x_g \succeq x_i \forall x \in \{x_k \in \mathcal{F} | k < i, x_i \not\succeq x_k\}$. Then set $g(x_i) := \frac{g(x_l)+g(x_g)}{2}$ and continue with the next element. This is well-defined since steps 1 through 3 guarantee that every x_l that can be chosen this way is mapped to the same value by g. Analogously for x_g.

One easily sees that this yields a function g for which $g(x) \geq g(y) \Leftrightarrow x \succeq y \forall x, y \in \mathcal{F}$ and thus $r(x, y) = g(x) - g(y) \geq 0 \Leftrightarrow x \succeq y$.

Next, we expand g to a continuous function in $\mathbb{R}^n$. Since $\mathcal{F}$ is uniformly discrete, $\exists \delta > 0 \forall i \in \mathbb{N} : B_\delta(x_i) \cap \mathcal{F} = \{x_i\}$, where $B_\delta(x_i) := \{x \in \mathbb{R}^n | \|x - x_i\| < \delta\}$. For every $i \in \mathbb{N}$ define $\tilde{B}_i : \overline{B_{\delta/42}(x_i)} \to \mathbb{R}$, $x \mapsto 1 - \frac{42\|x-x_i\|}{\delta}$. $\tilde{B}_i$ is obviously continuous on $\overline{B_{\delta/42}(x_i)}$. Expanding this definition to

$$B_i(x) := \begin{cases} \tilde{B}_i(x) & \text{if } x \in \overline{B_{\delta/42}(x_i)} \\ 0 & \text{else} \end{cases}$$

allows us to define a function $g_c : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \sum_{i=1}^{\infty} g(x_i) B_i(x)$ which results in the same value as g for all relevant points x_i. This can easily be checked, since $B_n(x_m) = \delta_{mn}$ (using the Kronecker-delta). Thus, it still represents $\succeq$ on $\mathcal{F}$. B_i is continuous since $B_i|_{\overline{B_{\delta/42}}} = \tilde{B}_i$ and $B_i|_{\mathbb{R}^n \setminus B_{\delta/42}} \equiv 0$ are continuous and, therefore, B_i is continuous on the union of these closed sets. We now show that g_c is continuous using the ε-δ-definition:

Let $\varepsilon > 0$ and $x \in \mathbb{R}^n$. If there is no $n \in \mathbb{N}$ such that $x \in \overline{B_{\delta/42}(x_i)}$, we can choose $\tilde{\delta} > 0$ such that $B_{\tilde{\delta}}(x) \cap \overline{B_{\delta/42}(x_i)} = \emptyset \ \forall n \in \mathbb{N}$ since $\mathcal{F}$ is uniformly discrete. Therefore, $g_c|_{B_{\tilde{\delta}}} \equiv 0$ and $|g_c(\tilde{x}) - g_c(x)| = 0 < \varepsilon \forall \tilde{x} \in B_{\tilde{\delta}}(x)$.

If there is such an n, then $B_{\delta/4}(x) \cap \overline{B_{\delta/42}(x_i)}$ is non-empty, if and only if $n = i$. Hence, we can choose $\frac{\delta}{4} > \tilde{\delta} > 0$, such that $|g_c(\tilde{x}) - g_c(x)| < \varepsilon \forall \tilde{x} \in B_{\tilde{\delta}}(x)$

since $g_c|_{B_{\delta/4}(x_i)} = g(x_i) \cdot B_i|_{B_{\delta/4}(x_i)}$ is clearly continuous. Therefore, for every $\varepsilon > 0$ and $x \in \mathbb{R}^n$ we can find $\tilde{\delta} > 0$ such that $|g_c(\tilde{x}) - g_c(x)| < \varepsilon \forall \tilde{x} \in B_{\tilde{\delta}(x)}$, i.e., g_c is continuous.

If $\mathcal{F}$ is finite with N elements, set $g(x_k) = 0$ for $k > N$. Then the proof works analogously as the above. □

Therefore, it theoretically feasible to successfully train the DirectRanker on any finite dataset, and consequently on any real-world dataset. However, the function g might be arbitrarily complex depending on the explicit order. In real-world applications, the desired order is usually not discrete and the task at hand is to predict the order of elements *not* represented in the training data. In the following, we give a reasonably weak condition for which an order $\succeq$ can be approximated by the DirectRanker on more general feature spaces:

Theorem 4. *Let $\succeq$ implement (A)-(C) and $\mathcal{F} \subset \mathbb{R}^n$ be convex and open. For every $x \in \mathcal{F}$ define*

$$\mathcal{P}_x := \{y \in \mathcal{F} | x \not\succeq y\},\ \mathcal{N}_x := \{y \in \mathcal{F} | y \not\succeq x\},\ \partial_x := \{y \in \mathcal{F} | x \succeq y \wedge y \succeq x\}\,.$$

Furthermore, let $(\mathcal{F}/\sim, d)$ be a metric space, where $x \sim y \Leftrightarrow y \in \partial_x$ and $d(\partial_x, \partial_y) = \inf_{x' \in \partial_x, y' \in \partial_y} \|x' - y'\|$. Then, the DirectRanker can approximate $\succeq$ if $\mathcal{P}_x$ and $\mathcal{N}_x$ are open for all $x \in \mathcal{F}$.

Proof (Sketch of the proof). By using the relations (A)-(C), one can show that the function $g : \mathcal{F} \to \mathbb{R}$,

$$g(x) = \begin{cases} d(\partial_{x_0}, \partial_x) & \text{if } x \in \mathcal{P}_{x_0} \\ -d(\partial_{x_0}, \partial_x) & \text{if } x \in \mathcal{N}_{x_0} \\ 0 & \text{if } x \in \partial_{x_0} \end{cases} \quad \forall x \in \mathcal{F}$$

with $g(x_0) = 0$ for some $x_0 \in \mathcal{F}$ satisfies the requirements that g is continuous and implements (A)-(C). This can be done by regarding lines from $x_0' \in \partial_{x_0}$ to $x \in \mathcal{P}_{x_0}$ and elaborating on the fact that the line has to pass through some $y' \in \partial_y$ if $x \in \mathcal{P}_y$. For more detail, see the supplementary material to this article. □

If there are no two documents with the same features but different relevances, any finite dataset can be extended to $\mathbb{R}^n$ such that the conditions for 4 are met. In real-world applications, i.e. applications with noisy data, it is in general possible that $\mathcal{P}_x$, $\mathcal{N}_x$, and ∂_x blur out and overlap. In this case, it is of course impossible to find any function that represents $\succeq$. However, the DirectRanker still ought to be able to find a "best fit" of a continuous function that maximizes the predictive power on any new documents, even if some documents in the training set are mislabeled. Experiments investigating this are discussed in sec. 5.2.

4 Experimental Setup

To evaluate the performance of our model and to compare it to other learning to rank approaches, we employ common evaluation metrics and standard datasets

(MSLR-WEB10K, MQ2007, MQ2008 [19]). Furthermore we use synthetic data to investigate the dependence of the performance on certain characteristics of the data. Reliable estimates for the performance are gained by averaging over different splits of the dataset using cross-validation on the predefined folds from the the original publications and are compared to other models. In all tests, we use the tensorflow library [1] and its implementation of the Adam-Optimizer [15]. In sec. 4.1 we briefly describe the structure of the LETOR datasets and in sec. 4.2 how the models are evaluated. In sec. 4.3 we illustrate how the synthetic datasets are generated and analyzed. For evaluating different models we apply the commonly used metrics NDCG and MAP which are further discussed in the supplemental material.

4.1 The LETOR Datasets

The *Microsoft Learning to Rank Datasets* (LETOR) and especially the MSLR–WEB10K set are standard data sets that are most commonly used to benchmark *learning to rank* models. The dataset consists of 10,000 queries and is a subset of the larger MSLR–WEB30K dataset. Each instance in the dataset corresponds to a query-document pair which is characterized by 136 numerical features. Additionally, relevance labels from 0 (irrelevant) to 4 (perfectly relevant) indicate the relevance of the given document with respect to the query. Ranking documents according to their relevance is often simplified by binarizing the relevance labels using an appropriate threshold, as is done by [12, 13]. In our case, we map relevance labels ≥ 2 to 1 and relevance labels ≤ 1 to 0. We use this approach to compare our model to others.

Additionally we evaluate the different algorithms on the much smaller MQ2007 and MQ2008 datasets. These are similar in structure to the MSLR–WEB10K set with some minor differences: The relevance labels range from 0 to 2 and each document consists of 46 features. In this case we binarize the relevance labels by mapping labels ≥ 1 to 1 and relevance labels $= 0$ to 0.

4.2 Grid search and LETOR Evaluation

We perform grid searches for hyperparameter optimization of our model. The grid searches were performed using the *GridSearchCV* class implemented in the *scitkit-learn* library [18]. One grid search was done to optimize the NDCG@10 and one optimizing the MAP. For each hyperparameter point a 5-fold cross validation (internal) was performed on each training set on each of the 5 predefined folds of the datasets. The models were then again trained using the best hyperparameters using the entire training set before averaging the performance on independent test sets over all 5 folds (external). Before training the model the data was transformed in such a way that the features are following a normal distribution with standard deviation of $1/3$.

For benchmarking the results, the most common *learning to rank* algorithms were also trained and tested with the same method as described above. The

implementation of these algorithms are taken from the RankLib library implemented in the Lemur Project [21]. The algorithms are: RankNet [2], AdaRank [23], LambdaMART [22], and ListNet [5].

The used datasets contain queries with only non relevant documents for which the evaluation metrics are not defined. Consequently we exclude those queries from the data.

Furthermore, there are queries with less than 10 documents. For such queries with $k < 10$ documents the NDCG@k is evaluated during the tests.

4.3 Synthetic Data Generation and Evaluation

To study how the DirectRanker performs for differently structured datasets, synthetic data with different characteristics were created and evaluated.

To achieve comparability between the different sets, all datasets have the following properties in common:

(i) The dataset consists of separate training and test sets which are generated independently, but with the same parameters.
(ii) For each relevance class, the features follow a Gaussian distribution in feature space with a constant, but random mean between 0 and 100, and a constant, but random standard deviation between 50 and 100.
(iii) Except for a test in which the performance depending on the size of the dataset is studied, all training sets consist of 10^5 and all test sets consist of 10^4 documents.
(iv) During training, $r(d_i)(1 - o_1(d_i, d_j))^2$ is applied as the cost function as pairs are constructed such that $r(d_i) - r(d_j) = 1$. Here, $r(d)$ is the relevance label of document d.

For the different tests, one parameter describing the dataset is changed and evaluated for different values, keeping all other parameters fixed. These parameters include the size of the training set, the number of relevance classes, the number of features, and noise on the labels. The noise for the labels is generated by assuming a Gaussian for each label with variable standard deviation and rounding to the next integer. This allows for testing larger greater degrees of confusion between more distant relevance classes.

The general procedure for the experiments is the following:

(1) A dataset with the respective parameters is created.
(2) The DirectRanker is trained on the generated training set using our framework.
(3) The trained ranker is tested on the generated test set, again using our framework. For this, 50-150 random samples are drawn from the test set. This subset is then sorted using the trained model and the NDCG@20 is calculated. The whole test is repeated 50 times and the average value of NDCG@20 over these 50 random subsets is calculated.
(4) These three steps are repeated at least four more times to determine a mean value μ for the NDCG@20, averaged over different datasets with the same characteristics. The standard error is calculated as an uncertainty $\Delta\mu$ of μ.

In the plots showing our test results (Fig. 2a, Fig. 2b, Fig. 2d, Fig. 2c), every data point is the result of applying these four steps for one choice of the dataset parameters. nn_1 and nn_2 consist of a hidden layer with 70 neurons and another one with as many neurons as there are relevance classes. The results of these tests are discussed in sec. 5.2.

5 Experimental Results

In this section we present the experimental results. First, we compare our model with the commonly used ones (sec. 5.1). Additionally, we give an outline of the sensitivity on different dataset properties (sec. 5.2).

5.1 Comparison to other Rankers

In Table 1 the results of different models on the datasets discussed in sec. 4.1 are shown. On the MQ2007 and MQ2008 datasets the differences between the models are insignificant (0.54σ difference in the NDCG@10 between the best and worst performing algorithms on the MQ2008 dataset) making the experiments on these sets inconclusive. However, the results on the MSLR–WEB10K set differ significantly. Here LambdaMart outperforms the DirectRanker by 7.2σ on the NDCG@10. On the MAP however, the difference is only 0.2σ. It is important to note that for LambdaMart the model was explicitly boosted on NDCG@10 and MAP respectively for the two performance values while the DirectRanker uses a cost function independent of the evaluation metric. On the MSLR–WEB10K set the DirectRanker outperforms all other methods by at least 2.4σ (NDCG@10) or 3.2σ (MAP).

Table 1. Performance comparison for different rankers on multiple Letor datasets. The values for ES-Rank, IESR-Rank and IESVM-Rank are taken from [13]. These values are marked with italic. LambdaMart was boosted using the corresponding evaluation metric during training.

	MSLR-WEB10K		MQ2008		MQ2007	
Algorithm	⟨NDCG⟩	⟨MAP⟩	⟨NDCG⟩	⟨MAP⟩	⟨NDCG⟩	⟨MAP⟩
DirectRanker	0.440(4)	0.365(3)	0.720(12)	0.636(11)	0.540(10)	0.534(9)
RankNet	0.157(3)	0.195(2)	0.716(11)	0.642(10)	0.525(11)	0.525(7)
ListNet	0.157(3)	0.192(2)	0.719(10)	0.647(6)	0.526(10)	0.525(9)
LambdaMart	0.476(3)	0.366(3)	0.723(7)	0.624(6)	0.531(12)	0.510(11)
AdaRank	0.400(16)	0.322(10)	0.722(10)	0.653(9)	0.526(10)	0.527(10)
ES-Rank	*0.382*	*0.570*	*0.507*	*0.483*	*0.451*	*0.470*
IESR-Rank	*0.415*	*0.603*	*0.517*	*0.494*	*0.455*	*0.473*
IESVM-Rank	*0.224*	*0.457*	*0.498*	*0.473*	*0.436*	*0.456*

To demonstrate the simplicity of the DirectRanker, we present experiments on the runtime for the model training in Table 2. All tests performed by us have

Table 2. Comparing the average run times over the five folds of the MSLR–WEB10K data set in seconds. The values with ∗ were trained on the machine mentioned in the text. The values with † were trained using RankLib. The values with ‡ are taken from [13]. For RankNet two implementations were used. One with Tensorflow and one from RankLib.

Algorithm	time in seconds
DirectRanker*	151.94(41)
RankNet*	142.27(69)
RankNet*†	2215(351)
AdaRank*†	1261(50)
LambdaMART*†	2664(234)
ListNet*†	3947(481)
ES-Rank‡	1800
IESR-Rank‡	1957
IESVM-Rank‡	34 209

been conducted on an Intel® Core™ i7-6850K CPU @ 3.60GHz using the above mentioned MSLR–WEB10K dataset averaging over the five given folds. Our model was trained using Tensorflow [1], contrary to the other implementations. This makes the comparison of the run times difficult, however, we also reimplemented RankNet using Tensorflow in the same way as our model. Here it can be seen that the runtime of the DirectRanker and RankNet are of the same order. Thus, we do not boost the training time of the model but only the performance. On the other hand the training time beats all the other models. [3]

5.2 Sensitivity on Dataset Properties

With the following tests we discuss how the DirectRanker performs under different circumstances. The tests were performed as described in sec. 4.3. The performance of the DirectRanker was tested for different numbers of relevance classes (Fig. 2a), features (Fig. 2b), for variations of noise on the class labels (Fig. 2c), and differently sized training sets (Fig. 2d).

The tests show that, given enough data, our model is able to handle a diverse range of datasets. It especially shows that the DirectRanker can handle many relevance classes as shown in Fig. 2a. As one would expect, the performance decreases with the number of relevance classes. However, this effect can be counteracted by increasing the size of the training set (see Fig. 2d) or the number of features (Fig. 2b). Additionally, Fig. 2c shows the robustness of the DirectRanker against noise on the relevance classes. Up to some small noise (approximately 5% mislabeling, i.e. $\sigma = 0.25$), the performance decreases only marginally, but drops significantly for larger noise. Still, even with 50% of the documents being mislabeled (i.e. $\sigma = 0.75$) the NDCG@20 does not drop below

[3] For our implementation of the model and the tests see https://github.com/kramerlab/direct-ranker.

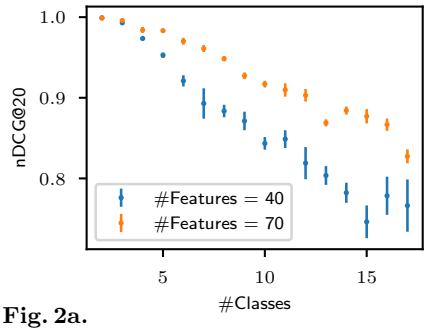

Fig. 2a.

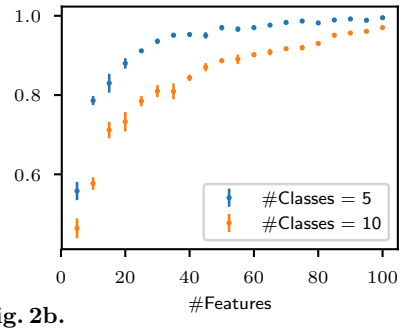

Fig. 2b.

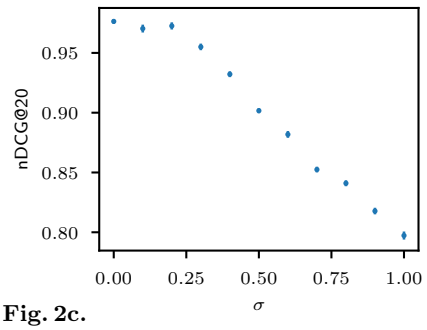

Fig. 2c.

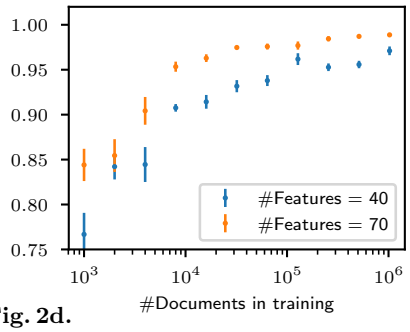

Fig. 2d.

Fig. 2. Plots depicting the sensitivity of the DirectRanker performance on certain data properties, evaluated on the synthetic data (sec. 4.3). (Fig. 2a) Dependence on the number of relevance classes (10^5 documents in training set). (Fig. 2b) Dependence on the number of features (10^5 documents in training set, five relevance classes). (Fig. 2c) Performance of the DirectRanker with different noise levels on the class labels with 5 classes and 70 features. (Fig. 2d) Dependence on the size of the training set (five relevance classes).

0.80. This suggests that the theoretical findings in sec. 3 for ideal data stay valid for real-world data.

6 Discussion and Conclusions

The scheme for network structures proposed and analyzed in this paper is a generalization of RankNet: We show which properties of components of RankNet are essential to bring about its favorable behavior and doing so, pave the way for performance improvements. As it turns out, only a few assumptions about the network structures are necessary to be able to learn an order of instances. The requirements on the data for training are also minimal: The method can be applied to discrete and continuous data, and can be employed for simplified training schedules with the comparison of neighboring classes (or other relevant pairs of relevance classes) only. Theoretical results shed some light on the reasons

why this is the case. Experiments confirm this and show that the scheme delivers excellent performance also on real-world data, where we may assume that instances are mislabeled with a certain probability. In many recent comparisons, RankNet is shown to exhibit inferior performance, leading to the conclusion that listwise approaches are to be preferred over pairwise approaches. Looking at the experimental results on the LETOR dataset in this paper, there may be reason to reconsider that view. However, it might be interesting to adapt the ideas LambdaRank and LambdaMART for listwise optimization to the DirectRanker model.

Also, it is remarkable that such a simple, transparent approach can match or outperform the performance of more recent and much more complex models, like ES-Rank and the like. Experiments with synthetic data show how the performance can degrade when given more relevance classes, fewer features or fewer training instances. However, these results also indicate how the loss of performance can be compensated by the other factors. Additionally to standard ranking, we suggest that the DirectRanker can be used for classification as well. First tests showed promising results. A more systematic investigation of this is the subject of future work.

Acknowledgement

We would like to thank Dr. Christian Schmitt for his contributions to the work presented in this paper.

We also thank Luiz Frederic Wagner for proof(read)ing the mathematical aspects of our model.

Parts of this research were conducted using the supercomputer Mogon and/or advisory services offered by Johannes Gutenberg University Mainz (hpc.uni-mainz.de), which is a member of the AHRP (Alliance for High Performance Computing in Rhineland Palatinate, www.ahrp.info) and the Gauss Alliance e.V.

The authors gratefully acknowledge the computing time granted on the supercomputer Mogon at Johannes Gutenberg University Mainz (hpc.uni-mainz.de).

This research was partially funded by the Carl Zeiss Foundation Project: 'Competence Centre for High-Performance-Computing in the Natural Sciences' at the University of Mainz. Furthermore, Andreas Karwath has been co-funded by the MRC grant MR/S003991/1.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `http://tensorflow.org/`, software available from tensorflow.org

2. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of the 22Nd International Conference on Machine Learning. pp. 89–96. ICML '05, ACM, New York, NY, USA (2005), http://doi.acm.org/10.1145/1102351.1102363
3. Burges, C., Ragno, R., Le, Q., Burges, C.J.: Learning to rank with non-smooth cost functions. In: Advances in Neural Information Processing Systems 19. MIT Press, Cambridge, MA (January 2007), https://www.microsoft.com/en-us/research/publication/learning-to-rank-with-non-smooth-cost-functions/
4. Cao, Y., Xu, J., Liu, T.Y., Li, H., Huang, Y., Hon, H.W.: Adapting ranking svm to document retrieval. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 186–193. ACM (2006). https://doi.org/10.1145/1148170.1148205
5. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: From pairwise approach to listwise approach p. 9 (April 2007), https://www.microsoft.com/en-us/research/publication/learning-to-rank-from-pairwise-approach-to-listwise-approach/
6. Cooper, W.S., Gey, F.C., Dabney, D.P.: Probabilistic retrieval based on staged logistic regression. In: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 198–210. ACM (1992), http://doi.acm.org/10.1145/133160.133199
7. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y.: An efficient boosting algorithm for combining preferences. Journal of machine learning research **4**(Nov), 933–969 (2003), http://dl.acm.org/citation.cfm?id=945365.964285
8. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29**, 1189–1232 (2000), http://www.jstor.org/stable/2699986
9. Fuhr, N.: Optimum polynomial retrieval functions based on the probability ranking principle. ACM Transactions on Information Systems (TOIS) **7**(3), 183–204 (1989)
10. Herbrich, R., Graepel, T., Obermayer, K.: Large margin rank boundaries for ordinal regression. advances in large margin classifiers (2000)
11. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8
12. Ibrahim, O.A.S., Landa-Silva, D.: Es-rank: evolution strategy learning to rank approach. In: Proceedings of the Symposium on Applied Computing. pp. 944–950. ACM (2017). https://doi.org/10.1145/3019612.3019696
13. Ibrahim, O.A.S., Landa-Silva, D.: An evolutionary strategy with machine learning for learning to rank in information retrieval. Soft Computing **22**(10), 3171–3185 (2018). https://doi.org/10.1007/s00500-017-2988-6
14. Jiang, L., Li, C., Cai, Z.: Learning decision tree for ranking. Knowledge and Information Systems **20**(1), 123–135 (2009)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Li, P., Wu, Q., Burges, C.J.: Mcrank: Learning to rank using multiple classification and gradient boosting. In: Advances in neural information processing systems. pp. 897–904 (2008)
17. Liu, T.Y.: Learning to rank for information retrieval. Found. Trends Inf. Retr. **3**(3), 225–331 (Mar 2009). https://doi.org/10.1561/1500000016
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

19. Qin, T., Liu, T.: Introducing LETOR 4.0 datasets. CoRR **abs/1306.2597** (2013), http://arxiv.org/abs/1306.2597
20. Rigutini, L., Papini, T., Maggini, M., Bianchini, M.: A neural network approach for learning object ranking. In: International Conference on Artificial Neural Networks. pp. 899–908. Springer (2008), https://doi.org/10.1007/978-3-540-87559-8_93
21. W. B. Croft, J.C.: Lemur toolkit (2001-2012), http://lemurproject.org/contrib.php
22. Wu, Q., Burges, C.J., Svore, K.M., Gao, J.: Adapting boosting for information retrieval measures. Information Retrieval **13**, 254–270 (June 2010), https://www.microsoft.com/en-us/research/publication/adapting-boosting-for-information-retrieval-measures/
23. Xu, J., Li, H.: Adarank: A boosting algorithm for information retrieval. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 391–398. SIGIR '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1277741.1277809