



Pairwise learning to rank by neural networks revisited: reconstruction, theoretical analysis and practical performance

Marius Köppel^{1,2} · Alexander Segner¹ · Martin Wagener² · Lukas Pensel¹ · Andreas Karwath³ · Stefan Kramer¹

Received: 23 November 2021 / Revised: 3 February 2024 / Accepted: 22 September 2024 /
Published online: 4 March 2025
© The Author(s) 2025

Abstract

We reevaluate the pairwise learning to rank approach based on neural nets, called Rank-Net, and present a theoretical analysis of its architecture. We show mathematically that the model can, under certain conditions, learn reflexive, antisymmetric, and transitive relations, enabling simplified training and improved performance. Experimental results on the LETOR MSLR-WEB10K, MQ2007 and MQ2008 datasets show that the model outperforms numerous state-of-the-art methods (including a listwise approach), 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 come to be one of the most important applications of machine learning in past decades. The vast amount of data in everyday life, research and economy makes it necessary to retrieve only relevant data. One of the key problems in information retrieval is the *learning to rank* problem (Cooper et al., 1992; Liu, 2009). Given a query q and a set of documents $d_1, \dots, d_n$, one wants to find a *ranking* that gives a total quasiorder of the documents according to their relevance relative to q . Documents can in fact be instances from arbitrary sets and do not necessarily have to correspond to queries.

Editor: Eyke Hüllermeier.

Marius Köppel, Alexander Segner, and Martin Wagener have contributed equally to this work.

✉ Marius Köppel
mkoeppel@uni-mainz.de

¹ Johannes Gutenberg-Universität Mainz, Mainz, Germany

² Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland

³ University of Birmingham, Birmingham, UK

Web search is one of the most straightforward 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 (Burges et al., 2005; Freund et al., 2003; Herbrich et al., 2000; Jiang et al., 2009). In this paper we use the artificial neural network architecture RankNet (Burges et al., 2005) which, in a pair of documents, retrieves 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 network gives rise to certain properties which significantly enhance the performance compared to other approaches. We note that the structure of RankNet is known and has been applied in many applications. However, we relax some constraints used there and use a more modern optimization algorithm. This leads to a significantly enhanced performance and establishes the approach head-to-head with state-of-the-art methods. This is 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 RankNet and find that it is inherently reflexive, antisymmetric, and transitive. In summary, the contributions of this paper are:

1. We propose a scheme for analyzing neural network structures for pairwise ranking based on the RankNet architecture.
2. Theoretical analysis shows which of the components of RankNet give rise to its 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 demonstrate that, contrary to general belief, *pairwise methods* can still be competitive with the more recent and much more complex *listwise methods*.
4. Finally, we show that the method can be used in principle for classification as well. In particular, it has the ability to discover subclasses in classes, if the latter exist.

The paper is organized as follows: We discuss different models related to RankNet in Sect. 2. The RankNet model itself and certain theoretical properties are discussed in Sect. 3, before describing the used setup for experiments in Sect. 4 and their results in Sect. 5. Finally, we draw our overall conclusions in Sect. 6.

2 Related work

There are a few fundamentally different approaches to the learning to rank problem that have been followed in the past. They mainly differ in the underlying machine learning model and in the number of documents compared in the cost function during training. 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 particular 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 receives two documents and determines the more relevant one. We study the concept of such models in our work. If $n > 2$, the approach is called *listwise* and the cost is computed on a whole list of sorted documents. Common examples for *pointwise* approaches are a stacked logistic regression model (Cooper et al., 1992), a polynomial retrieval function (Fuhr, 1989), and McRank, which uses gradient

boosting (Li et al., 2008). Examples for *pairwise* approaches are the neural network based RankNet (Burges et al., 2005), a support vector machine (Cao et al., 2006), and a gradient boosting algorithm (Friedman, 2000). For *listwise* models, the neural network based ListNet model (Cao et al., 2007), the evolutionary method ES-Rank (Ibrahim and Landa-Silva, 2017), and the boosting algorithms AdaRank (Xu and Li, 2007), and LambdaMART (Wu et al., 2010) are common examples.

The focus of this paper lies mainly on the pairwise approach RankNet (Burges et al., 2005) and the listwise approach LambdaMART (Wu et al., 2010). RankNet is a neural network defining a single output for a pair of documents. For training purposes, a cross entropy cost-function is defined on this output. The architecture of two networks sharing their weights, used by RankNet, was originally proposed by Tesauro (1988). Although weight sharing has since become a popular concept in natural language processing with Siamese networks (Reimers and Gurevych, 2019; Siekiera et al., 2022), our approach generalizes, amongst others, to parameter sharing beyond neural network architectures.

To summarize, the main extension in comparison to the original RankNet are:

1. The use of an antisymmetric and sign conserving activation function.
2. The use of the Adam optimizer (Kingma and Ba, 2014).
3. The use of modern regularization techniques such as dropout (Hinton et al., 2012), weight regularization, and a staircase decay of the learning rate.
4. The idea of weight sharing is generalized beyond neural networks by supporting any universal function approximator in the feature part of the model.

In the experimentation section we will name the original hyperparameter setting of the RankNet implementation RankNet and we will note the results for RankNet plus the optimizations we applied as RankNet . The same has been done for ListNet to highlight the improvements by using modern optimization techniques (ListNet). LambdaMART on the other hand is a boosted tree version of LambdaRank (Burges et al., 2007), 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. During the training of LambdaMART, it is necessary to compute a contribution to λ_i for every combination of two documents of a query in the training set. This leads to a computationally more demanding training of LambdaMART compared to the pairwise optimization of RankNet (cf Table 3). Multiple publications suggest that listwise approaches are fundamentally superior to pairwise ones (Cao et al., 2007; Rigutini et al., 2008). As the results of the experiments discussed in Sect. 5.1 show, this is not necessarily the case.

To properly implement a reasonable order on the documents, reflexivity, totality and transitivity must be fulfilled. The fundamental need for totality and transitivity are

discussed in the peer-reviewed literature (Rigutini et al., 2008; Burges et al., 2005). If these criteria are to be respected in a learned model, it is usually helpful to restrict the model to order embeddings (Vendrov et al., 2016; Davey and Priestley, 2002; Cohen, 2007). However, to the best of our knowledge, a rigorous proof of these essential characteristics for a ranking model has not been presented so far. A theoretical analysis along those lines is presented in Sect. 3 of the paper.

3 Theoretical analysis of RankNet

The approach to ranking we analyzed in this work is of the pairwise kind, i.e. it takes two documents and decides which one is more relevant than the other. To realize a consistent and unique ranking the model has to define an order. To achieve this, we prove that RankNet like architectures do implement a total quasiorder $\succeq$ on the feature space that satisfies the following conditions for all $x, y, z \in \mathcal{F}$:

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

$\mathcal{F}$ can in principle be any ordered set, as long as it can be represented in binary data. In practice, however, $\mathcal{F}$ is oftentimes simply a subset of $\mathbb{R}^n$.

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

$$x \succeq y \Leftrightarrow r(x, y) \geq 0. \quad (1)$$

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 previous requirements. The antisymmetry¹ of r can easily be guaranteed in neural network approaches by removing the biases of the neurons and choosing antisymmetric activation functions (Rigutini et al., 2008). Obviously, 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 trivial to extract such features containing enough information about the documents. RankNet-like architectures prevent this issue by taking features extracted from each of the documents and optimizing suitable antisymmetric features as a part of the network itself during the training process. This is achieved by using the structure depicted in Fig. 1.

¹ Throughout this work, the term *antisymmetry* is to be understood in the numerical sense, if not stated otherwise.

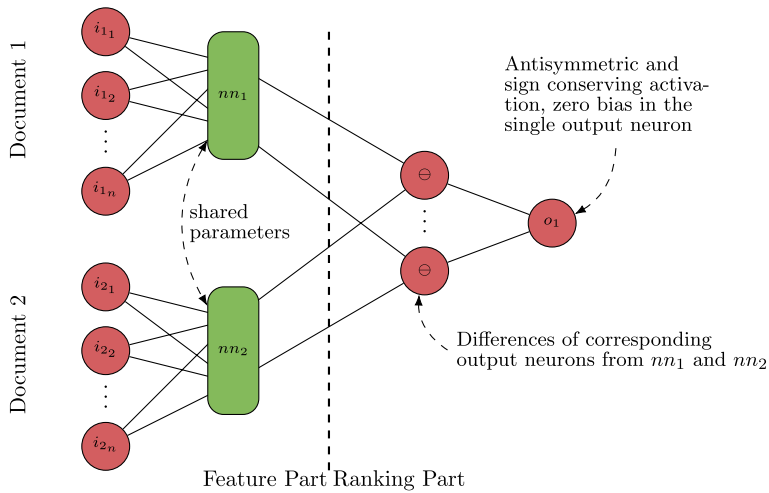


Fig. 1 Schema of RankNet. nn_1 and nn_2 can represent arbitrary networks (or other universal function approximators) as long as they produce the same output for the same inputs i_j . The bias of the output neuron o_1 has to be zero and the activation antisymmetric and sign conserving

The corresponding features of two documents are fed into the two submodels nn_1 and nn_2 , respectively. These models can in principle be of arbitrary kind and structure, yet they have to be identical, i.e. share the same structure and parameters like weights, biases, activation, etc. in the case of neural networks. The difference of the submodels' outputs is fed into a third part, which consists exclusively of one output neuron with antisymmetric activation and without a bias, representing the above defined function r .

Throughout this paper, we use neural networks for the submodels nn_1 and nn_2 . This, however, is not a necessary restriction to this model. As long as these submodels produce values in $\mathbb{R}^n$, the architecture still ensures that the resulting model complies with the criteria (I)–(III). Nonetheless, in the following we will often make this restriction to discuss cases in which neural networks can be shown to be capable of approximating an order embedding arbitrarily well given enough complexity. With the following lemma we show this network satisfies conditions (I) through (III):

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

Proof Let the activation function of the output neuron o_1 be $\tau : \mathbb{R} \rightarrow \mathbb{R}$ with $\tau(-x) = -\tau(x)$ and $\text{sign}(\tau(x)) = \text{sign}(x)$ as required.

- (II) The two models nn_1 and nn_2 are identical. Hence, they implement the same function $f : \mathcal{F} \rightarrow \mathbb{R}^n$. The output of the complete network for the two input vectors $x, y \in \mathcal{F}$ is then given by:

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

where w is a weight vector for the output neuron, $\langle \cdot, \cdot \rangle$ denotes the standard scalar product on $\mathbb{R}^n$, and $g : \mathcal{F} \rightarrow \mathbb{R}$. Furthermore, we can write for the output neuron:

$$o_1 =: \tau[g(x) - g(y)]. \quad (3)$$

The antisymmetry for x and y can be shown with

$$r(x, y) = \tau[g(x) - g(y)] = -\tau[g(y) - g(x)] = -r(y, x), \quad (4)$$

thus satisfying the second condition (II).

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

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

- (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 Eq. 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

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

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

□

Note that the requirements (I)–(III) above are stricter than necessary. In fact, any function $f : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$, for which a homeomorphism $h : \mathbb{R} \rightarrow \mathbb{R}$ exists, such that $r = h \circ f$ fulfills these requirements, has the same implications that we discuss below iff $f(x, x) = h^{-1}(0)$ for all $x \in \mathcal{F}$. In this regard, one could also use a sigmoid activation function for the last neuron, as this can be mapped to a tanh via $h(x) = 2x - 1$, which fulfills the requirements (I)–(III). For the sake of brevity and simplicity of the proofs, however, we assume the stricter requirements in the following without loss of generality. Assuming the existence of such a homeomorphism, it is straightforward to show that the results in this paper are also true for different activation functions.

Note, furthermore, that if a total quasiorder $\geq$ on $\mathcal{F}$ is represented by the ranking function $r(x, y) = \tau[g(x) - g(y)]$ as above, then $g : \mathcal{F} \rightarrow \mathbb{R}$ is an order embedding $(\mathcal{F}, \geq) \rightarrow (\mathbb{R}, \geq)$ (Davey and Priestley, 2002; Cohen, 2007). If, however, $\geq$ is only a partial order, RankNet will always extrapolate it to a total order.

If the order of documents belonging to the same relevance class is not restricted, the properties discussed in Lemma 1 offer advantages during the training phase of RankNet in distinguishing between 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 unnecessary to compare extremely relevant and extremely irrelevant documents directly during training. Every document passed to RankNet should at least be compared with documents from the corresponding neighbouring relevance classes. By using the transitivity of the model, it is implicitly be trained for all combinations, given that all classes are represented in the training data.

- (iii) It might seem to be sensible to train the model such that it is capable to predict the equality of two different documents of the same relevance class. Although the model is actually restricted when doing so: If RankNet 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 RankNet is not relevant for equivalent documents. This leads to the benefit that only document pairs with different classes can be used during training. With this setup the optimizer employed in the training phase enjoys more freedom to achieve an optimal solution for ranking relevant cases. This feature is potentially enhancing the general performance of RankNet.

Note that enhancing the training of RankNet improves the original training of the model (Burges et al., 2005). Furthermore, we do not restrict the models nn_1 and nn_2 to be neural networks. All of these three properties are free parameters in RankNet.

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

In the following, we try putting RankNet on a more sound basis by analyzing some cases in which RankNet 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 RankNet with nn_1 and nn_2 being neural nets, i.e. there is a continuous function $r : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$ 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 (Hornik et al., 1989), the second condition implies that r can be approximated to arbitrary precision by RankNet. In the following, we will discuss interesting cases, in which these assumptions are valid:

Proposition 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 $g : \mathcal{F} \rightarrow \mathbb{N}$. Therefore, choose $x \succeq y \Leftrightarrow g(x) \geq g(y)$ for $x, y \in \mathcal{F}$. $\square$

In fact, every sorting of the elements of countable sets satisfies (A)–(C), and as we show in the following theorem, it can be approximated by RankNet, if the set is normed and is uniformly discrete²:

Theorem 3 *Let $\succeq$ implement (A)–(C) on an uniformly discrete metric feature space $\mathcal{F} \subset \mathbb{R}^n$. Then, RankNet can approximate a function that represents $\succeq$.*

² A uniformly discrete metric feature space $\mathcal{F}$ is a feature space with a metric d for which there is a $r > 0$ such that for all $x, y \in \mathcal{F}$, $x \neq y$ the condition $d(x, y) > r$ holds.

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 (Hornik et al., 1989), it is sufficient to show that a continuous function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 $\exists j : j < i \wedge x_i \geq x_j \wedge x_j \geq x_i$, set $g(x_i) := g(x_j)$ and continue with the next element.
 p1.2 If $\forall j$ with $j < i : x_i \geq x_j$, set $g(x_i) := \max_{j < i} g(x_j) + 1$ and continue with the next element.
 p1.3 If $\forall j$ with $j < i : x_j \geq x_i$, set $g(x_i) := \min_{j < i} g(x_j) - 1$ and continue with the next element. If there are $j, k < i$ with $x_k \geq x_i \geq x_j$, choose an arbitrary “largest element smaller than x_i ”, i.e. an element $x_l \in \mathcal{F}, l < i$ satisfying $x_i \geq x_l \geq x \forall x \in \{x_j \in \mathcal{F} | j < i, x_j \not\geq x_i\}$, and an arbitrary “smallest element larger than x_i ”, i.e. an element $x_g \in \mathcal{F}, g < i$ such that $x \geq x_g \geq x_i \forall x \in \{x_k \in \mathcal{F} | k < i, x_i \not\geq x_k\}$. 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 $g(x) \geq g(y) \Leftrightarrow x \geq y \forall x, y \in \mathcal{F}$ and thus $r(x, y) = g(x) - g(y) \geq 0 \Leftrightarrow x \geq y$.

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 $\tilde{B}_i : \overline{B_{\delta/42}(x_i)} \rightarrow \mathbb{R}, x \mapsto 1 - \frac{42|x - x_i|}{\delta}$. $\tilde{B}_i$ 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 $g_c : \mathbb{R}^n \rightarrow \mathbb{R}, x \mapsto \sum_{i=1}^{\infty} g(x_i) B_i(x)$ 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 $\geq$ on $\mathcal{F}$. B_i is continuous since $B_i|_{\overline{B_{\delta/42}}} = \tilde{B}_i$ and $B_i|_{\mathbb{R}^n \setminus \overline{B_{\delta/42}}} \equiv 0$ 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. $\square$

Therefore, it is theoretically feasible to successfully train RankNet 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 RankNet 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 RankNet in the form of a neural network can approximate $\succeq$ if $\mathcal{P}_x$ and $\mathcal{N}_x$ are open for all $x \in \mathcal{F}$.

Proof First, note that $\partial_x = \mathcal{F} \setminus (\mathcal{P}_x \cup \mathcal{N}_x)$ and that $\mathcal{P}_x \cap \mathcal{N}_x = \emptyset$ because of the antisymmetry of $\succeq$, dividing $\mathcal{F}$ into three distinct subsets $\mathcal{P}_x, \mathcal{N}_x, \partial_x$. The relation $x \sim y \Leftrightarrow y \in \partial_x$ defines an equivalence relation with equivalence classes $[x] = \partial_x$.

According to the universal approximation theorem it is sufficient to show that a continuous function $g : \mathcal{F} \rightarrow \mathbb{R}$ exists such that $\forall x, y \in \mathcal{F} : x \succeq y \Leftrightarrow g(x) \geq g(y)$. Again this will be done by explicit construction:

Now, define $g : \mathcal{F} \rightarrow \mathbb{R}$ by first choosing an arbitrary $x_0 \in \mathcal{F}$. For all points $x \in \mathcal{F}$, set

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

First, let us show, that g is continuous:

Consider $d_{\partial_{x_0}} : (\mathcal{F}/\sim) \rightarrow \mathbb{R}, \partial_x \mapsto d(\partial_{x_0}, \partial_x)$. Since d as a metric is continuous, we know that $d_{\partial_{x_0}}$ is, too. Thus, for every $\partial_x \in \mathcal{F}/\sim$ and every $\varepsilon > 0$ there is a $\delta > 0$ such that for every ∂_y with $d(\partial_x, \partial_y) < \delta$ $|d_{\partial_{x_0}}(\partial_x) - d_{\partial_{x_0}}(\partial_y)| < \varepsilon$. Therefore, for a given $\varepsilon > 0$ and $x \in \mathcal{F}$ we can always choose δ such that for every $y \in \mathcal{F}$ with $|x - y| < \delta$ the above holds for ∂_x and ∂_y . Then, $||g(x)| - |g(y)|| = |d_{\partial_{x_0}}(\partial_x) - d_{\partial_{x_0}}(\partial_y)| < \varepsilon$. Thus, $|g|$ is continuous, which means that g is continuous on $\mathcal{P}_{x_0} \cup \partial_{x_0} = \mathcal{F} \setminus \mathcal{N}_{x_0}$ and $\mathcal{N}_{x_0} \cup \partial_{x_0} = \mathcal{F} \setminus \mathcal{P}_{x_0}$. Since these two sets are closed in $\mathcal{F}$, this implies that g is continuous on their union $\mathcal{F} = \mathcal{P}_{x_0} \cup \partial_{x_0} \cup \mathcal{N}_{x_0}$.

Finally, we need to show now that $r(x, y) = g(x) - g(y)$ represents $\succeq$, i.e. $x \succeq y \Leftrightarrow g(x) \geq g(y)$:

For the case $x \succeq x_0 \succeq y$ the equivalence is obvious from the definition of g . For the remaining cases concerning $x \succeq y \Rightarrow g(x) \geq g(y)$, suppose there are elements $x, y \in \mathcal{P}_{x_0}$ with $x \succeq y$ and $g(x) < g(y)$. We can then choose a continuous curve $\gamma : [0, 1] \rightarrow \mathcal{F}$ with $\gamma(0) = x_0 \in \mathcal{N}_y$ and $\gamma(1) = x \in \mathcal{P}_y \cup \partial_y$. Choose $t_0 := \sup\{t \in [0, 1] | \gamma(t) \in \mathcal{N}_y\}$. Then, for every neighborhood U of t_0 , there are $t', t'' \in U$, such that $\gamma(t') \in \mathcal{N}_y$, $\gamma(t'') \notin \mathcal{N}_y$. Therefore, $\gamma(t_0)$ is a boundary point of $\mathcal{N}_y$ and therefore no element of $\mathcal{N}_y$. Also,

since $\mathcal{P}_y \cap \mathcal{N}_y = \emptyset$, $\gamma(t_0)$ is no inner point of P_y and therefore no element of P_y . Thus $\gamma(t_0) =: \tilde{y} \in \mathcal{F} \setminus (\mathcal{P}_y \cup \mathcal{N}_y) = \partial_y$. If $y \geq x$, t_0 may equal 1.

Since this holds for any continuous curve γ with the given boundary conditions, we can choose $\gamma(t) = tx + (1-t)x_0$. The length of $\gamma|_{[0,t_0]}$ is then given by

$$|\tilde{y} - x_0| = L(\gamma|_{[0,t_0]}) = \int_0^{t_0} |\dot{\gamma}(t)| dt \leq \int_0^1 |\dot{\gamma}(t)| dt = |x - x_0|$$

The same argument holds when replacing x and x_0 by arbitrary $x' \in \partial_x$ and $x'_0 \in \partial_{x_0}$. Therefore, $\forall x' \in \partial_x, x'_0 \in \partial_{x_0} \exists \tilde{y}_{x',x'_0} \in \partial_y : |y_{x',x'_0} - x'_0| \leq |x' - x'_0|$, and therefore

$$\begin{aligned} g(y) &= d(\partial_y, \partial_{x_0}) = \inf_{y' \in \partial_y, x'_0 \in \partial_{x_0}} |y' - x'_0| \\ &\leq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0| = d(\partial_x, \partial_{x_0}) = g(x), \end{aligned}$$

which contradicts the assumption that $g(x) < g(y)$. Following a similar chain of reasoning, the case $g(x) < g(y)$ for $x, y \in \mathcal{N}_{x_0}$ leads to contradictions.

To see that $g(x) \geq g(y) \Rightarrow x \geq y$, suppose $\exists x, y \in \mathcal{F}, g(x) \geq g(y), x \not\geq y$. Because of the antisymmetry of $\geq$, this implies $y \geq x$, which (as shown above) leads to $g(y) \geq g(x)$. This, however, is only possible if $g(x) = g(y)$.

Analogously to the above, for every curve $\gamma(t) = tx' + (1-t)x'_0$ for $t \in [0, 1]$ with $x' \in \partial_x, x'_0 \in \partial_{x_0}$, there exists a $\tilde{y}_{x',x'_0} \in \partial_y$ and $t_0 \in]0, 1[$ such that $\gamma(t_0) = \tilde{y}_{x',x'_0}$. We also know that

$$\begin{aligned} |x' - x'_0| &= \int_0^1 |\dot{\gamma}(t)| dt \\ &= \int_0^{t_0} |\dot{\gamma}(t)| dt + \int_{t_0}^1 |\dot{\gamma}(t)| dt = |\tilde{y}_{x',x'_0} - x'_0| + |x' - \tilde{y}_{x',x'_0}| \end{aligned}$$

However, $g(x) = g(y) \Leftrightarrow \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0| = \inf_{y' \in \partial_y, x'_0 \in \partial_{x_0}} |y' - x'_0|$ implies

$$\begin{aligned} \inf_{y' \in \partial_y, x'_0 \in \partial_{x_0}} |y' - x'_0| &\leq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |\tilde{y}_{x',x'_0} - x'_0| \\ &\leq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0| \\ &\leq \inf_{y' \in \partial_y, x'_0 \in \partial_{x_0}} |y' - x'_0| \end{aligned}$$

This means that $\inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |\tilde{y}_{x',x'_0} - x'_0| = \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0|$, which implies

$$\begin{aligned} \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0| &= \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} (|\tilde{y}_{x',x'_0} - x'_0| + |x' - \tilde{y}_{x',x'_0}|) \\ &\geq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |\tilde{y}_{x',x'_0} - x'_0| + \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - \tilde{y}_{x',x'_0}| \\ &\geq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - x'_0| + \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - \tilde{y}_{x',x'_0}| \\ &\Rightarrow d(\partial_x, \partial_y) = \inf_{x' \in \partial_x, y' \in \partial_y} |x' - y'| \leq \inf_{x' \in \partial_x, x'_0 \in \partial_{x_0}} |x' - \tilde{y}_{x',x'_0}| \leq 0 \\ &\Rightarrow d(\partial_x, \partial_y) = 0 \end{aligned}$$

Which contradicts the assumption $x \not\preceq y$ that implies $\partial_x \neq \partial_y$. $\square$

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 Theorem 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 Sect. 5.3.

Furthermore, we show in the following that the magnitude of the output of RankNet is a pseudometric. This property is interesting for, e.g., clustering or nearest neighbor algorithms employing the learned pseudometric.

Proposition 5 *The absolute value of the function $r(x, y)$ defines a pseudometric on the feature space $\mathcal{F}$.*

Proof A pseudometric is defined by three conditions: identity of indiscernibles, symmetry and the triangle inequality. The identity of indiscernibles is given by $|r(x, x)| = 0$ and $|r(x, y)| \geq 0$. The symmetry holds since $r(x, y) = -r(y, x) \rightarrow |r(x, y)| = |r(y, x)|$. For the last condition, we start with $|r(x, z)| = |r(x, y) + r(y, z)|$, and since $\tau : \mathbb{R} \rightarrow \mathbb{R}$, we find $|r(x, z)| \leq |r(x, y)| + |r(y, z)|$. $\square$

This leads to the possibility to learn a pseudometric on the feature space, even if only the absolute value of the ranking function $r(x, y)$ is taken. This holds even if there is no ranking function found on the feature space. However, more work is needed to further investigate this in a practical setting, e.g., for clustering.

It should be mentioned that the above discussion is non-exhaustive and that one may encounter cases in which the presented architecture cannot approximate a given quasiorder to the desired precision. The applicability of RankNet to problems, for which the conditions we discuss are not met, usually needs to be considered on a case-by-case basis. In some cases, the learnability of an embedding might be achieved by using non-standard activation functions and, if that is not sufficient, the model might be adapted as discussed in the following.

It was mentioned above that the feature part (nn_1 and nn_2) does not necessarily need to be a neural network, but could rather be any model producing a value in $\mathbb{R}^n$. As we only explore the case of neural networks in the experiments, we want to close this discussion by first giving a hypothetical model different from a neural network as a function approximator, and then giving practical examples of alternative function approximators.

The hypothetical model, given as an example, does not rely on the same kind of feature part of a neural network on a feature space $\mathcal{F} \subseteq \mathbb{R}^n$. Consider the identical models nn_1 and nn_2 to be a polynomial

$$g(x) = \sum_{\alpha \in P_n} a_\alpha \prod_{i=1}^n x_i^{\alpha_i}$$

of n variables and degree D , where $P_n := \{(\alpha_1, \dots, \alpha_n) \in \mathbb{N}_0^n \mid \sum_i \alpha_i \leq D\}$. Furthermore, choose $o_1(\xi) = \tanh(\xi)$ such that the whole model is given by $r(x, y) = \tanh(g(x) - g(y))$. The coefficients a_α could then be fixed similarly to the way described above by minimizing the loss function $L(x, y) = (1 - r(x, y))^2$ for a set of samples with $x \succeq y$.

In general, when using other function approximators for the feature part, an optimization target and an appropriate optimizer for the specific case are needed. Our proposed ranking part then guarantees that a total quasiorder is always present, regardless of the function approximator used. This can be considered as one of our main contributions, generalizing RankNet, which does not restrict itself to differentiable models.

Practically speaking, gradient-based optimizations lend themselves to the integration into this ranking framework. Differentiable decision tree models, such as the Tree Ensemble Layer (TEL) (Hazimeh et al., 2020), could be integrated, e.g., by connecting nm_1 and nm_2 with the output neuron o_1 . While technically not necessary, incrementality of a learning scheme is beneficial: This is naturally given by gradient descent or ascent algorithm, but also by streaming algorithms (Bahri et al., 2021). We conjecture that other optimization techniques, such as sampling-based models or boosting-type methods, can also be employed.

4 Experimental setup

To evaluate the performance of RankNet, and to compare it to other learning to rank approaches, we employ common evaluation metrics and standard datasets (MSLR-WEB10K, MQ2007, MQ2008 (Qin and Liu, 2013)). Furthermore, we use synthetic data to investigate the dependence of the performance on certain characteristics of the data. Reliable estimates for comparing the different methods can be obtained by conducting three distinct experiments. The first experiment involves using predefined folds from various datasets. The second experiment evaluates the models using 15-hold-out splits with binary classes, while the third experiment uses the same 15-hold-out splits with multiple classes. In all tests, two versions of RankNet were utilized: one implemented in TensorFlow version 1, and another implemented in version 2 of the library (Abadi et al., 2015).

For evaluating different models, we apply the commonly used metrics NDCG and MAP as defined in Sect. 4.1 and Sect. 4.2. In Sect. 4.3 we briefly describe the structure of the LETOR datasets and in Sect. 4.4 how the models are evaluated. In Sect. 4.5 we illustrate how the synthetic datasets are generated and analyzed.

4.1 The NDCG metric

In the field of learning to rank, a commonly used measure for the performance of a model is the normalized discounted cumulative gain of top- k documents retrieved (NDCG@ k). This metric is based on the discounted cumulative gain of top- k documents (DCG@ k):

$$\text{DCG}@k = \sum_{i=1}^k \frac{2^{r(d_i)} - 1}{\log_2(i + 1)},$$

where $d_1, d_2, \dots, d_n$ is the list of documents sorted by the model with respect to a single query and $r(d_i)$ is the relevance label of document d_i . The NDCG@ k can be computed by dividing the DCG@ k by the ideal (maximum) discounted cumulative gain of top- k documents retrieved (IDCG@ k), i.e. the DCG@ k for a perfectly sorted list of documents is defined as $\text{NDCG}@k = \frac{\text{DCG}@k}{\text{IDCG}@k}$.

4.2 The MAP metric

A frequently used alternative to the NDCG is the mean average precision (MAP). For this metric, the precision for the top- k documents of query q is introduced. Since this only makes sense for binary classes, a multi-class system has to be binarized such that $r(d_i) = 1$ indicates a relevant and $r(d_i) = 0$ indicates an irrelevant document:

$$P_q@k = \frac{1}{k} \sum_{i=1}^k r(d_i)$$

Now one needs to calculate the average precision over all *relevant* documents:

$$\text{AvgP}_q = \frac{1}{n \cdot P@n} \sum_{k=1}^n r(d_k) P_q@k,$$

where n is the number of documents in the query q . Finally, the MAP is given by the mean over all queries:

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^Q \text{AvgP}_q,$$

where Q is the number of queries in a data set.

4.3 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 represent 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 on the test set using an appropriate threshold (Ibrahim and Landa-Silva, 2017, 2018). In our case, we map relevance labels ≥ 2 to 1 and relevance labels ≤ 1 to 0. We employ this approach to compare RankNet 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.

4.4 Grid search and LETOR evaluation

We conducted a grid search for hyperparameter optimization of all models (all hyperparameters are shown in Table 4 of the appendix). We carried out three experiments. In the first experiment, we evaluated different models on the five predefined folds of each of the three datasets. For each hyperparameter setting, we performed 5-fold cross-validation

(internal) on each training set. The best hyperparameters were used to train the models again on the entire training set, before computing the performance on independent test sets across all 5 folds (external). In addition to the first experiment, we conducted two other tests: one with binarized classes in the training set, and another without. For these tests, we used the corrected t -test by Nadeau and Bengio (2003) and evaluated the models on a 15-fold-out split without using the predefined folds. For each hold-out split, we evaluated the best hyperparameters using a 3-fold internal cross-validation.

To benchmark our results, we utilized commonly used *learning to rank* algorithms, namely AdaRank (Xu and Li, 2007), LambdaMART (Wu et al., 2010), and ListNet (Cao et al., 2007). We obtained the implementation of these algorithms from the RankLib v2.1 library in the Lemur Project (Croft, 2001). However, due to the limitations in testing flexibility and comparable hyperparameters with the RankLib implementation, we re-implemented the different models and integrated them into our Python framework.³ For the AdaRank algorithm, we improved the computation time by improving the original algorithm of the paper (see Algorithm A).

For the cross-validation of all Python models, we used the *GridSearchCV* class implemented in the scikit-learn 0.24.2 library (Pedregosa et al., 2011). In contrast, for the RankLib models, we manually created different run-scripts to evaluate the models with different hyperparameters. Before training the model, the data was transformed in such a way that the features are following a normal distribution with standard deviation of $1/3$.

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 $\text{NDCG}@k$ is evaluated during the tests.

4.5 Synthetic data generation and evaluation

To study how the optimizations, RankNet, 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 that 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 $r(d_i) - r(d_j) = 1$.

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

³ For our implementation of the model and the tests see <https://github.com/kramerlab/direct-ranker>. For the LambdaMart implementation, we adapted the implementation from <https://github.com/discobot/LambdaMart>. For the AdaRank implementation we optimized parts using the Rust programming language (Matsakis and Klock, 2014).

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 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) RankNet is trained on the generated training set using our framework.
- (3) The trained RankNet 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. 4a–d), 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 Sect. 5.3.

4.6 Training size and label-ratio dependency

To further study the performance of RankNet compared to other ranking methods, two dataset properties are explored in more detail. For this study we employed synthetic data, generated with the same procedure as described in Sect. 4.5, and the MSLR-WEB10K dataset. To compare the performance of RankNet, we evaluated the tests on the ranking methods ListNet, RankNet and LambdaRank. All four models are implemented using our framework.

To explore training set size dependency, we increased the number of training samples from 100 to 10^6 (in the case of the MSLR-WEB10K dataset, the maximum amount of training data was set to 700,000), while keeping 10^4 documents in the test set. The number of relevance classes and the generated features was set to 5 and 70, respectively.

The second test focuses on the ratio of instances with high relevance compared to ones with lower relevance. For this test, the training set size was set to 10^5 , and the number of classes was set to 6. The rest of the dataset was kept the same as for the training set size test. We reduced the number of documents with relevance classes 5, 4 and 3 by drawing without replacement, while the other three classes were kept the same. For the MSLR-WEB10K dataset, the more relevant classes are chosen to be 4, 3 and 2, while the less relevant ones are 1 and 0. Compared to the synthetic data, the number of instances per relevance class is different. To overcome this, we reduced the number of instances per class according to their absolute frequency in the whole dataset.

5 Experimental results

In this section, we present the results of our experiments. Specifically, we compare RankNet with frequently used *learning to rank* algorithms in Sect. 5.1. In Sect. 5.2, we analyzed the effect of different activation functions and optimizers. We also investigate the

Table 1 Performance comparison for different rankers on multiple LETOR datasets

	Algorithm	MSLR-WEB10K		MQ2008		MQ2007	
		$\langle \text{NDCG} \rangle$	$\langle \text{MAP} \rangle$	$\langle \text{NDCG} \rangle$	$\langle \text{MAP} \rangle$	$\langle \text{NDCG} \rangle$	$\langle \text{MAP} \rangle$
5 Fold LETOR folds	RankNet* [†]	0.440(4)	0.365(3)	0.720(12)	0.636(11)	0.540(10)	0.534(9)
	ListNet [‡]	0.157(3)	0.192(2)	0.719(10)	0.647(6)	0.526(10)	0.525(9)
	RankNet [‡]	0.157(3)	0.195(2)	0.716(11)	0.642(10)	0.525(11)	0.525(7)
	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)
	<i>ES-Rank</i>	<i>0.382</i>	<i>0.570</i>	<i>0.507</i>	<i>0.483</i>	<i>0.451</i>	<i>0.470</i>
	<i>IESR-Rank</i>	<i>0.415</i>	<i>0.603</i>	<i>0.517</i>	<i>0.494</i>	<i>0.455</i>	<i>0.473</i>
	<i>IESVM-Rank</i>	<i>0.224</i>	<i>0.457</i>	<i>0.498</i>	<i>0.473</i>	<i>0.436</i>	<i>0.456</i>
15 Hold-out binary-class	RankNet* [†]	0.422(8)	0.380(6)	0.448(25)	0.390(26)	0.526(6)	0.457(6)
	RankNet [†]	0.362(15)	0.332(11)	0.384(50)	0.321(51)	0.476(38)	0.406(35)
	ListNet* [†]	0.423(4)	0.381(4)	0.449(12)	0.391(14)	0.525(6)	0.456(5)
	ListNet [†]	0.363(23)	0.333(17)	0.381(44)	0.318(46)	0.449(40)	0.382(39)
	LambdaMart [†]	0.212(9)	0.154(8)	0.342(29)	0.229(23)	0.403(15)	0.281(16)
	AdaRank [†]	0.351(4)	0.320(7)	0.440(57)	0.356(104)	0.515(29)	0.443(51)
	RankNet* [†]	0.349(10)	0.324(8)	0.469(14)	0.409(16)	0.508(7)	0.436(7)
multi-class	RankNet [†]	0.327(7)	0.305(5)	0.358(42)	0.294(44)	0.438(38)	0.372(35)
	ListNet* [†]	0.390(9)	0.354(7)	0.449(15)	0.390(18)	0.512(22)	0.442(24)
	ListNet [†]	0.377(10)	0.344(9)	0.412(34)	0.347(35)	0.461(40)	0.394(39)
	LambdaMart [†]	0.209(4)	0.153(8)	0.359(31)	0.244(26)	0.409(22)	0.288(22)
	AdaRank [†]	0.351(4)	0.315(7)	0.447(45)	0.376(80)	0.520(5)	0.454(5)

We performed three tests. The first one was done on the predefined five folds from the LETOR datasets. The other two tests were done by performing the Nadeau and Bengio correction of dependent Student's t-test using 15 hold-out splits. One was done binarized classes the other one with the predefined classes of the datasets. The values with $\ddagger$ were trained using RankLib while the values with $\dagger$ are trained within our Python framework. The values for ES-Rank, IESR-Rank and IESVM-Rank are taken from the literature (Ibrahim and Landa-Silva, 2018), since there is currently no implementation available. These values are shown in *italics*

sensitivity of RankNet to different dataset properties in Sects. 5.3 and 5.4. Finally, in Sect. 5.5, we briefly describe a method for using pairwise approaches for classification and discovering potential new subclasses.

5.1 Comparison to other rankers

Table 4 of the appendix shows the hyperparameters used in our experiments. We found that using an activation function of **tanh** for the feature and ranking parts of RankNet resulted in good performance across all experiments. The size of the network was also found to be an important hyperparameter, with a small network without dropout and weight regularization or a large network with dropout and weight regularization performing the best. Surprisingly, we found that in contrast to the originally used cost function the $\mathcal{L}_2$ cost function performed the best, even though it does not directly optimize a *listwise* metric like the NDCG. Finally, we observed that RankNet converges to optimal performance in a smaller

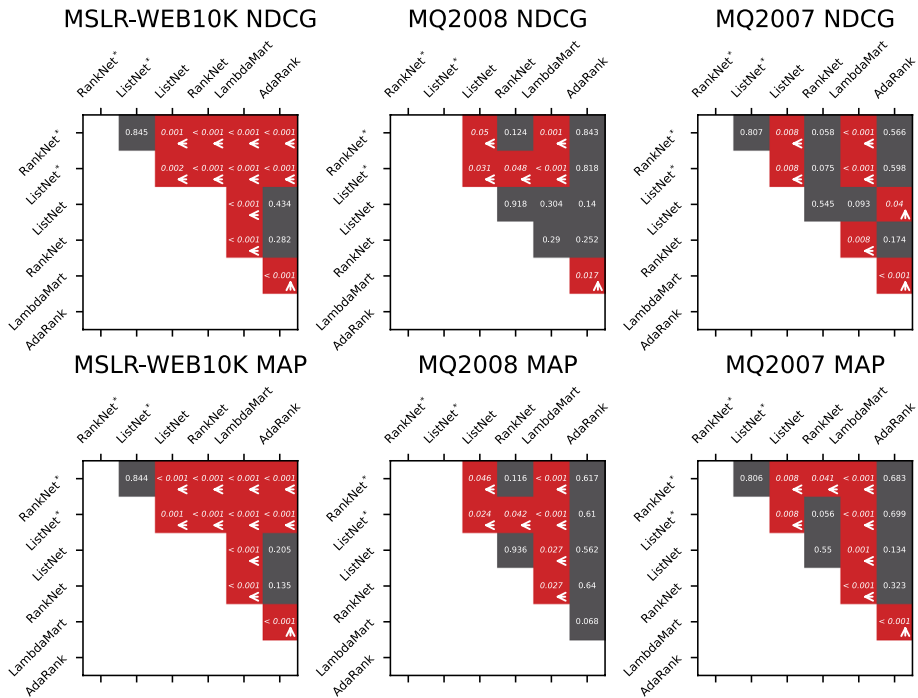


Fig. 2 p values for the corrected Student's t-test with binary classes. Values with a significant p -value are marked with red background and italic font while the non-significant ones are marked with gray background. If there is a significant difference the arrow points to the model with better performance

number of training steps or epochs than the other algorithms we tested, which is an interesting result that has also been observed in other studies (Reimers and Gurevych, 2019).

In Table 1, we present the results of different models on the LETOR datasets discussed in Sect. 4.3. We conducted a Nemenyi-Test (Peter, 1963) followed by a Friedman-Test (Friedman, 1937) on the five predefined folds of the LETOR datasets to compare all the different pairs of algorithms. However, the only significant difference we observed was between RankNet and IESVM-Rank. Upon further investigation, we found that the variances of the performance measures were similar across all the models for a single dataset, which could have made it difficult to detect significant differences in the tests.

To investigate this further, we conducted two additional tests: one training all models with binary classes (for MSLR-WEB10K we set the relevance classes 4,3,2 to 1 and 1,0 to 0; for the other two datasets we set the relevance classes 2,1 to 1 and 0 to 0) and the other training with multiple classes. We trained and evaluated all models on 15 hold-out splits, following the methodology proposed by Nadeau and Bengio (2003). The hold-out split method can reduce the variance of the performance measures by using different subsets of the data for training and testing. This reduces the dependence of the performance measures on specific instances in the data, resulting in a more reliable and accurate estimate of the methods' performance. It may also reveal significant differences that were not detected by the Nemenyi-Friedman-Test using the predefined folds. Additionally, to ensure a fair and consistent evaluation, as explained in Sect. 4.4, we re-implemented the benchmark models and included them into our Python framework.

In Fig. 2, we present the p -values obtained using the corrected Student's t-test by Nadeau and Bengio (2003) for the binary-class test. For all three datasets, RankNet and ListNet consistently outperforms the other methods across both metrics. LambdaMart performs significantly worse than all the other methods for the MSLR-WEB10K dataset and performs worse than the other methods in terms of the MAP metric for the other two datasets.

In Fig. 3, we present the p -values obtained for the multi-class test. The results are somewhat more varied in comparison to the binary-class test. While RankNet still outperforms LambdaMart across all three datasets, ListNet exhibits better performance than RankNet on the MSLR-WEB10K dataset. Additionally, ListNet emerges as the top-performing method on the MSLR-WEB10K and MQ2007 dataset, while AdaRank surpasses all other methods on the MQ2007 dataset. LambdaMart, however, continues to perform significantly worse than all other methods in most of the tests.

The results showed that RankNet method performed mostly better than the other methods over most of the datasets and evaluation metrics. However, we acknowledge that our study has some limitations. Firstly, the use of only three datasets may not be sufficient to generalize the findings to other datasets or experimental setups. Secondly, while the 15 hold-out splits method can reduce the variance of performance measures, it may still suffer from overfitting and not capture the true generalization ability of the methods. Finally, although we have re-implemented the benchmark models in Python, there may still be differences in the implementation details that could affect the results.

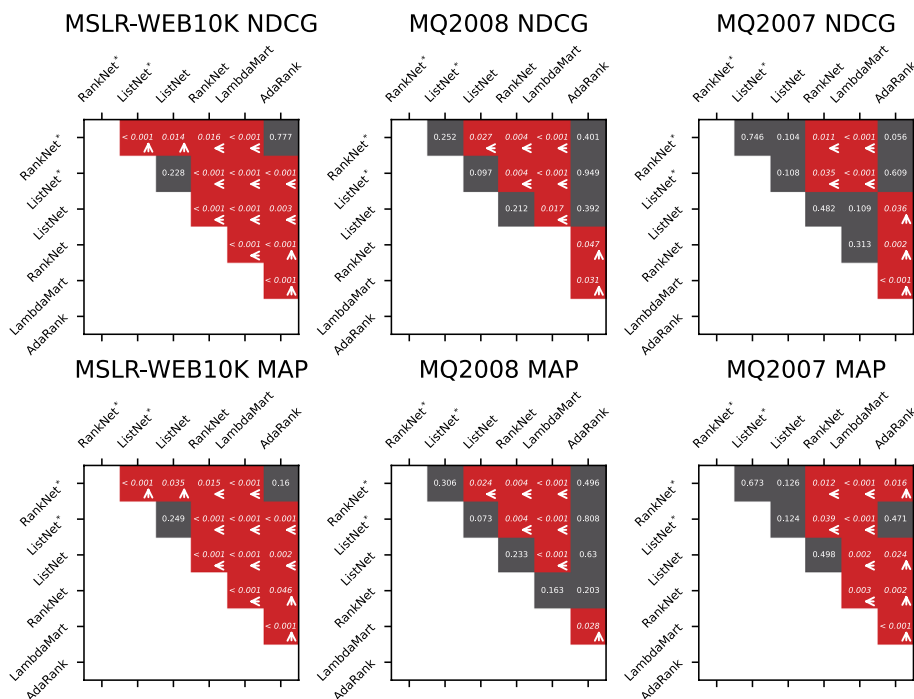


Fig. 3 p values for the corrected Student's t-test with multi classes. Values with a significant p -value are marked with red background and italic font while the non significant once are marked with gray background. If there is a significant difference the arrow points to the model which is better

5.2 Analysis of optimizer and activation function

To highlight the main influences of the performance, we conducted an additional analysis of RankNet by varying the activation function in the last neuron, as presented in Table 2. The models had the same architecture of three hidden layers with the numbers of neurons set to 32-20-5. We compute the NDCG@10 over the five predefined folds of the dataset. The results indicate that the choice of optimizer has a significant impact on model performance, while the use of the activation function does not significantly improve the method.

5.3 Sensitivity on dataset properties

With the following tests we discuss how RankNet performs under different circumstances. The tests were performed as described in Sect. 4.5. The performance of RankNet was tested for different numbers of relevance classes (Fig. 4a), features (Fig. 4b), for variations of noise on the class labels (Fig. 4c), and differently sized training sets (Fig. 4d).

The tests show that, given enough data, RankNet is able to handle a diverse range of datasets. It especially shows that the RankNet can handle many relevance classes as shown in Fig. 4a. 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. 4d) or the number of features (Fig. 4b). Additionally, Fig. 4c shows the robustness of RankNet 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 increasing noise. Still, even with 50% of the documents being mislabeled (i.e., $\sigma = 0.75$), the NDCG@20 does not drop below 0.80. This suggests that the theoretical findings in Sect. 3 for ideal data stay valid for real-world data.

5.4 Training size and label-ratio dependency

At this point we discuss how RankNet performs compared to other ranking models depending on specific dataset characteristics. The test procedure was explained in Sect. 4.6. For the tests, the performance based on differently sized training sets and with different label ratios between relevant and non-relevant classes is evaluated.

The results for the training size dependency show that RankNet is able to achieve a better performance than other ranking models containing fewer instances in the training set. This can be seen for synthetic data in Fig. 5a but also for the MSLR-WEB10K dataset in Fig. 5b.

Table 2 Performance comparison for different ranking activation functions for the RankNet architecture trained on the MSLR-WEB10K dataset

Model	Optimizer	Sigmoid	Tanh	Linear
RankNet	Gradient descent	0.187(6)	0.194 (16)	0.194 (10)
RankNet*	Adam	0.299 (9)	0.301 (13)	0.309 (6)

The models had the same architecture of 3 hidden layers (# neurons 32-20-5) and the implementation was done using Tensorflow v2.11.0. We compute the NDCG@10 over the five predefined folds of the dataset

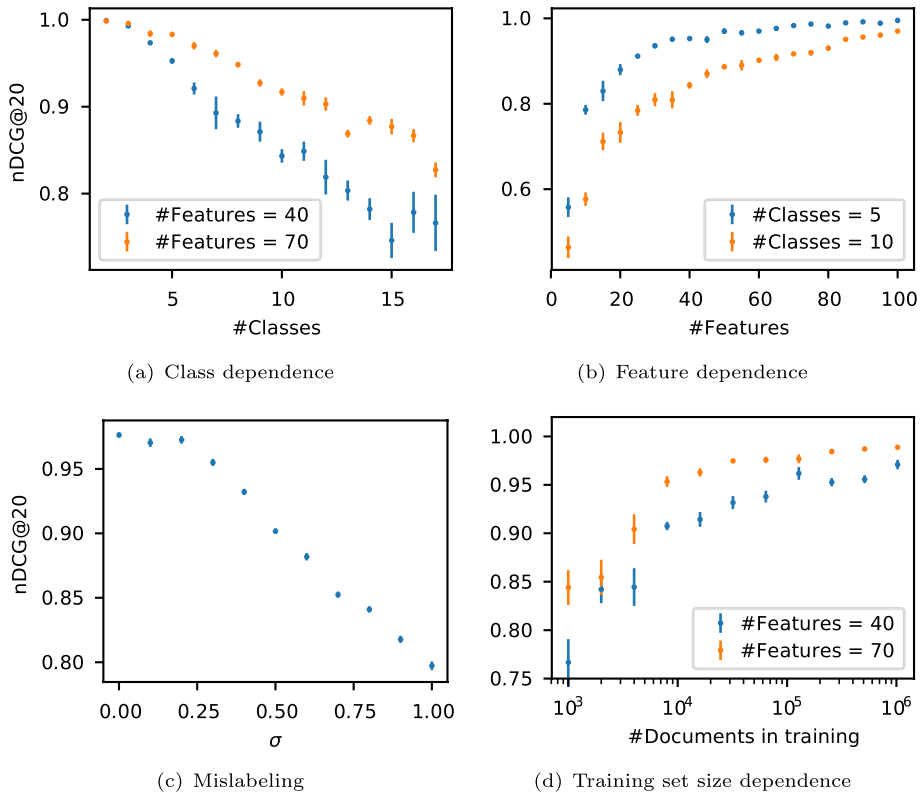


Fig. 4 Plots depicting the sensitivity of RankNet* performance on certain dataset properties, evaluated on synthetic data as explained in Sect. 4.5. In **a** the dependency on the number of relevance classes is changed, while the number of documents in the training set was fixed to 10^5 . In **b** the dependency on the number of features is shown. In this test, the number of documents in the training set was 10^5 and the number of relevance classes was set to five. In **c** the performance of RankNet* with different noise levels on the class labels with 5 classes and 70 features is displayed. **d** shows the results when the size of the training set with five relevance classes is varied

Looking at the results for different label-ratios, RankNet is better than the other ranking models considering the results on synthetic data in Fig. 5c. For the results on the MSLR-WEB10K dataset, ListNet performs the best for the lowest ratio, while for higher ratios the performance of RankNet is constantly increasing and around 10^{-2} the method outperforms the other ranking models.

5.5 Comparing neighbors in a sorted list

Additionally to standard ranking, we suggest that RankNet can be used for classification as well. Figure 6a depicts the output of RankNet for successive instance pairs inside a sorted list of synthetically generated data. Evidently, the output peaks at locations where relevance classes change. Classes can therefore be separated by finding these peaks, even on unlabeled

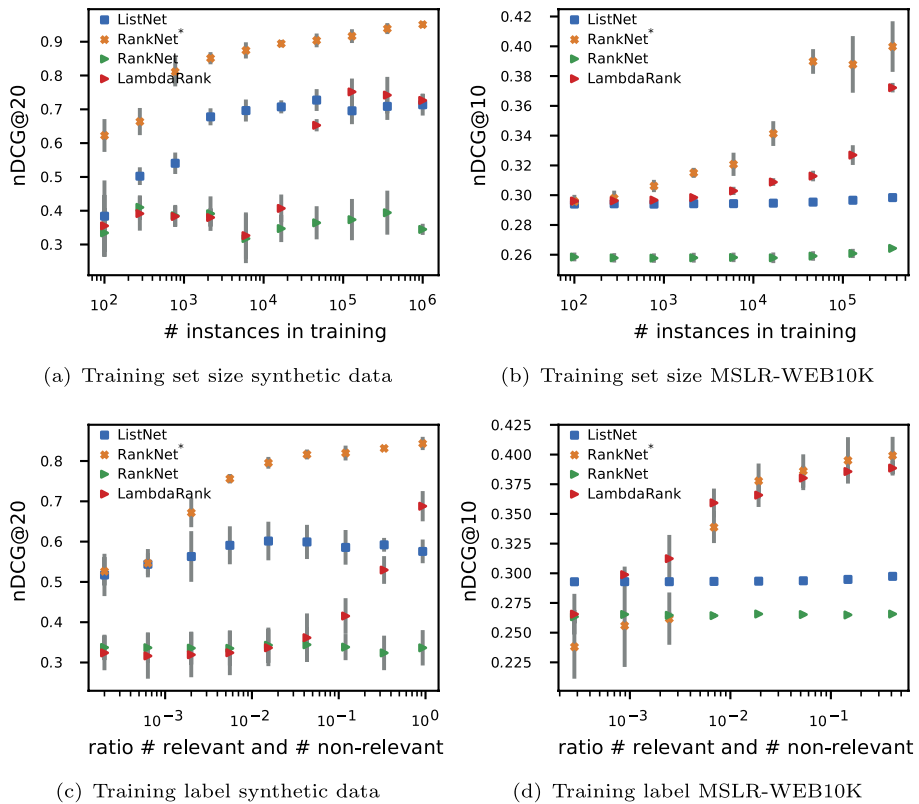


Fig. 5 Training set size and label-ratio dependency for different ranking algorithms. In Fig. 5a and b, the dependency of the performance on the training set size is shown for Ranknet*, RankNet, LambdaRank and ListNet (on synthetic data and on the MSLR-WEB10K dataset). In Fig. 5c and d, the dependency of the performance on the label-ratio dependency is shown, again on synthetic data and on the MSLR-WEB10K dataset. All models in this study were implemented using Tensorflow v2.11.0

data. The same effect can be observed for the MSLR-WEB10K dataset. In Fig. 6b RankNet was trained using all five relevance classes, in Fig. 6c the training was done by only using binarized classes. In both of these cases we can again observe the peaks in the RankNet output separating the classes, even though the effect is less pronounced compared to the synthetic data. It is remarkable that even in the binarized case the peaks separate the unbinarized relevance classes of the test set relatively well.

It may be feasible to train a classifier by using the feature part of RankNet and comparing pairs of instances with each other during training. The trained classifier would be obtained by connecting o_1 directly to m_1 keeping the weights. The discussed peaks can be used to find a threshold in o_1 separating the classes independent of knowledge about the true label.

This approach could be advantageous on datasets with small classes, because the large number of ways to combine two instances of different classes boosts the statistics.

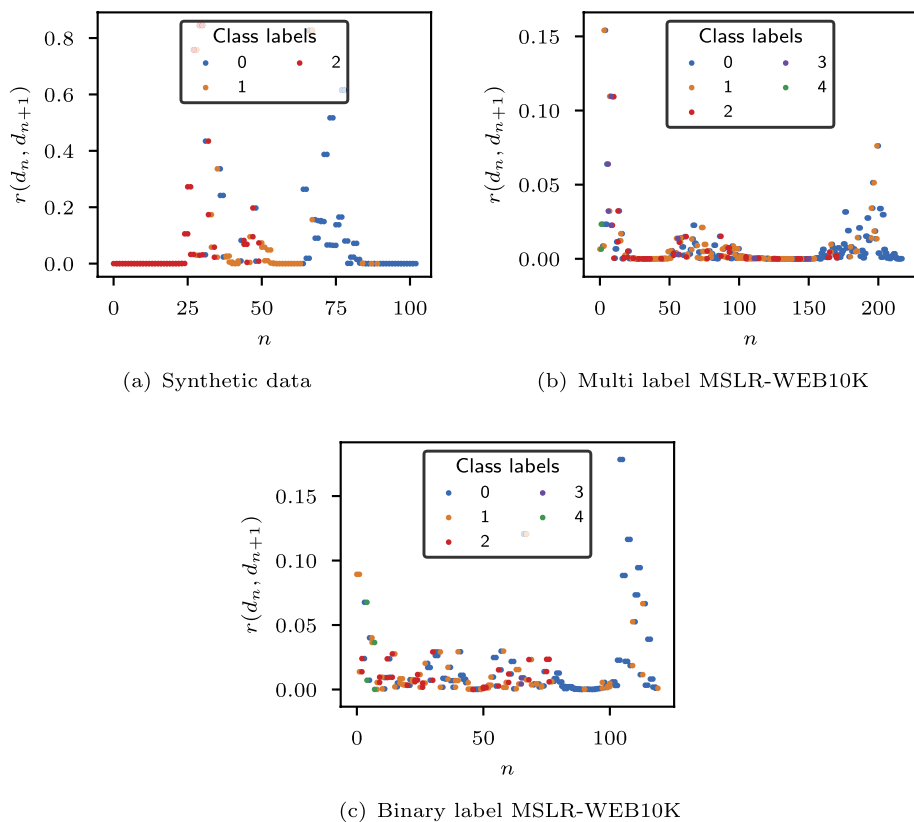


Fig. 6 Output $r(x, y)$ of RankNet* of two successive documents d_n, d_{n+1} inside a list sorted by RankNet*. In Fig. 6a the results are shown for synthetic data. In Fig. 6b and b the results for two different queries of the MSLR-WEB10K dataset are shown. In Fig. 6b the training data had unbinarized relevance classes, while in Fig. 6b the five relevance classes are binarized

6 Discussion and conclusions

The scheme for network structures proposed and analyzed in this paper is an optimization and generalization of RankNet. We show which properties 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 a total quasiorder 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. Our study provides insights into the performance of different learning to rank methods on the LETOR datasets, and suggests that the RankNet method is still a promising approach for improving the ranking performance. Another intriguing possibility would be to adapt some of the ideas of LambdaRank and LambdaMART for listwise optimization to the RankNet model.

In addition, it is remarkable that such a simple and transparent approach can match the performance of more recent and much more complex models, like ES-Rank. Experiments with synthetic data indicate 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 prediction performance can be compensated by the other factors. Furthermore, RankNet keeps its performance if the number of documents in the training set is minimal compared to the test set. The performance was also high if the number of relevant classes in the training set is only a fraction of the number of irrelevant ones. Additionally to standard ranking, we suggest RankNet can be used for classification as well. First tests showed promising results. A more systematic investigation of this is the subject of future work.

When utilizing different function approximators for the feature part, specific optimization targets and suitable optimizers are necessary. Our proposed ranking part ensures the presence of a total quasiorder, irrespective of the function approximator employed. This is an interesting insight, as it extends RankNet beyond neural networks.

Appendix A AdaRank runtime improvement

Algorithm 1 Runtime improvement of the original AdaRank algorithm by precomputing the weak rankers. S is the dataset with q are the queries, d the documents with k features x , y the relevance class of the document, m the number of queries, h the unweighted weak ranker and π the ranking function

Require: dataset $S = \{(q_i, d_i, y_i)\}_{i=1}^m$, performance measure function E and rounds T

$P_1(i) \leftarrow 1/m$

$h \leftarrow \sum_{i=1}^m E(\pi(q_i, d_i, x_k), y_i)$ ▷ Improvement

for $t = 1, \dots, T$ **do**

get weak ranker h_t by $\max_k \{P_t(i) \cdot h\}$.

choose $\alpha_t = \frac{1}{2} \cdot \ln \frac{\sum_{i=1}^m P_t(i) \{1 + E(\pi(q_i, d_i, h_t), y_i)\}}{\sum_{i=1}^m P_t(i) \{1 - E(\pi(q_i, d_i, h_t), y_i)\}}$.

create current ranking model $f_t(\vec{x}) = \sum_{k=1}^t \alpha_k h_k(\vec{x})$.

update $P_{t+1}(i) = \frac{\exp - E(\phi(q_i, d_i, f_t), y_i)}{\sum_{j=1}^m \exp - E(\phi(q_j, d_j, f_t), y_j)}$.

end for

Appendix B Run time studies

To demonstrate the efficiency of our framework, we present experiments on the runtime for the model training in Table 3. All tests performed by us have been conducted on an Intel®Core™ i7-6850K CPU @ 3.60GHz using the above mentioned MSLR-WEB10K dataset averaging over the five given folds. RankNet was trained using TensorFlow 1 and v2 (Abadi et al., 2015), contrary to the other implementations. This makes the comparison of the run times difficult, however, we also re-implemented all RankLib models using our framework. Here it can be seen that overall the runtime improves for all models. By doing this one can also split the time for reading in the data and the actual training time of the model. The runtime for the RankNet and RankNet are of the same order.

Table 3 Comparing the average run times over the five folds of the MSLR–WEB10K data set in seconds

Time / Model	RankNet*	RankNet	ListNet	AdaRank	LambdaMart
Read data† [s]	67(6)	67(6)	67(6)	67(6)	67(6)
Train model† [s]	103(1)	100(1)	53(0)	196(14)	1046(23)
Train model‡ [s]	–	2215(351)	3947(481)	1261(50)	2664(234)

The row with ‡ were trained using RankLib, while the values with † where trained using our Python framework

Appendix C Hyperparameters

See Table 4.

Table 4 Overview of the hyperparameters which are used for training the different models on the different datasets

Algorithms	Hyperparameters	Values
RankNet* TF 1 / TF2	activation m	tanh, sigmoid, linear
RankNet Python	#layers	2...5
ListNet* Python	#neurons per layer	5...200
ListNet Python	learning rate stepwise decay	1 to 0.001 / 0.0001
	training steps / epoch	10k...30k / 5, 10, 30
RankNet* TF 1 / TF2	activation o_1	tanh
	early-stopping / -	True, False / -
	L_2 weight regularization	0%, 0.1%, 0.01%
	dropout	0%, 20%, 50%
	optimizer	Adam
	cost-function	L_2
RankNet Python	activation o_1	linear
	optimizer	Gradient descent
	cost-function	logistic
ListNet* Python	activation o_1	sigmoid
	L_2 weight regularization	0%, 0.1%, 0.01%
	dropout	0%, 20%, 50%
	optimizer	Adam
	cost-function	listwise loss
ListNet Python	activation o_1	linsigmoidear
	optimizer	Gradient descent
	cost-function	listwise loss
LambdaMart Python	#trees	5,10
	max-depth	4
	learning rate	0.1, 0.01
AdaRank Python	# rounds	100
	early stopping	False, 10, 20
RankNet RankLib	epoch	50, 100, 200, 300
	#layers	1, 5, 10
	#neurons per layer	10, 20, 100
ListNet RankLib	epoch	1000, 1500, 2000
	#layers	1
	learning rate	0.0001, 0.00001
LambdaMart RankLib	#trees	1000,2000
	#leafs	5,10,15
	learning rate	0.1, 0.01
AdaRank RankLib	# rounds	500, 1000, 2000
	tolerance	0.01, 0.001, 0.002, 0.005
	max consecutively feature	2, 5, 10

We started with an implementation of our method in Tensorflow v1 (TF 1). After an update of the framework to Tensorflow v2 (TF 2) we changed the number of training steps to an epoch based training approach. However, in the table we report both sets of hyperparameters. We also group all similar models together to show there overlapping properties

Acknowledgements 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. This work is dedicated to the memory of Martin Wagener, whose brilliant mind and kind spirit was instrumental in writing this paper. His sudden passing left an unfillable hole, yet his contributions and friendship will always be remembered.

Author contributions All authors discussed the results and contributed to the final manuscript. Marius Köppel, Alexander Segner and Martin Wagener conceived the presented idea of RankNet*. Alexander Segner developed the theory which was verified by Marius Köppel and Martin Wagener. The experiments done on the LETOR dataset were designed by Marius Köppel, Alexander Segner and Martin Wagener and mainly carried out by Marius Köppel. The experimental design on the training size and label-ratio dependency was done by Alexander Segner. The experiments comparing these properties with ListNet, RankNet and LambdaRank were performed by Marius Köppel. Lukas Pensel helped by designing the gridsearch evaluation method and contributed to the experimental computations. The idea of comparing neighbors in the sorted list of RankNet was developed by Marius Köppel, Alexander Segner and Martin Wagener while the experiments were done by Marius Köppel. During the whole work Marius Köppel, Alexander Segner and Martin Wagener were supervised by Andreas Karwath and Stefan Kramer. While Stefan Kramer helped to develop and further strengthening the main idea of improving RankNet and Andreas Karwath helped Marius Köppel, Alexander Segner and Martin Wagener by designing the experimental setup.

Funding Open Access funding enabled and organized by Projekt DEAL. 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.

Data and materials availability Not applicable

Code availability <https://github.com/kramerlab/direct-ranker>

Declarations

Conflict of interest Dr. Christian Schmitt, Mattia Cerrato and Luiz Frederic Wagner (uni-mainz.de), Johannes Fürtkranz and Iryna Gurevych (tu-darmstadt.de), birmingham.ac.uk, ethz.ch

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- 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. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. <http://tensorflow.org/>.
- Bahri, M., Bifet, A., Gama, J., Gomes, H. M., & Maniu, S. (2021). Data stream analysis: Foundations, major tasks and tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(3), e1405.
- Burges, C., Ragno, R., & Le, Q. (2006). Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems*, 19.

- Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96.
- Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., & Li, H. (2007). Learning to rank: from pairwise approach to list-wise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136.
- Cao, Y., Xu, J., Liu, T. Y., Li, H., Huang, Y., & Hon, H. W. (2006). 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.
- Cohen, J. (2007). *Review of "Introduction to Lattices and Order by BA Davey and HA Priestley"*, Cambridge University Press. *ACM SIGACT News*, 38(1), 17–23.
- Cooper, W. S., Gey, F. C., & Dabney, D. P. (1992). 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.
- W. B. Croft, J.C. (2001). *Lemur toolkit* (2001).
- Davey, B. A. (2002). *Introduction to lattices and order*. Cambridge University Press.
- Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4, 933–969.
- Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200), 675–701.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.
- Fuhr, N. (1989). Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7(3), 183–204.
- Hazimeh, H., Ponomareva, N., Mol, P., Tan, Z., & Mazumder, R. (2020). The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pp. 4138–4148. PMLR.
- Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. advances in large margin classifiers.
- Hinton, G. E. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint [arXiv:1207.0580](https://arxiv.org/abs/1207.0580).
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 359–366. [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8)
- Ibrahim, O. A. S., & Landa-Silva, D. (2017). Es-rank: evolution strategy learning to rank approach. In: *Proceedings of the Symposium on Applied Computing*, pp. 944–950. <https://doi.org/10.1145/3019612.3019696>. ACM.
- Ibrahim, O. A. S., & Landa-Silva, D. (2018). An evolutionary strategy with machine learning for learning to rank in information retrieval. *Soft Computing*, 22(10), 3171–3185. <https://doi.org/10.1007/s00500-017-2988-6>
- Jiang, L., Li, C., & Cai, Z. (2009). Learning decision tree for ranking. *Knowledge and Information Systems*, 20(1), 123–135.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- Li, P., Wu, Q., & Burges, C. J. (2008). Mcrank: Learning to rank using multiple classification and gradient boosting. *Advances in Neural Information Processing Systems*, pp. 897–904.
- Liu, T.-Y. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3), 225–331. <https://doi.org/10.1561/15000000016>
- Matsakis, N. D., & Klock, F. S. (2014). The rust language. *ACM SIGAda Ada Letters*, 34(3), 103–104.
- Nadeau, C., & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, 52, 239–281. <https://doi.org/10.1023/A:1024068626366>
- 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. (2011). Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Peter, N. (1963). *Distribution-Free Multiple Comparisons*. PhD thesis, Dissertation Princeton University
- Qin, T., & Liu, T. Y. (2013). Introducing LETOR 4.0 datasets. arXiv preprint [arXiv:1306.2597](https://arxiv.org/abs/1306.2597).
- Reimers, N. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint [arXiv:1908.10084](https://arxiv.org/abs/1908.10084).
- Rigutini, L., Papini, T., Maggini, M., & Bianchini, M. (2008). A neural network approach for learning object ranking. In *International Conference on Artificial Neural Networks*, pp. 899–908. Springer. https://doi.org/10.1007/978-3-540-87559-8_93.

- Rigutini, L., Papini, T., Maggini, M., & Scarselli, F. (2023). Sortnet: Learning to rank by a neural-based sorting algorithm. arXiv preprint [arXiv:2311.01864](https://arxiv.org/abs/2311.01864).
- Siekiera, J., Köppel, M., Simpson, E., Stowe, K., & Kramer, S. (2022). Ranking creative language characteristics in small data scenarios. In *International Conference on Computational Creativity, Bolzano-Bozen*. Association for Computational Creativity (ACC).
- Tesauro, G. (1988). Connectionist learning of expert preferences by comparison training. *Advances in Neural Information Processing Systems, 1*.
- Vendrov, I., Kiros, R., Fidler, S., & Urtasun, R. (2015). Order-embeddings of images and language. arXiv preprint [arXiv:1511.06361](https://arxiv.org/abs/1511.06361).
- Wu, Q., Burges, C. J., Svore, K. M., & Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval, 13*, 254–270.
- Xu, J., & Li, H. (2007). 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.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.