Received: 3 May 2021 | Revised: 20 November 2021 | Accepted: 11 February 2022

DOI: 10.1002/asi.24629

RESEARCH ARTICLE

JASIST WILEY

Learning to rank from relevance judgments distributions

Alberto Purpura | **Gianmaria Silvello** | **Gian Antonio Susto**

Department of Information Engineering, University of Padua, Padua

Correspondence
Alberto Purpura, Department of Information Engineering, University of Padua, 35122 Padua, Italy.
Email: purpuraa@dei.unipd.it

Abstract

LEarning TO Rank (LETOR) algorithms are usually trained on annotated corpora where a single relevance label is assigned to each available document-topic pair. Within the Cranfield framework, relevance labels result from merging either multiple expertly curated or crowdsourced human assessments. In this paper, we explore how to train LETOR models with relevance judgments distributions (either real or synthetically generated) assigned to document-topic pairs instead of single-valued relevance labels. We propose five new probabilistic loss functions to deal with the higher expressive power provided by relevance judgments distributions and show how they can be applied both to neural and gradient boosting machine (GBM) architectures. Moreover, we show how training a LETOR model on a sampled version of the relevance judgments from certain probability distributions can improve its performance when relying either on traditional or probabilistic loss functions. Finally, we validate our hypothesis on real-world crowdsourced relevance judgments distributions. Overall, we observe that relying on relevance judgments distributions to train different LETOR models can boost their performance and even outperform strong baselines such as LambdaMART on several test collections.

1 | INTRODUCTION

1.1 | Motivation

Ranking is a problem that we encounter in a number of tasks we perform every day: from searching on the Web to online shopping. Given an unordered set of items, this problem consists of ordering the items according to a certain notion of relevance. Generally, in information retrieval (IR) we rely on a notion of relevance that depends on the information need of a user, expressed through a keyword query. When creating a new experimental collection, the corresponding relevance judgments are obtained by asking different judges to assign a relevance score to each document-topic pair. Multiple judges—either trained experts or participants of a crowdsourcing experiment—usually assess the same document-topic pair, and the final relevance label for the pair is obtained by aggregating these scores (Hosseini et al., 2012). This process is a cornerstone for system training and evaluation and has contributed to the continuous development of IR, especially in the context of international evaluation campaigns. Nonetheless, the opinion of different judges on the same document-topic pair might be very different or even diverge to the opposite ends of the spectrum—either because of random human errors or due to a different interpretation of a topic. Inevitably, the aggregation process conflates the multiple assessors viewpoints on document-topic pairs onto a single one, thus losing some information—even though it also reduces annotation errors and outliers. Our research hypothesis is that machine learning (ML) models—that is, LETOR (Tax et al., 2015) and neural information retrieval (NIR) (Onal et al., 2018) models—could use all the labels collected in the annotation process to improve the quality of their rankings. Indeed, judges disagreement on a certain document-topic pair can be due to an inherent difficulty of the topic or to the existence of multiple interpretations of it. We argue that designing ML models able to learn from the whole distributions of relevance judgments could improve the models' representation of relevance and their performance through the usage of this additional information.

1.2 | Methods

Following this idea, we propose to interpret the output of a LETOR model as a probability value or distribution—according to the experimental hypotheses—and define different Kullback–Leibler (KL) divergence-based loss functions to train a model using a distribution of relevance judgments associated to the current training item. Such a training strategy allows us to leverage all the available information from human judges without additional computational costs compared to traditional LETOR training paradigms.

The loss functions we propose can be used to train any ranking model that relies on gradient-based learning, including popular NIR models or LETOR ones.

In this work we focus on transformer-based neural LETOR models and on one decision tree-based GBM model—the model at the base of the popular LambdaMART (Burges, 2010) ranker and used as a strong baseline in many recent LETOR research papers such as Bruch (2019), Bruch et al. (2020), Bruch et al. (2019), Pasumarthi et al. (2020), and Zhen et al. (2021).

1.3 | Evaluation

We assess the quality of the proposed training strategies on four standard LETOR collections (MQ2007, MQ2008, MSLR-WEB30K (Qin & Liu, 2013) and OHSUMED (Qin, Liu, Xu, & Li, 2010)) and three different transformer-based LETOR models. We also conduct a crowdsourcing experiment to build a new LETOR collection based on the COVID-19 MultiLingual Information Access (MLIA) data.[1] We then use the raw relevance labels and their aggregated version to assess the impact on a LETOR model trained on raw relevance labels versus their aggregated form.

1.4 | Contributions

(a) The definition of five new loss functions to train different LETOR models using probability distributions of relevance labels; (b) an extensive evaluation of the loss functions on neural and decision tree-based LETOR models using standard test collections and a newly created crowdsourced LETOR test collection based on COVID-19 MLIA data.

1.5 | Outline

In section 2, we describe the most relevant training strategies for LETOR and NIR models; in section 3, we present the probabilistic loss functions leveraging on relevance judgments distributions and the novel neural LETOR model we employ for evaluation; in section 4, we describe the experimental setup and in section 5 we discuss the evaluation results; in section 6, we draw some conclusions and discuss future work.

2 | RELATED WORK

Decision tree-based approaches such as LambdaMART (Burges, 2010) have been for many years the most popular ML models in this domain, but recently—thanks to the growing size of LETOR collections, new optimization functions (Bruch et al., 2020) and feature normalization strategies (Zhuang, Wang, Bendersky, & Najork, 2020)—deep learning approaches, transformer-based ones in particular (Zhen et al., 2021), are showing an increasingly competitive performance.

Transformer-based LETOR models rely on one or more self-attention layers. This type of neural layer architecture was originally proposed in Vaswani et al. (2017), and later popularized by language models such as bidirectional encoder representations from transformers (BERT) (Devlin et al., 2018). This architecture allows LETOR models to efficiently evaluate and compare lists of candidate relevant documents to a user query, providing a numerical estimate of their relevance. One of the latest and most successful approaches of this kind is data augmented self-attentive latent cross model (DASALC) (Zhen et al., 2021). It relies on a few strategies such as neural feature transformation, self-attention layers, a listwise ranking loss, and model ensembling, to outperform strong non-neural baselines such as LambdaMART on public LETOR collections. Strategies such as neural feature transformation are frequently employed in the context of neural LETOR models to normalize the representation of their inputs so that they could be better interpreted by such models (Zhuang, Wang, Bendersky, & Najork, 2020). Self-attention layers—such as the one we employ in the proposed neural LETOR model described in section 3—allow to efficiently compare groups of items at the feature level. Finally, listwise ranking losses and model ensembling strategies are already popular solutions in LETOR and machine learning to improve the performance of ranking models (Bruch et al., 2019; Burges, 2010).

In this work, we will show how we can achieve similar improvements in a more efficient way through the usage of a new class of *probabilistic* loss functions that we propose.

Indeed, in addition to the development of new architectures, another important branch of research in the LETOR domain focuses on the study of new loss functions specific to ranking problems (Tax et al., 2015). These loss functions are generally categorized as *pointwise*, *pairwise*, and *listwise* (Chen et al., 2009). Pointwise loss functions are used to train a model to fit the corresponding relevance score for each document-topic pair as in a regression task. Loss functions belonging to this class can be described using the following general formulation: Pointwise$(q,d,y) = f(s(q,d),y)$, where q indicates a query, d a document, y its relevance label, $s(\cdot,\cdot)$ the function learned by a ML model to compute the relevance of a document given a query, and $f(\cdot)$ the generic function which compares the score computed by the model that is being trained with the corresponding relevance label. One of the possible implementations of $f(\cdot)$ is the mean squared error (MSE) (Liu et al., 2018). Pairwise loss functions consider pairs of documents and compare their relevance scores using different strategies. This class of losses can be formalized as: Pairwise$(q,d_1,d_2) = f(s(q,d_1),s(q,d_2))$, where the function f can have different formulations such as the Hinge function $\phi(z_1,z_2) = \max(0,1-z_1+z_2)$, where $z_i = s(q,d_i) \forall i \in \{1,...,n\}$ (Guo et al., 2016). Finally, listwise loss functions take into account a set of documents relative to a certain query and compute the loss for the group of items as: Listwise$(q,\{d_1,...,d_n\},\{y_1,...,y_n\}) = f((s(q,d_1),...,s(q,d_n)),\{y_1,...,y_n\})$, where $\{y_1,...,y_n\}$ are the relevance judgments associated to $\{d_1,...,d_n\}$ and the function $f(\cdot)$ can take different formulations such as the ApproxNDCG (Qin, Liu, & Li, 2010) loss.

Among the numerous loss function formulations proposed by the LETOR community, the most widely employed in the latest state-of-the-art text-based NIR (MacAvaney et al., 2019) and LETOR models (Zhuang, Wang, Bendersky, & Najork, 2020) are the pairwise Hinge (Onal et al., 2018) and the listwise ApproxNDCG loss (Bruch et al., 2019; Qin, Liu, & Li, 2010), respectively. The formulation of the Hinge loss as used in the NIR domain is Hinge$(q,d^+,d^-) = \max(0,1-s(q,d^+)+s(q,d^-))$, where d^+ and d^- identify, respectively, a relevant and a not-relevant document for the query q. The goal of this loss function is to maximize the difference between the relevance probabilities—indicated by the function $s(q,\cdot)$—computed for each document. The Hinge loss is frequently chosen to train text-based NIR models such as DRMM (Guo et al., 2016), MatchPyramid (Pang et al., 2016), or the most recently proposed CEDR (MacAvaney et al., 2019), but was also used as a baseline in LETOR research works such as in Zamani and Croft (2018), where it demonstrated to be a competitive candidate among other pointwise or pairwise loss functions. The main advantage of this loss function is its ability to perform well on relatively small datasets—as it is often the case for text-based NIR models when evaluated on shared IR test collections. Differently from the Hinge loss, the ApproxNDCG loss can take into account more than two documents at a time and, as the name suggests, provides a differentiable approximation of the normalized Discounted Cumulated Gain (nDCG) measure for the evaluation of a ranked list. Given a permutation π of items $\{x_1,...,x_n\}$ and the corresponding sequence of relevance labels $\mathbf{y}$, the nDCG is defined as

$$\text{nDCG}(\pi,\mathbf{y}) = \frac{\text{DCG}(\pi,\mathbf{y})}{\text{DCG}(\pi*,\mathbf{y})}, \tag{1}$$

where π^* is the ideal ranking of the items according to the relevance labels $\mathbf{y}$. The discounted cumulated gain (DCG) is computed as (Chen et al., 2009)

$$\text{DCG}(\pi,\mathbf{y}) = \sum_{i=1}^{n} \frac{2^{y_i}-1}{\log_2(1+r(x_i,\pi))}, \tag{2}$$

where $r(x_i,\pi)$ is the function returning the rank of item x_i in π and y_i is its label. This measure, however, is not differentiable because the function $r(\cdot,\cdot)$ is not. To tackle this problem, Qin, Liu, and Li (2010) propose a differentiable approximation of nDCG which can be used as a loss function to train ML models (Bruch et al., 2019). In the ApproxNDCG loss, the non-differentiable $r(\cdot,\cdot)$ function is replaced by its approximation,

$$\hat{r}(x_i,\pi) = 1 + \sum_{j \neq i} \mathbf{I}_{s(x_i) < s(x_j)}, \forall x_j \in \pi, \tag{3}$$

where $s(\cdot)$ indicates the scoring function of the model we are training and $\mathbf{I}_{u<v}$ is the indicator function which is 1 if $u < v$ and 0 otherwise. In turn, the indicator function is approximated with: $\sigma(v-u) = (1+e^{-\alpha(v-u)})^{-1}$, where α is a parameter to control the steepness of the sigmoid function $\sigma(\cdot)$. More recently, a stochastically treated version of the ApproxNDCG function was proposed by Bruch et al. (2020). In the paper, zero-mean logistic noise is added to the sigmoid function presented earlier, which becomes $\sigma((v-u)+Z_{vu})$, where $Z_{vu} \sim \text{Logistic}(\text{mean}=0,\text{scale}=\beta)$.

Intuitively, adding random noise to the model outputs induces it to increase the difference between the relevance scores of relevant and not-relevant items to maintain their relative ordering in the final ranked list. ApproxNDCG and its variant with stochastic treatment (ST) are the most popular and best performing listwise loss functions among the recently published neural LETOR works such as Bruch et al. (2020), Bruch et al. (2019), Zhuang, Wang, Bendersky, Grushetsky, et al. (2020), and Zhuang, Wang, Bendersky, and Najork (2020).

Regarding the collection of relevance judgments, there is still an open debate in the IR community on how to reliably collect them and on the best strategies for their aggregation. Crowdsourcing is a valuable option in this context (Alonso, 2019). The TREC crowdsourcing track (Lease & Kazai, 2011; Smucker et al., 2012; Smucker et al., 2013), for example, explored the challenges related to the collection and management of relevance judgments for Web pages and search topics. Numerous aggregation options are also described in the IR literature on crowdsourcing (Ferrante, Ferro, & Piazzon, 2020; Hosseini et al., 2012; Ravana & Moffat, 2009) involving strategies to weigh the annotations of each judge depending on the topic difficulty and/or his/her level of confidence on it. Here, we take a new approach to this problem, eliminating the relevance judgments aggregation step and training a model on the distribution of relevance judgments associated to each document-topic pair.

We propose five different loss functions that allow ranking models to take advantage of relevance judgments distributions prior to their aggregation. Since, to the best of our knowledge, no similar method was previously presented in the literature, we compare the newly proposed training strategies to five high-performing loss functions representative for each class of functions described before, that is, MSE, Hinge, ApproxNDCG, ApproxNDCG with ST and ListMLE.

3 | PROPOSED APPROACH

In this section we introduce the proposed pointwise, pairwise, and listwise loss functions and the neural LETOR model that we use as reference architecture.

3.1 | Pointwise loss functions

These formulations rely on KL divergence to compare two probability distributions, that is, the relevance score computed by our LETOR model for a document-topic pair and its corresponding true relevance label. We interpret the relevance label assigned to a document as if it was generated by a Binomial random variable modeling the judges' annotation process. For example, we assume that n assessors provided one binary relevance label for each document-topic pair, that is, to state whether the pair was a relevant or a not-relevant one. This process can be modeled as a Binomial random variable $P \sim \mathrm{Bin}(n,p)$ where the success probability p for each sample is the average of the binary responses submitted in n trials. Since in most LETOR datasets, relevance judgments are not real values in the [0,1] range, we normalize them to fit this interval.

We then apply the same reasoning for the interpretation of our model output probability score as another Binomial distribution $\widehat{P} \sim \mathrm{Bin}(n,\widehat{p})$ with the same parameter n—empirically tuned for the numerical stability of the gradients during training—and probability $\widehat{p}$ equal to the output of the model which is kept in the $[0,1] \in \mathbb{R}$ range employing a sigmoid activation function at the output layer.

At this point, the model output and relevance labels distributions are expressed as Binomial distributions with the same parameter n and different success probabilities; hence, we can compare them using the KL divergence,

$$D_{\mathrm{KL_{Bin}}}\left(P \middle\| \widehat{P}\right) = \log\left(\frac{p}{\widehat{p}}\right)np + \log\left(\frac{1-p}{1-\widehat{p}}\right)n(1-p), \quad (4)$$

that is differentiable because it is the sum of two differentiable and continuous functions for p and $\widehat{p}$ in the open (0,1) interval. Since KL divergence is not symmetric—which would lead to issues in the comparison of different items during model optimization—we employ the following symmetric and non-negative formulation as the loss function for each data point in a training batch:

$$\mathrm{Pointwise}_{\mathrm{KL(Bin)}} = \left(D_{\mathrm{KL}}\left(P_i \middle\| \widehat{P}_i\right) + D_{\mathrm{KL}}\left(\widehat{P}_i \middle\| P_i\right)\right) * w_i. \quad (5)$$

Since we train our model feeding it all the items corresponding to one or more ranked lists provided in the training data, we need to balance the contributions of relevant (the minority) and not-relevant (the majority) items to the final loss function value. For this reason, before computing the total loss value in a batch by summing the contribution of each data point, we rescale each term by a factor w_i, inversely proportional to the number of times an item of the same class appeared in the batch.

Since our relevance labels are graded and normalized between $[0,1] \in \mathbb{R}$, we consider as not-relevant the data points associated to a relevance label lower than an empirically set threshold, and the remaining ones as relevant. In our experiments, we set this threshold to 0.1—

instead of for example 0.0—for it to be used also when simulating the relevance labels distributions as described in section 5, sampling them from continuous probability distributions.

Whenever an actual relevance judgments distribution is available, we propose to use another pointwise loss function which takes into account the distribution of values over a number of relevance grades, interpreting them as outcomes from a multinomial distribution (Agrawal, 2020; Bishop, 2006) $P \sim \mathrm{Mul}(n, (p_1, ..., p_k))$—which is a generalization of the aforementioned Binomial distribution with the same parameter n. The outcomes of the modeled random process take a finite number of values, where $\sum_{i=1}^{k} p_i = 1$. Each p_i indicates the probability of one of the k possible relevance grades to be selected by the pool of judges that were employed to asses a certain document-topic pair. To obtain a comparable distribution to the output of our model we adjust the output layer size to k and employ a softmax activation function over each output sequence obtaining the probabilities $(\widehat{p}_1, ..., \widehat{p}_k)$, which allow us to define the random variable $\widehat{P} \sim \mathrm{Mul}(n, (\widehat{p}_1, ..., \widehat{p}_k))$. We then compare the two distributions with the same strategy used before, changing the formulation of the KL divergence to Multinomial distributions,

$$D_{\mathrm{KL(Mul)}}\left(P \middle\| \widehat{P}\right) = \sum_{i=j}^{k} p_j \log \frac{p_j}{\widehat{p}_j}. \tag{6}$$

This function is continuous and differentiable for non-zero probability values but, again, not symmetric. Therefore, we train our model using the following symmetric and non-negative formulation to evaluate the quality of approximation of the relevance of each item:

$$\mathrm{Pointwise}_{\mathrm{KL(Mul)}} = \left(D_{\mathrm{KL(Mul)}}\left(P_i \middle\| \widehat{P}_i\right) + D_{\mathrm{KL(Mul)}}\left(\widehat{P}_i \middle\| P_i\right)\right) * w_i. \tag{7}$$

As in the previous case, before computing the total loss in a batch, we rescale the contribution of each data point in it by a factor w_i.

In this case, we recommend the collection of a number of relevance labels for each query-document pair sufficient to estimate the distribution of opinions on its relevance for the user base of the search system. For example, if we are evaluating the relevance of a recently published academic study with respect to a certain topic, if our audience is made of experts from the same academic field, then a wider range of relevance labels should be collected to capture all of the nuances of the opinions field professionals could have on the topic. On the other hand, if our audience is the general public, the number of relevance judgments required can be smaller and proportional to the public agreement on the specific topic.

Note also that, despite the interpretation we provide for the parameter n in the Binomial and Multinomial distributions as the number of judges providing relevance labels for the same query-document pair, in the remainder of this paper we consider n as a hyperparameter of the model. Indeed, even if there could be a relation between the number of judges providing relevance labels and the corresponding random variable which could model this process, this intuition would be hard to verify empirically. Indeed, the number of judges available for the creation of a new IR collection is often limited by real-world constraints—such as labor cost or the availability of an appropriate number of trained professionals—therefore, our hypothesis becomes hard to verify for a large number of topics where potentially dozens or even hundreds of judges could be required for the annotation of each query-document pair in an exhaustive study.

3.2 | Pairwise loss functions

We also propose two pairwise loss functions. The intuition here is to increase the model robustness and generalization power through a comparison between distributions instead of real-valued relevance scores. To achieve this goal, given a topic, we compare all pairs of relevant and not-relevant—or less relevant if relevance labels are not binary—documents in a batch by considering the output relevance scores produced by our model and the respective relevance labels. In other words, for each topic in an experimental collection we consider every pair of documents available with a different relevance label—obtained by either aggregating relevance scores if relevance judgments distributions are available, or taking their exact value—and compare them to train a LETOR model with one of the loss functions described below. We evaluate two possible interpretations for these relevance scores.

The first option, similarly to what we did for the $\mathrm{Pointwise}_{\mathrm{KL(Bin)}}$ loss function, is to consider the relevance scores as the success probabilities of Binomial random variables and then to compute their KL divergence.

The second option is to consider the relevance scores of each pair as samples from two different Gaussian random variables $P^+ \sim \mathcal{N}\left(\mu_{p^+}, \sigma\right)$ and $P^- \sim \mathcal{N}\left(\mu_{p^-}, \sigma\right)$ with the same standard deviation σ but centered on the relevance labels—or model output scores—μ_{p+} and μ_{p^-}. Our hypothesis is that relevance judgments provided by different judges for a certain document-topic pair (relevant or not-relevant) will have a certain standard deviation (σ)

and will be all centered around a certain value μ, following a Gaussian random process. Therefore, if we assume the standard deviation—that is, the disagreement of different judges over each annotated sample—to be constant over time, we can model the process with a Gaussian random variable with μ equal to the output relevance score of our model—interpreted as a sample from the true distribution—and standard deviation σ to be adjusted according to the level of agreement/disagreement that we hypothesize in the annotation process.

Depending on the modeling strategy, the proposed loss functions take the following formulations typical of pairwise hinge losses (Chen et al., 2009), where we replace the term dedicated to compare a pair of item with their sign-corrected KL divergence value,

$$\text{Pairwise}_{\text{KL(Bin)}} = \max\left(0, m - \text{sign}(p^+ - p^-) D_{\text{KL(Bin)}}\left(P^+_{\text{Bin}}, P^-_{\text{Bin}}\right)\right), \tag{8}$$

where m is a slack parameter to adjust the distance between the two distributions, p^+ and p^- are the outputs of the LETOR model associated to two documents—the former with a higher relevance label than the latter—$P^+_{\text{Bin}} \sim \text{Bin}(n, p^+)$ and $P^-_{\text{Bin}} \sim \text{Bin}(n, p^-)$ are two Binomial distributions corresponding to a relevant and to a not-relevant document-topic pair, respectively. The respective success probabilities are equal to the sigmoid-bounded relevance probability scores returned by the LETOR model to train, as done for the corresponding pointwise loss function, see Equation (5). In this case, if the relevant data point has a relevance probability $p^+ > p^-$, the loss is equal to the difference between m and the KL divergence between the distributions. This difference is lower bounded by zero, thanks to the max operation implemented with the rectified linear unit (ReLU) function. In this situation, the slack variable m can be tuned to increase or decrease the distance between the two distributions. Conversely, if $p^- > p^+$, then the loss is positive and equal to the sum of m and the value of the KL divergence between the two distributions.

The formulation of the pairwise loss function relying on Gaussian distributions has a similar form, but it uses the formulation of the KL divergence between two Gaussian random variables $P^+ \sim \mathcal{N}\left(\mu_{p^+}, \sigma_{p^+}\right)$ and $P^- \sim \mathcal{N}\left(\mu_{p^-}, \sigma_{p^-}\right)$,

$$D_{\text{KL}(\mathcal{N})}\left(P \middle\| P^-\right) = \frac{1}{2}\left[2\log\frac{\sigma_{p^-}}{\sigma_{p^+}} - 1 + \frac{\sigma^2_{p^+} + \left(\mu_{p^+} - \mu_{p^-}\right)^2}{\sigma^2_{p^-}}\right]. \tag{9}$$

This function is differentiable and, if the two random variables have the same variance, also symmetric and not-negative—that is, $D_{\text{KL}(\mathcal{N})}\left(P \middle\| P^-\right) = \frac{\sigma^2 + \left(\mu_{p^+} - \mu_{p^-}\right)^2}{2\sigma^2} - \frac{1}{2}$. Therefore, we can employ it to train the model as

$$\text{Pairwise}_{\text{KL}(\mathcal{N})} = \max\left(0, m - \text{sign}(p^+ - p^-) D_{\text{KL}(\mathcal{N})}\left(P^+_{\mathcal{N}}, P^-_{\mathcal{N}}\right)\right). \tag{10}$$

3.3 | Listwise loss function

Finally, we propose a listwise loss function also based on the KL divergence between distributions. In this case, we consider the whole set of relevance probabilities associated to k documents in a ranked lists of a batch and their respective relevance labels as two multivariate Gaussian distributions. In a similar way as in Equation (9), we compute the KL divergence between two multivariate Gaussian distributions $\hat{P} \sim \mathcal{N}\left(\mu_{\hat{p}}, \Sigma_{\hat{p}}\right)$ and $P \sim \mathcal{N}\left(\mu_p, \Sigma_p\right)$, obtaining

$$D_{\text{KL}(\mathcal{N}_{\text{mult}})} = \frac{1}{2}\left[\text{tr}\left(\Sigma_p{}^{-1}\Sigma_{\hat{p}}\right) + \left(\mu_p - \mu_{\hat{p}}\right)^T \Sigma_p^{-1}\left(\mu_p - \mu_{\hat{p}}\right) - k + \log\left(\frac{\det\Sigma_p}{\det\Sigma_{\hat{p}}}\right)\right]. \tag{11}$$

If the two distributions have the same diagonal covariance matrix, Equation (11) reduces to

$$D_{\text{KL}(\mathcal{N}_{\text{mult}})} = \frac{1}{2}\left(\mu_p - \mu_{\hat{p}}\right)^T \Sigma_p^{-1}\left[\left(\mu_p - \mu_{\hat{p}}\right)\right], \tag{12}$$

which is also symmetric and not negative. As for the $\text{Pointwise}_{\text{KL(Bin)}}$ and $\text{Pointwise}_{\text{KL(Mul)}}$, we rescale the components of the distributions associated to relevant and not-relevant items by a factor $\boldsymbol{w}$ inversely proportional to their respective class frequency in the current ranked list. Therefore, the loss function value associated to one ranked list in a batch is computed as

$$\text{Listwise}_{\text{KL}(\mathcal{N}_{\text{mult}})} = D_{\text{KL}(\mathcal{N}_{\text{mult}})}\left(\hat{P} \middle\| P\right) * w, \tag{13}$$

where $\hat{P} \sim \mathcal{N}\left(\mu_{\hat{p}}, \Sigma\right)$ represents the output distribution of relevance probabilities returned by our model and $P \sim \mathcal{N}\left(\mu_p, \Sigma\right)$ indicates the distribution of true relevance judgments associated to each item in a ranked list. The loss function values associated to each ranked list are then averaged over a batch.

3.4 | Neural LETOR model

The architecture of the transformer-based neural model that we employ in our experiments is depicted in Figure 1 and is inspired to other popular ones such as Pobrotyn et al. (2020), Sun and Duh (2020), and Zhen et al. (2021). The first layer of the proposed architecture—depicted in Figure 2—is a standard multi-head self-attention (SA) layer as the one introduced in Vaswani et al. (2017) to compare the different document representations in input. The outputs of each SA head—of size $m = f/k$, that is, equal to the original document representation size f divided by the number k of attention heads used—are then concatenated and fed to a regularization layer with the goal of eliminating the redundancy in the representations of different attention heads and extracting only the features and their combinations which are useful for our goal. The regularization layer that we employ takes as input the concatenation of the outputs of different attention heads associated to the documents and begins by normalizing their components to have mean 0 and standard deviation equal to 1. Then, we feed these normalized representations into a feed-forward neural network (FFNN) of size t times the input vector size f, using the ReLU activation function. Next, we normalize its output—with the goal of forcing to zero the components of the output which are redundant—and feed them to another FFNN of size f. The output of the latter layer is then summed to the normalized input to the layer as described in Figure 3. Finally, we normalize the components of the output vector of this layer to have 0 mean and unit standard deviation. Each normalization layer estimates the components mean and standard deviations considering each feature separately. This layer is used to improve the numerical stability of the operations in the model by maintaining the values within a constant range.

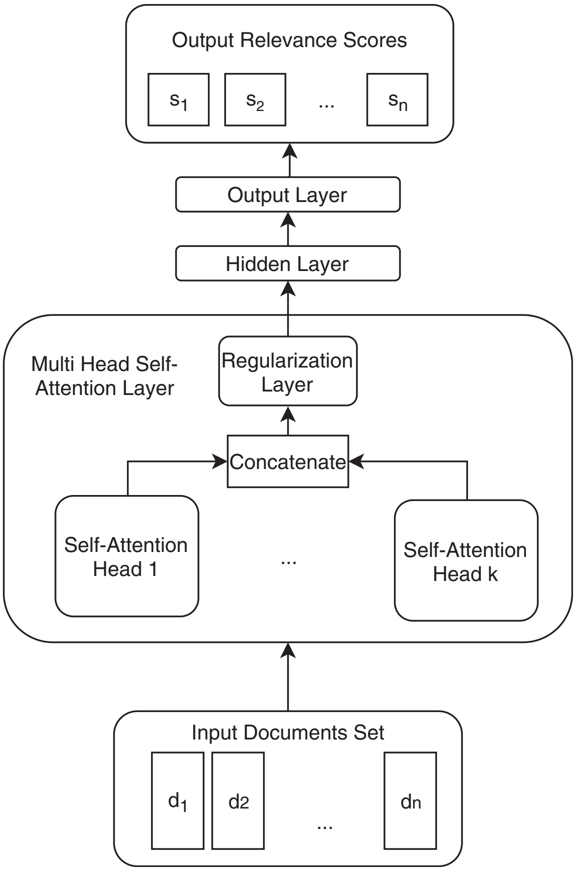

FIGURE 1 Schema of the neural model employed in our experiments

The representation of each document returned by our regularization layer is then fed to a hidden layer of size h with a ReLU activation function and to an output layer. The output layer size is equal to the number of relevance levels of the current collection if we are using the $\text{Pointwise}_{\text{KL(Mul)}}$ loss function, one otherwise. The activation function we use depends on the output size and is the *softmax* function if we are considering a multi-class output and the *sigmoid* in the other cases.

4 | EXPERIMENTAL SETUP

In this section, we describe the main technical details of our experimental setup, additional details are available in our code repository, along with the source code of the proposed approaches and the newly created test collection: https://github.com/albpurpura/PLTR.

4.1 | Experimental collections

The experimental collections that we consider in our experiments are MQ2007, MQ2008, MSLR-WEB30K (Qin & Liu, 2013) and OHSUMED (Qin, Liu, Xu, &

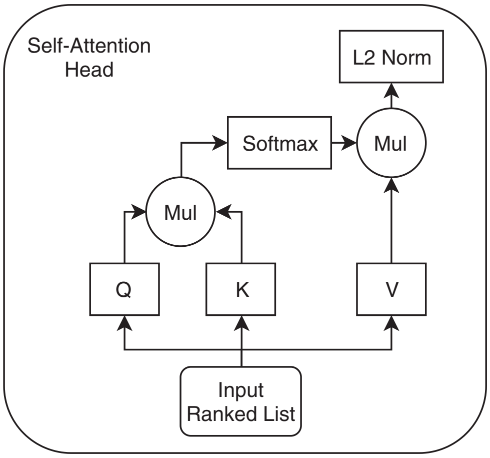

FIGURE 2 Schema of the self-attention layer we employ in the proposed transformer-based LETOR model

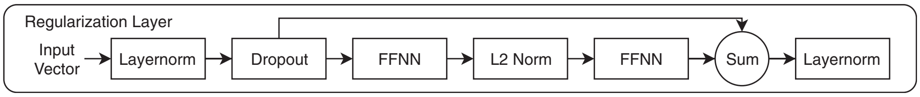

FIGURE 3 Schema of the proposed regularization layer that we use in our LETOR model

Li, 2010). All collections are already organized in five different folds with the respective training, test and validation subsets. We report the performance of our model averaged over these folds with the exception of the MSLR-WEB30K collection where we only consider Fold 1 as in other popular research works (Ibrahim & Carman, 2016; Zhen et al., 2021; Zhuang, Wang, Bendersky, Grushetsky, et al., 2020; Zhuang, Wang, Bendersky, & Najork, 2020).

The MQ2007 and MQ2008 collections contain 1,700 and 800 queries, with a respective average of 40 and 18 assessed documents per query. Their relevance scores are integer values ranging from 0 to 2, indicating an increasing degree of relevance. The MSLR-WEB30K collection is a subset of 30,000 queries from the retired training set of the commercial search engine Microsoft Bing with an average number of 125 judged documents per query. Since these features x_i are not normalized, we normalize them applying the following transformation proposed in Zhuang, Wang, Bendersky, and Najork (2020) and also used in Zhen et al. (2021): $\widehat{x_i} = \log(|1+x_i|) * \text{sign}(x_i)$. Relevance labels here are integers from 0 to 4. The OHSUMED (Qin, Liu, Xu, & Li, 2010) collection contains about 16 K documents from MEDLINE and 106 queries with an average of 125 assessed documents. Relevance labels here are in the {0,1,2} set.

4.2 | Crowdsourcing relevance judgments

We also conducted a crowdsourcing experiment to obtain relevance judgments distributions for a subset of document-topic pairs from the COVID-19 MLIA collection. We consider the English MLIA subset, which contains 30 topics related to the COVID-19 pandemic and documents scraped from different online sources.[2] We collected relevance judgments on an average of 64 documents per topic ($SD = 4.37$) with an average of 4 different judges ($SD = 7.12$) per document and with an average pairwise inter-annotator agreement of 70%. Judges are voluntary master or Ph.D. students in computer engineering or foreign languages. The relevance labels that we consider are in the {0,1,2} set, indicating increasing relevance. We built a LETOR collection computing the documents features by employing different retrieval models and configurations available in the Apache Lucene framework.[3] We considered 24 different retrieval pipelines using a combination of one component from each the following sets: {BM25, Language Model with Dirichlet Smoothing (LMD), Divergence From Randomness} retrieval model, {Lucene, Indri, Atire, Okapi} stoplist – available online[4] – and {Porter, Lovins} stemmer.

When required by the loss functions, we aggregated the relevance judgments distributions to obtain a real-valued relevance score. This was achieved by computing a weighted average with the following weights [−1.0, 0.5, 1.0], obtaining a score within −1 and 1 which was then rescaled and shifted between 0 and 1 to be used for training and evaluation purposes. We chose this aggregation strategy as an alternative to the majority vote method, to better preserve all the possible degrees of relevance for a document-topic pair.

4.3 | Model hyperparameters and training

The training parameters of the aforementioned neural model are the number of attention heads k, the factor t which is used to determine the size $f \times t$ of the first FFNN in the proposed regularization layer, and the hidden layer size h. These parameters were adjusted according to the experimental collection size and number of features of the input data points and tuned on the validation sets of the first folds of each collection. The model was trained for a maximum of 100 epochs with early stopping with patience of 20 epochs on the MQ2007, MQ2008 and MSLR-WEB30K collections, while we reduced the maximum number of training epochs to 50 on the OHSUMED and MLIA collection. The best model was then selected for each fold according to the nDCG@1 performance on the validation set. The number of attention heads k was dependent both on the collection size and the number of features available to represent each document. This parameter was set to 2 for the experiments on the MQ2007 and MQ2008 collections, 4 on the MSLR-WEB30K, 3 on the OHSUMED, and 1 on the MLIA collection. The factor t was empirically set to 3 for the experiments on all collections, while the hidden layer size h was set to 32 for the experiments on the MQ2007, MQ2008, and OHSUMED collections, to 128 for the ones on the MSLR-WEB30K and to 8 for the experiments on MLIA.

Since in the MSLR-WEB30K and OHSUMED collections the maximum number of documents per query is much larger than the average on the other datasets, we reranked in these cases only the top 150 documents ranked by a LightGBM LambdaMART model (Ke et al., 2017) tuned considering the nDCG@1 measure on the validation set of each collection fold. We cut the ranked lists at 150 documents since this was also the maximum value of documents available to rerank per-query on the MQ2007, MQ2008, and MLIA collections.

In our experimental section, we also provide a performance comparison of our probabilistic training approach simulating a noisy annotation process. For each annotated query-document pair, we sample a new set of relevance labels from a Binomial distribution with parameters $n = 32$—we determined this value empirically according of the performance of the model on the validation set of the first fold of the experimental collections in the previous experiments with pointwise loss functions—and p equal to the normalized relevance label found in the dataset. In other words, if a dataset employs relevance labels in the set {0, 1, 2}, then the p values we employ to sample new relevance labels will take values in the set {0, 0.5, 1}, in the same order. After the sampling step, we average the sampled values to generate a new "sampled" relevance label. This label is used as the parameter p in Equations (5), (7), and (13) and compared to the real-valued output of the model we train indicated as $\hat{p}$. The sampled relevance label is also used to compute which documents to consider in each training pair provided to the pairwise loss functions described in Equations (8) and (10). This process does not change the relevance label of highly relevant or not-relevant documents—that is, the documents with the highest or lowest relevance grade that have a success probability of 1.0 or 0.0, respectively—but provides some range of variability for the documents judged as *partially-relevant*, with a variance proportional to the uncertainty on the relevance of the document. Indeed the variance of each Binomial variable $P \sim \mathrm{Bin}(n,p)$ is defined as $np(1-p)$, it is therefore higher for values closer to 0.5—that is, values of p associated to relevance labels in the middle of the grading scale. Intuitively, the parameter n increases or decreases the width of the probability density function (PDF) of the distributions we compare, that is, a larger n can be used in the cases where there is a large variability in the relevance judgments distribution for the same query-document pair and vice versa. Therefore, we recommend treating n as a model parameter and tuning it on a separate validation set.

4.4 | Evaluation measures and baselines

We evaluate the performance of the proposed loss functions relying on top-heavy, widely used measures as ERR (Chapelle et al., 2009), nDCG@{1, 3, 5},[5] and P@{1, 3, 5}; these are also among the most used measures in the LETOR literature. Moreover, despite recent critics (Ferrante, Ferro, & Losiouk, 2020; Fuhr, 2018) and the open debate in the community, to ease the comparison with other LETOR approaches in the literature (Ibrahim & Carman, 2016; Köppel et al., 2019; Tax et al., 2015), we also report the average precision (AP) of each of our runs averaged over all topics (MAP). We also compute a paired Student's t test for each measure and report the performance difference between the baseline of choice (i.e., the proposed neural model trained with the ApproxNDCG loss function or the LambdaMART model, depending on the experiment at hand) and the same model trained with the other loss functions; we indicate with ↑ or ↓ a statistically significant difference ($\alpha < .05$), accordingly to the sign of the difference. We selected the ApproxNDCG loss as a reference loss function for the proposed neural model since it was the one with the best performance across all the considered collections. To simplify our experimental analysis, we report the performance of our model when trained using the following loss functions as representatives from the pointwise, pairwise and listwise loss categories: MSE, Hinge, ApproxNDCG with and without ST and ListMLE.

Given the success of the LambdaMART model in the LETOR domain and its widespread usage as a strong reference baseline for the evaluation of new systems (Zhen et al., 2021), we also evaluate the impact of the proposed loss functions to train a decision tree-based GBM model, that is the model at the base of LambdaMART (Burges, 2010). We also evaluate the impact of the proposed training strategies on two popular neural LETOR models, that is, DASALC (Zhen et al., 2021) and a simple transformer model with one self-attention layer similar to the one used in Pobrotyn et al. (2020).

To perform the experiments with the GBM model, we rely on the LightGBM library. On each collection, we tune the hyperparameters of the GBM to obtain the highest nDCG@1 on the validation set of Fold 1 of each experimental collection. The hyperparameter optimization process is performed—similarly to Qin and Liu (2013)—through a grid search over the following parameters: learning rate {0.001, 0.05, 0.1, 0.5}, number of trees {300, 500, 1000}, number of leaves {200, 500, 1000} and followed by an additional manual tuning around the best hyperparameters combination found in the previous step. The best model hyperparameters for each collection are reported in our online repository. We also compare the performance of the neural LETOR and GBM model to the LightGBM LambdaMART implementation and to other neural models. We tuned the hyperparameters of each neural model we consider following the same criteria described in Zhen et al. (2021).

5 | EVALUATION

In Table 1, we report the performance of the proposed LETOR neural model trained using different loss functions on the MQ2007, MQ2008, MSLR-WEB30K, and OHSUMED collections. We observe that the neural model achieves the best performance in the majority of the collections when trained with the Pairwise$_{\mathrm{KL}(\mathcal{N})}$ loss function. On the MQ2007 and MQ2008 collections, all the proposed pairwise and listwise functions outperform

TABLE 1 Performance of the proposed LETOR neural model averaged over all topics

	Loss function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	ApproxNDCG	0.3169	0.4639	0.4291	0.4105	0.4152	0.4150	0.4219	0.4603
	ListMLE	0.3178	0.4681	0.4336	0.4116	0.4178	0.4175	0.4215	0.4596
	ApproxNDCG (ST)	0.2704↓	0.3842↓	0.3621↓	0.3494↓	0.3363↓	0.3391↓	0.3459↓	0.4021↓
	Hinge	0.2674↓	0.3723↓	0.3633↓	0.3511↓	0.3239↓	0.3334↓	0.3420↓	0.4004↓
	MSE	0.3154	0.4574	0.4291	0.4115	0.4113	0.4136	0.4205	0.4550↓
	Pointwise KL (binomial)	0.3168	0.4551	0.4334	0.4132	0.4087	0.4177	0.4230	0.4601
	Pairwise KL (binomial)	0.3196	0.4728	0.4377	0.4186↑	0.4249	0.4226	0.4297↑	0.4647
	Pairwise KL (Gaussian)	**0.3218**	**0.4817**↑	**0.4381**	**0.4201**↑	**0.4350**↑	**0.4249**↑	**0.4318**↑	**0.4665**↑
	Listwise KL (Gaussian)	0.3177	0.4657	0.4332	0.4145	0.4173	0.4192	0.4255	0.4634
MQ2008	ApproxNDCG	0.2972	0.4222	0.3639	0.3337	0.3750	0.4039	0.4484	0.4585
	ListMLE	0.2801↓	0.3954↓	0.3482↓	0.3153↓	0.3508↓	0.3795↓	0.4236↓	0.4399↓
	ApproxNDCG (ST)	0.2783↓	0.4005	0.3571	0.3171↓	0.3457↓	0.3856	0.4234↓	0.4373↓
	Hinge	0.2642↓	0.3533↓	0.3350↓	0.3099↓	0.3087↓	0.3548↓	0.4035↓	0.4228↓
	MSE	0.2997	**0.4401**	0.3844↑	0.3431↑	0.3795	0.4177↑	0.4596↑	0.4709↑
	Pointwise KL (binomial)	0.2968	0.4222	0.3797↑	0.3375	0.3699	0.4126	0.4520	0.4626
	Pairwise KL (binomial)	0.2995	0.4388	0.3797↑	0.3375	0.3839	0.4179	0.4567	0.4681↑
	Pairwise KL (Gaussian)	**0.3019**	0.4375	**0.3852**↑	0.3398	**0.3871**	**0.4222**↑	0.4603↑	0.4697↑
	Listwise KL (Gaussian)	0.3008	0.4349	0.3814↑	**0.3457**↑	0.3827	0.4171	**0.4630**↑	**0.4729**↑
WEB30K	ApproxNDCG	0.3523	0.7504	0.7158	0.6936	0.5263	0.5090	0.5084	0.5798
	ListMLE	0.3354↓	0.7756↑	0.7362↑	0.7103↑	0.5217	0.5066	0.5068	**0.5983**↑
	ApproxNDCG (ST)	0.3707↑	0.7804↑	0.6994↓	0.6811↓	0.5769↑	0.5065	0.5040↓	0.5818↑
	Hinge	**0.3748**↑	**0.7885**↑	0.6899↓	0.6745↓	**0.5884**↑	0.5054	0.5046	0.5877↑
	MSE	0.3623↑	0.7709↑	0.7334↑	0.7093↑	0.5461↑	**0.5268**↑	**0.5248**↑	0.5921↑
	Pointwise KL (binomial)	0.3502	0.7596	0.7271↑	0.6985↑	0.5293	0.5136↑	0.5093	0.5862↑
	Pairwise KL (binomial)	0.3467↓	0.7812↑	**0.7446**↑	**0.7184**↑	0.5357↑	0.5242↑	0.5241↑	0.5971↑
	Pairwise KL (Gaussian)	0.3454↓	0.7753↑	0.7423↑	0.7166↑	0.5315	0.5209↑	0.5214↑	0.5971↑
	Listwise KL (Gaussian)	0.3523	0.7612↑	0.7258↑	0.7016↑	0.5322	0.5152↑	0.5141↑	0.5871↑
OHSUMED	ApproxNDCG	0.4981	0.5660	0.5377	0.5057	0.4764	0.4544	0.4371	0.3828
	ListMLE	0.4732	0.5377	0.4717↓	0.4321↓	0.4575	0.3972↓	0.3737↓	0.3220↓
	ApproxNDCG (ST)	0.4159↓	0.5000	0.4119↓	0.3660↓	0.4009	0.3330↓	0.3098↓	0.2882↓
	Hinge	0.4609	0.5660	0.4748↓	0.4566	0.4434	0.3930↓	0.3805↓	0.3530↓
	MSE	0.4376↓	0.5377	0.4465↓	0.4057↓	0.4340	0.3663↓	0.3417↓	0.3049↓
	Pointwise KL (binomial)	0.4875	0.5189	0.5031	0.5019	0.4481	0.4269	0.4312	0.3800
	Pairwise KL (binomial)	0.4206↓	0.4811	0.4151↓	0.3868↓	0.3915↓	0.3360↓	0.3235↓	0.2886↓
	Pairwise KL (Gaussian)	0.4520	0.5377	0.4403↓	0.4000↓	0.4481	0.3707↓	0.3481↓	0.2903↓
	Listwise KL (Gaussian)	**0.5248**	**0.6038**	**0.5692**	**0.5057**	**0.5189**	**0.4796**	**0.4456**	**0.3861**

Note: ↑ or ↓ indicate a statistically significant difference ($\alpha < .05$) with the performance obtained using the ApproxNDCG loss function. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

the ApproxNDCG loss and most of the other losses with the exception of the MSE loss on the MQ2008 collection which is the most competitive among the baselines. On the MSLR-WEB30K collection the best of the proposed Pairwise$_{\mathrm{KL(Bin)}}$ loss functions outperforms the best loss function overall on this collection, that is, the Hinge loss, in terms of P@3, P@5, nDCG@3, nDCG@5, and AP. We also observe that the performance of the model trained using the ApproxNDCG with ST loss is higher here compared to other collections. Indeed, on smaller collections

TABLE 2 Performance of the proposed LETOR neural model averaged over all topics

	Loss function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	ApproxNDCG	0.3175	0.4645	0.4342	0.4090	0.4125	0.4186	0.4217	0.4608
	ListMLE	0.3174	0.4639	0.4350	0.4110	0.4158	0.4176	0.4209	0.4591
	ApproxNDCG (ST)	0.2749↓	0.3895↓	0.3690↓	0.3579↓	0.3398↓	0.3459↓	0.3540↓	0.4061↓
	Hinge	0.2667↓	0.3806↓	0.3599↓	0.3486↓	0.3295↓	0.3331↓	0.3416↓	0.4017↓
	MSE	0.3162	0.4569	0.4334	0.4117	0.4122	0.4167	0.4208	0.4554↓
	Pointwise KL (binomial)	0.3185	0.4569	0.4297	0.4142	0.4119	0.4159	0.4246	0.4602
	Pairwise KL (binomial)	0.3147	0.4592	0.4356	0.4158	0.4116	0.4163	0.4225	0.4612
	Pairwise KL (Gaussian)	**0.3193**	**0.4758**	**0.4383**	**0.4201**↑	**0.4288**↑	**0.4235**	**0.4294**↑	**0.4655**
	Listwise KL (Gaussian)	0.3187	0.4616	0.4370	0.4178↑	0.4146	0.4204	0.4272	0.4621
MQ2008	ApproxNDCG	0.2915	0.4120	0.3588	0.3286	0.3661	0.3951	0.4381	0.4498
	ListMLE	0.3008↑	0.4413↑	0.3797↑	0.3362↑	0.3897↑	0.4186↑	0.4582↑	0.4694↑
	ApproxNDCG (ST)	0.2659↓	0.3890	0.3384↓	0.3077↓	0.3361	0.3676↓	0.4091↓	0.4297↓
	Hinge	0.2606↓	0.3457↓	0.3316↓	0.3092↓	0.3061↓	0.3521↓	0.4005↓	0.4194↓
	MSE	0.2991	0.4260	0.3835↑	0.3416↑	0.3744	0.4135↑	0.4568↑	0.4663↑
	Pointwise KL (binomial)	0.2999	0.4311	0.3831↑	0.3378↑	0.3782	0.4193↑	0.4554↑	0.4671↑
	Pairwise KL (binomial)	0.3016↑	0.4362↑	0.3814↑	0.3390↑	0.3871	0.4173↑	0.4584↑	0.4667↑
	Pairwise KL (Gaussian)	**0.3081**↑	**0.4503**↑	**0.3946**↑	**0.3449**↑	**0.4005**↑	**0.4314**↑	**0.4680**↑	**0.4769**↑
	Listwise KL (Gaussian)	0.2996	0.4311	0.3865↑	0.3439↑	0.3788	0.4210↑	0.4611↑	0.4717↑
WEB30K	ApproxNDCG	0.3522	0.7523	0.7171	0.6938	0.5263	0.5091	0.5082	0.5797
	ListMLE	0.3336↓	0.7732↑	0.7352↑	0.7102↑	0.5181	0.5052	0.5061	0.5980↑
	ApproxNDCG (ST)	0.1762↓	0.5103↓	0.5086↓	0.5060↓	0.2614↓	0.2773↓	0.2913↓	0.5041↓
	Hinge	**0.3853**↑	**0.7919**↑	0.7126	0.6998↑	**0.5926**↑	**0.5310**↑	**0.5316**↑	0.5887↑
	MSE	0.3638↑	0.7718↑	0.7344↑	0.7098↑	0.5483↑	0.5286↑	0.5262↑	0.5924↑
	Pointwise KL (binomial)	0.3545	0.7542	0.7245↑	0.6986↑	0.5305	0.5137↑	0.5106	0.5883↑
	Pairwise KL (binomial)	0.3475↓	0.7777↑	0.7406↑	0.7150↑	0.5352↑	0.5208↑	0.5212↑	0.5957↑
	Pairwise KL (Gaussian)	0.3506	0.7812↑	**0.7475**↑	**0.7202**↑	0.5402↑	0.5272↑	0.5267↑	**0.5988**↑
	Listwise KL (Gaussian)	0.3513	0.7529	0.7234↑	0.6987↑	0.5250	0.5129	0.5111	0.5870↑
OHSUMED	ApproxNDCG	0.5157	**0.6038**	**0.5377**	0.4906	**0.5189**	**0.4684**	0.4395	0.3729
	ListMLE	0.4616↓	0.5377	0.4748↓	0.4151↓	0.4387↓	0.4002↓	0.3667↓	0.3116↓
	ApproxNDCG (ST)	0.4584↓	0.5660	0.4465↓	0.4208↓	0.4623	0.3765↓	0.3613↓	0.3257↓
	Hinge	0.4278↓	0.5094↓	0.4465↓	0.4358↓	0.4009↓	0.3604↓	0.3517↓	0.3475↓
	MSE	0.4474↓	0.5566	0.4528↓	0.4075↓	0.4434↓	0.3776↓	0.3493↓	0.3160↓
	Pointwise KL (binomial)	0.4820	0.5094	0.5000	0.4830	0.4340	0.4244	0.4177	0.3727
	Pairwise KL (binomial)	0.4203↓	0.4717↓	0.4182↓	0.3811↓	0.3915↓	0.3376↓	0.3185↓	0.2887↓
	Pairwise KL (Gaussian)	0.4446↓	0.5094↓	0.4340↓	0.3906↓	0.4245↓	0.3647↓	0.3401↓	0.2933↓
	Listwise KL (Gaussian)	**0.5165**	0.5755	**0.5377**	**0.5113**	0.5000	0.4654	**0.4473**	**0.3873**↑

Note: The model is trained sampling the relevance labels from a Binomial distribution. ↑ or ↓ indicate a statistically significant difference ($\alpha < .05$) with the ApproxNDCG baseline. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

the amount of training data is probably not sufficient for the model to benefit from this training strategy. On the OHSUMED collection, the best loss function is the $\text{Listwise}_{\text{KL}(\mathcal{N}_{\text{mult}})}$, followed by the ApproxNDCG loss. This is due to the combination of a high proportion of relevant documents in the ranked lists and little amount of training data. For all the experiments on this collection, we rerank the top 150 documents returned by a LambdaMART model and, for this reason, we expect the representations of the documents in each ranked list to be more similar to each other rather than in other collections. Hence, approaches that consider multiple documents at a time, might have an advantage in finding the differences between them and providing more insightful information through their gradients to the model during training.

In Table 2, we report the results of the experiments training the proposed LETOR neural model using sampled relevance judgments as described in section 4. As we can see in this table, the relative performance of the neural model when relying on different loss functions remains similar and the best loss function overall is still the $\text{Pairwise}_{\text{KL}(\mathcal{N})}$ loss. However, the neural model performance is often higher in this case than in the previous experimental setup, regardless of the loss function used. This is true for at least one performance measure when using all but the $\text{Pairwise}_{\text{KL(Bin)}}$ loss on the MQ2007, and for all the proposed losses on the MQ2008 collection. We also observe a few performance improvements in the MSLR-WEB30K collection when using all loss functions with the exception of the ApproxNDCG with ST and the ListMLE loss. Finally, on the OHSUMED collection, we observe a performance improvement in at least one measure when using the ApproxNDCG, ApproxNDCG with ST, MSE, or $\text{Pairwise}_{\text{KL(Bin)}}$ loss. These results support our hypothesis that acknowledging and exploiting the possible inconsistencies in the training data can be a viable way to improve a LETOR model's performance.

In Table 3, we report the experiments of the crowdsourcing experiments on the COVID19-MLIA collection. In this case, we set the output size of the neural model to 3—the same number of relevance grades that we used for our annotation—and trained the model either (a) by aggregating—as explained in section 4—the model output probability scores and the collected relevance judgments distributions in the same way, obtaining one relevance score and relevance label to train the model with the previously evaluated loss functions, or (b) by training the model with the proposed $\text{Pointwise}_{\text{KL(Mul)}}$ loss function which can take into account the raw probability distributions over the three relevance classes. Note that, since relevance judgments distributions are only available for this collections, this is the only scenario where we can employ the proposed $\text{Pointwise}_{\text{KL(Mul)}}$ loss function.

We observe that the proposed $\text{Pointwise}_{\text{KL(Mul)}}$ loss function is the best loss function to train the neural model on this collection. This can also be partially due to the small size of the collection which favors pointwise and pairwise loss functions. However, $\text{Pointwise}_{\text{KL(Mul)}}$ still outperforms other pointwise loss functions such as the MSE and the $\text{Pointwise}_{\text{KL(Bin)}}$ losses by a sizable margin. The Hinge loss is the second best loss function on this collection, outperforming all other baselines. The results from this experiment further confirm our initial hypothesis on the feasibility of training a LETOR model using raw probability distributions as training data.

Finally, in Table 4 we report the performance comparison between the best performing systems relying on

TABLE 3 Performance of the proposed LETOR neural model on the COVID19-MLIA collection averaged over all topics

	Loss function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MLIA	ApproxNDCG	0.3313	0.4000	0.4667	0.4800	0.3556	0.3896	0.3973	0.3593
	ApproxNDCG (ST)	0.2544	0.3000	0.3333↓	0.3267↓	0.2708	0.2788	0.2820↓	0.3144
	ListMLE	0.2463↓	0.2667	0.3111↓	0.3133↓	0.2387	0.2640	0.2705↓	0.3162
	Hinge	0.3455	**0.5000**	0.4667	0.4467	**0.4330**	0.3830	0.3736	0.3822
	MSE	0.2675	0.3333	0.3333	0.3333↓	0.2917	0.2894	0.2900↓	0.3101↓
	Pointwise KL (multinomial)	**0.3377**	0.4333	**0.5000**	**0.4933**	0.3628	**0.4096**	**0.4054**	**0.3834**↑
	Pointwise KL (binomial)	0.2562	0.3333	0.3222↓	0.3533↓	0.2798	0.2684↓	0.2898↓	0.3122
	Pairwise KL (binomial)	0.2639	0.3667	0.3444↓	0.3200↓	0.3111	0.2919↓	0.2814↓	0.3023↓
	Pairwise KL (Gaussian)	0.2423↓	0.3333	0.3000↓	0.2867↓	0.2750	0.2515↓	0.2498↓	0.3062
	Listwise KL (Gaussian)	0.2521	0.3333	0.3111↓	0.3133↓	0.2715	0.2656↓	0.2653↓	0.3258

Note: ↑ or ↓ indicate a statistically significant difference ($\alpha < .05$) with the ApproxNDCG baseline. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

TABLE 4 Performance of different LETOR models (decision tree-based gradient boosted machine [GBM] model or the neural model [NM]) trained with the best-performing proposed loss functions averaged over all topics

	Loss function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	GBM – LambdaMART	0.3211	0.4669	0.4397	0.4167	0.4217	0.4243	0.4285	0.4646
	GBM – Pointwise KL (binomial)	**0.3233**	0.4752	0.4354	0.4167	0.4303	0.4207	0.4275	0.4591↓
	GBM – Listwise KL (Gaussian)	0.3219	0.4592	**0.4399**	0.4164	0.4152	0.4240	0.4267	0.4595
	NN – Pairwise KL (Gaussian)	0.3218	**0.4817**	0.4381	**0.4201**	**0.4350**	**0.4249**	**0.4318**	**0.4665**
	NN – Listwise KL (Gaussian)	0.3177	0.4657	0.4332	0.4145	0.4173	0.4192	0.4255	0.4634
MQ2008	GBM – LambdaMART	0.3045	0.4413	0.3869	0.3446	0.3858	0.4260	0.4664	0.4746
	GBM – Pointwise KL (binomial)	**0.3072**	**0.4439**	**0.3941**	**0.3464**	**0.3935**	**0.4325**	**0.4690**	0.4771
	GBM – Listwise KL (Gaussian)	0.2991	0.4362	0.3852	0.3444	0.3801	0.4214	0.4633	**0.4775**
	NN – Pairwise KL (Gaussian)	0.3019	0.4375	0.3852	0.3398	0.3871	0.4222	0.4603	0.4697
	NN – Listwise KL (Gaussian)	0.3008	0.4349	0.3814	0.3457	0.3827	0.4171	0.4630	0.4729
WEB30K	**GBM – LambdaMART**	**0.3955**	**0.7918**	0.7541	0.7288	**0.5925**	**0.5711**	**0.5670**	0.6299
	GBM – Pointwise KL (binomial)	0.3550↓	0.7789↓	0.7503	0.7261	0.5435↓	0.5344↓	0.5343↓	0.6326↑
	GBM – Listwise KL (Gaussian)	0.3861↓	**0.7918**	**0.7565**	**0.7323**↑	0.5825↓	0.5647↓	0.5615↓	**0.6347**↑
	NN – Pairwise KL (Gaussian)	0.3454↓	0.7753↓	0.7423↓	0.7166↓	0.5315↓	0.5209↓	0.5214↓	0.5971↓
	NN – Listwise KL (Gaussian)	0.3523↓	0.7612↓	0.7258↓	0.7016↓	0.5322↓	0.5152↓	0.5141↓	0.5871↓
OHSUMED	GBM – LambdaMART	0.4704	0.5283	0.4874	0.4906	0.4387	0.3980	0.4037	0.4175
	GBM – Pointwise KL (binomial)	0.5036	0.5755	0.5220	0.5000	0.4953	0.4474	0.4330	0.4210
	GBM – Listwise KL (Gaussian)	0.5139	0.5755	0.5314	**0.5151**	0.5000	0.4643↑	**0.4525**↑	**0.4243**
	NN – Pairwise KL (Gaussian)	0.4520	0.5377	0.4403	0.4000↓	0.4481	0.3707	0.3481↓	0.2903↓
	NN – Listwise KL (Gaussian)	**0.5248**	**0.6038**	**0.5692**↑	0.5057	**0.5189**	**0.4796**↑	0.4456	0.3861↓

Note: ↑ or ↓ indicate a statistically significant ($\alpha < .05$) difference with the LambdaMART model trained on the original relevance judgments. Best performance measures per collection are in bold as the loss function with the most best measures per collection.

the proposed neural model or on a decision tree-based GBM from the LightGBM library. We also report here the performance of a LambdaMART model trained with the LigthGBM library as a baseline. LambdaMART, and in particular its LightGBM implementation is generally considered a very competitive baseline in many other LETOR research works (Bruch, 2019; Bruch et al., 2019; Bruch et al., 2020; Pasumarthi et al., 2020; Zhen et al., 2021). We observe that the proposed probabilistic loss functions are in most of the cases able to improve the performance of a decision tree-based GBM model, surpassing the one of LambdaMART. We also observe that, on the MQ2007 and OHSUMED experimental collections, the proposed neural model outperforms LambdaMART and all the GBM-based models by a sizable margin.

TABLE 5 Performance of different LETOR models trained with the proposed loss functions

	Optimization function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	MAP
MQ2007	DASALC – Pointwise KL (binomial)	−0.0014	−0.0047	−0.0032	**+0.0032**	−0.0062	−0.0057	−0.0014	−0.0011
	DASALC – Pairwise KL (binomial)	**+0.0016**	**+0.0065**	**+0.0041**	**+0.0067**↑	**+0.0059**	**+0.0027**	**+0.0050**	**+0.0060**↑
	DASALC – Pairwise KL (Gaussian)	**+0.0007**	**+0.0035**	**+0.0047**	**+0.0080**↑	**+0.0027**	**+0.0038**	**+0.0062**	**+0.0062**↑
	DASALC – Listwise KL (Gaussian)	−0.0015	−0.0089	−0.0006	**+0.0020**	−0.0080	−0.0014	**+0.0004**	−0.0002
	TRANSFORMER – Pointwise KL (binomial)	**+0.0099**↑	**+0.0035**	**+0.0106**↑	−0.0007	**+0.0098**	**+0.0112**↑	**+0.0033**	−0.0005
	TRANSFORMER – Pairwise KL (binomial)	**+0.0084**↑	**+0.0047**	**+0.0091**	**+0.0041**	**+0.0095**	**+0.0078**	**+0.0033**	**+0.0025**
	TRANSFORMER – Pairwise KL (Gaussian)	−0.0022	−0.0154	−0.0022	−0.0045	−0.0115	−0.0060	−0.0087	−0.0039
	TRANSFORMER – Listwise KL (Gaussian)	**+0.0015**	**+0.0024**	**+0.0002**	−0.0022	**+0.0038**	**+0.0007**	−0.0012	**+0.0001**
MQ2008	DASALC – Pointwise KL (binomial)	**+0.0116**↑	**+0.0255**	**+0.0136**↑	**+0.0128**↑	**+0.0236**	**+0.0172**↑	**+0.0226**↑	**+0.0179**↑
	DASALC – Pairwise KL (binomial)	**+0.0162**↑	**+0.0306**↑	**+0.0234**↑	**+0.0158**↑	**+0.0344**↑	**+0.0284**↑	**+0.0275**↑	**+0.0236**↑
	DASALC – Pairwise KL (Gaussian)	**+0.0114**↑	**+0.0230**	**+0.0170**↑	**+0.0089**↑	**+0.0249**↑	**+0.0210**↑	**+0.0201**↑	**+0.0181**↑
	DASALC – Listwise KL (Gaussian)	**+0.0160**↑	**+0.0268**↑	**+0.0238**↑	**+0.0171**↑	**+0.0268**↑	**+0.0277**↑	**+0.0279**↑	**+0.0224**↑
	TRANSFORMER – Pointwise KL (binomial)	**+0.0119**↑	**+0.0242**	**+0.0162**↑	**+0.0056**	**+0.0198**	**+0.0231**↑	**+0.0170**↑	**+0.0146**↑
	TRANSFORMER – Pairwise KL (binomial)	**+0.0129**↑	**+0.0293**↑	**+0.0174**↑	**+0.0074**	**+0.0281**↑	**+0.0235**↑	**+0.0178**↑	**+0.0143**↑
	TRANSFORMER – Pairwise KL (Gaussian)	**+0.0022**	**+0.0051**	**+0.0230**↑	**+0.0018**	−0.0064	**+0.0173**↑	**+0.0051**	**+0.0046**
	TRANSFORMER – Listwise KL (Gaussian)	**+0.0117**↑	**+0.0281**↑	**+0.0170**↑	**+0.0094**↑	**+0.0191**	**+0.0239**↑	**+0.0213**↑	**+0.0190**↑
WEB30K	DASALC – Pointwise KL (binomial)	−0.0083↓	−0.0262↓	−0.0152↓	−0.0144↓	−0.0192↓	−0.0175↓	−0.0173↓	−0.0023↓
	DASALC – Pairwise KL (binomial)	−0.0120↓	**+0.0165**↑	**+0.0209**↑	**+0.0216**↑	**+0.0028**	**+0.0062**↑	**+0.0077**↑	**+0.0158**↑
	DASALC – Pairwise KL (Gaussian)	−0.0169↓	**+0.0094**↑	**+0.0168**↑	**+0.0182**↑	−0.0074	−0.0009	**+0.0020**	**+0.0148**↑
	DASALC – Listwise KL (Gaussian)	−0.0067↓	−0.0263↓	−0.0175↓	−0.0138↓	−0.0163↓	−0.0139↓	−0.0125↓	−0.0039↓
	TRANSFORMER – Pointwise KL (binomial)	−0.0238↓	**+0.0138**↑	−0.0028	−0.0146↓	−0.0099↓	−0.0193↓	−0.0243↓	−0.0082↓
	TRANSFORMER – Pairwise KL (binomial)	−0.0440↓	**+0.0135**↑	**+0.0142**↑	**+0.0101**↑	−0.0386↓	−0.0235↓	−0.0197↓	**+0.0042**↑
	TRANSFORMER – Pairwise KL (Gaussian)	−0.0419↓	−0.0041	−0.0001	−0.0003	−0.0416↓	−0.0336↓	−0.0279↓	**+0.0019**↑
	TRANSFORMER – Listwise KL (Gaussian)	−0.1000↓	−0.0798↓	−0.0826↓	−0.0786↓	−0.1242↓	−0.1136↓	−0.1068↓	−0.0328↓
OHSUMED	DASALC – Pointwise KL (binomial)	**+0.0232**	0.0000	**+0.0220**	**+0.0208**	**+0.0189**	**+0.0350**	**+0.0354**	**+0.0328**↑
	DASALC – Pairwise KL (binomial)	**+0.0022**	**+0.0094**	−0.0346	−0.0547↓	**+0.0283**	−0.0141	−0.0260	−0.0107
	DASALC – Pairwise KL (Gaussian)	−0.0176	−0.0189	−0.0629	−0.0717↓	**+0.0047**	−0.0408	−0.0480↓	−0.0140
	DASALC – Listwise KL (Gaussian)	**+0.0266**	**+0.0472**	−0.0094	−0.0189	**+0.0566**	**+0.0178**	**+0.0050**	**+0.0026**
	TRANSFORMER – Pointwise KL (binomial)	**+0.1563**↑	**+0.1321**↑	**+0.1478**↑	**+0.1434**↑	**+0.1745**↑	**+0.1604**↑	**+0.1573**↑	**+0.0825**↑
	TRANSFORMER – Pairwise KL (binomial)	**+0.0530**	−0.0189	**+0.0409**	**+0.0434**	**+0.0283**	**+0.0477**	**+0.0474**	**+0.0393**↑
	TRANSFORMER – Pairwise KL (Gaussian)	**+0.0845**↑	**+0.0755**	**+0.0786**	**+0.0302**	**+0.1038**	**+0.0909**	**+0.0673**	**+0.0298**
	TRANSFORMER – Listwise KL (Gaussian)	**+0.1485**↑	**+0.1132**	**+0.1604**↑	**+0.1358**↑	**+0.1604**↑	**+0.1559**↑	**+0.1441**↑	**+0.0768**↑

Note: We indicate with ↑ or ↓ a statistically significant (p-value <.05) difference with the performance obtained by the same model trained with the ApproxNDCG loss function on the original relevance judgments available in each dataset. We indicate in bold all the cases where we observe a performance improvement over the respective baseline.

To conclude our evaluation, in Table 5 we report a comparison of the performance improvements obtained using the proposed loss functions to train two state-of-the-art models, that is, DASALC (Zhen et al., 2021) and a simpler transformer model similar to the one used in Pobrotyn et al. (2020). More specifically, we report—for each model—the performance difference when training it using one of the proposed loss functions or the ApproxNDCG loss. From the results reported in Table 5, we observe that in the majority of our tests the proposed family of loss functions allows a better performance of both DASALC and the transformer model compared to the ApproxNDCG loss, especially on the MQ2008 collection. On the MQ2007 collection, we observe performance improvements on all evaluation measures, with some differences between the two models. In this case, the two best loss functions to train the DASALC model are the pairwise ones, while the transformer model benefits more from the pointwise and listwise losses. This effect is observed because of the simpler nature of the latter model and its lower number of parameters to train. On the MSLR-WEB30K collection, we observe fewer performance improvements compared to other datasets. However, the performance differences are more similar across models. Here, the best loss function is the Pairwise KL (Binomial), which leads to statistically significant performance improvements for both the considered neural LETOR models. Finally, on the OHSUMED collection we observe a significant performance improvement on almost all performance measures when training the transformer model with any of the proposed loss functions. On the other hand, the DASALC model shows fewer performance improvements, likely because of the higher model complexity combined with the small number of topics in this collection. Hence, the best loss functions in this case are the Listwise KL (Gaussian) and the Pointwise KL (Binomial), the same that performed the best on the proposed transformer-based neural LETOR model.

Observing all the above results, we notice that the performance of different models trained with the proposed loss functions varies according to the experimental collection used. Indeed, we conducted our evaluation selecting a number of datasets with different characteristics to show all the strengths and weaknesses of each of the proposed loss functions and to show the best scenarios where they can be employed. For example, for what concerns the MQ2007 and M2008 collections—with 1,700 and 800 topics, respectively—we always ranked 128 (or less) documents for each topic. On the other hand, for the MSLR-WEB30K and OHSUMED collection—with 30,000 and 106 topic each—we considered a subset of size 128 (or smaller if fewer documents were available) of all documents provided for each topic in the dataset. We selected these subsets by ranking all the available documents for each topic with a lambdaMART model, and then then discarding the items with a rank higher than 128.

As a consequence of the diversity of the considered collections, we observe a few differences in the performance of the proposed loss functions in each of our experiments. On medium-sized collections—where documents were not filtered prior to ranking them—such as the MQ2007 and MQ2008, we observe overall a similar performance of all the proposed pointwise, pairwise and listwise loss functions, with sizeable differences noticeable only with certain evaluation metrics such as P@{1-5} and nDCG@{1-5}. On the MSLR-WEB30K dataset we observe a similar trend, here however we notice a more sizeable performance difference between the pairwise and listwise loss functions we propose and the pointwise variant—especially when using them to train the DASALC and the proposed LETOR model. In this case, the larger availability of training data and the document filtering step we applied before reranking contribute to the better performance of more complex training strategies—that is, pairwise and listwise loss functions.

Finally, on the OHSUMED collection—the smallest of all the datasets we considered—we observe a different situation. Here, despite the small number of topics available, the best-performing loss function is the listwise KL (Gaussian). This is likely due to the document pruning step we perform prior to ranking. This step promotes the selection of documents which are more similar to each other than in the previous cases and therefore gives an advantage to loss functions which compare multiple items at a time, that is, the proposed listwise loss function. The second best performing probabilistic loss function however is the pointwise KL one. Indeed, this loss function allows a LETOR model to learn better than other training strategies when a few training examples are available—since it considers each document-topic pair as a valid training data-point, simplifying the training objective.

6 | CONCLUSIONS AND FUTURE WORK

We presented different strategies to train a LETOR model relying on relevance judgments distributions. We introduced five different loss functions relying on the KL divergence between distributions, opening new possibilities for the training of LETOR models. The proposed loss functions were evaluated on a newly proposed neural model, two transformer-based neural LETOR systems, and on a decision tree-based GBM model—the same model employed by the popular LambdaMART algorithm

(Burges, 2010)—over a number of experimental collections of different sizes.

We compared the performance of the proposed loss functions to the most representative loss functions in the IR domain: the pointwise mean squared error (MSE) loss (Liu et al., 2018), the pairwise Hinge loss (Guo et al., 2016), the listwise ApproxNDCG loss (Qin, Liu, & Li, 2010) with and without the stochastic treatment (ST) proposed in (Bruch et al., 2020) and the ListMLE loss (Xia et al., 2008). In our experiments, the proposed loss functions outperformed the aforementioned baselines in several cases and gave a significant performance boost to LETOR approaches—especially the ones based on neural models—allowing them to also outperform other strong baselines in the LETOR domain such as the LightGBM implementation of LambdaMART (Burges, 2010; Zhen et al., 2021).

We also evaluated the option of training a neural LETOR model simulating the distribution of relevance judgments for each document-topic pair in the training data. The results from this experiment further confirmed our hypothesis on the utility of using relevance judgments distributions to train a LETOR model, showing performance improvements across different measures.

Finally, we conducted a crowdsourcing experiment on the COVID-19 MLIA collection, building a new LETOR collection with real relevance judgments distributions. We share this collection and labels to be used for the development and evaluation of other LETOR approaches that will follow the proposed training paradigm. These experiments consolidated our hypothesis and showed encouraging results on the usage of probabilistic loss functions also on this dataset. As future work, we plan to further develop the proposed neural architecture to take advantage of probability distributions on model weights—that is, employing Bayesian neural layers (Anoop et al., 2015)—and to evaluate the performance of the proposed loss functions to train a model based on implicit user feedback signals such as clicks and dwell time.

ACKNOWLEDGMENT

Open access funding enabled and organized by Projekt DEAL.

ORCID

Alberto Purpura https://orcid.org/0000-0003-1701-7805

ENDNOTES

[1] http://eval.covid19-mlia.eu

[2] http://eval.covid19-mlia.eu

[3] https://lucene.apache.org

[4] https://bitbucket.org/frrncl/gopal/src/master/src/main/resources/

[5] The nDCG formulation we employ is the one used by the TREC_eval tool. https://github.com/usnistgov/trec_eval

REFERENCES

Agrawal, R. (2020). Finite-sample concentration of the multinomial in relative entropy. *IEEE Transactions on Information Theory*, *66*, 6297–6302.

Alonso, O. (2019). The practice of crowdsourcing. In *Synthesis lectures on information concepts, retrieval, and services* (p. 11). Morgan & Claypool Publishers.

Anoop, K., Rathod, V., Murphy, K., & Welling, M. (2015). Bayesian dark knowledge. In *Proceedings of the 28th international conference on neural information processing systems* (pp. 3438–3446). ACM.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Bruch, S. (2019). An alternative cross entropy loss for learning-to-rank. arXiv:1911.09798.

Bruch, S., Han, S., Bendersky, M., & Najork, M. (2020). A stochastic treatment of learning to rank scoring functions. In *Proceedings of the 13th international conference on web search and data mining* (pp. 61–69). ACM.

Bruch, S., Zoghi, M., Bendersky, M., & Najork, M. (2019). Revisiting approximate metric optimization in the age of deep neural networks. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval* (pp. 1241–1244). ACM.

Burges, C. (2010). *From RankNet to LambdaRank to LambdaMART: An overview* (Microsoft Research Technical Report MSR-TR-2010-82). Microsoft Research.

Chapelle, O., Metlzer, D., Zhang, Y., & Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on information and knowledge management* (pp. 621–630). ACM.

Chen, W., Liu, T., Lan, Y., Ma, Z., & Li, H. (2009). Ranking measures and loss functions in learning to rank. In *Proceedings of the 22nd international conference on neural information processing systems*. ACM.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Ferrante, M., Ferro, N., & Losiouk, E. (2020). How do interval scales help us with better understanding IR evaluation measures? *IR Journal*, *23*, 289–317.

Ferrante, M., Ferro, N., & Piazzon, L. (2020). s-AWARE: Supervised measure-based methods for crowd-assessors combination. In *11th international conference of the CLEF Association*. ACM.

Fuhr, N. (2018). Some common mistakes in ir evaluation, and how they can be avoided. *SIGIR Forum*, *51*, 32–41.

Guo, J., Fan, Y., Ai, Q., and Croft, W. (2016). A deep relevance matching model for ad-hoc retrieval. *Proceedings of 25th international conference of the CIKM*. ACM .

Hosseini, M., Cox, I., Milić-Frayling, N., Kazai, G., & Vinay, V. (2012). On aggregating labels from multiple crowd workers to infer relevance of documents. In *Proceedings of the 34th European conference on advances in information retrieval* (pp. 182–194). ACM.

Ibrahim, M., & Carman, M. (2016). Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM TOIS*, *34*, 1–38.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 3149–3157). ACM.

Köppel, M., Segner, A., Wagener, M., Pensel, L., Karwath, A., & Kramer, S. (2019). Pairwise learning to rank by neural networks revisited: Reconstruction, theoretical analysis and practical performance. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer.

Lease, M. & Kazai, G. (2011). *Overview of the TREC 2011 crowdsourcing track*. Paper presented at the proceedings of TREC, National Institute of Standards and Technology.

Liu, X., Van De Weijer, J., & Bagdanov, A. (2018). Leveraging unlabeled data for crowd counting by learning to rank. In *IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. IEEE.

MacAvaney, S., Yates, A., Cohan, A., & Goharian, N. (2019). CEDR: Contextualized embeddings for document ranking. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval* (pp. 1101–1104). ACM.

Onal, K., Zhang, Y., Altingovde, I., Rahman, M., Karagoz, P., Braylan, A., Dang, B., Chang, H., Kim, H., McNamara, Q., Angert, A., Banner, E., Khetan, V., Mcdonnell, T., Nguyen, A., Xu, D., Wallace, B., Rijke, M., & Lease, M. (2018). Neural information retrieval: At the end of the early years. *Information Retrieval*, *21*, 111–182.

Pang, L., Lan, Y., Guo, J., Xu, J., & Cheng, X. (2016). A study of matchpyramid models on ad-hoc retrieval. arXiv:1606.04648.

Pasumarthi, R., Zhuang, H., Wang, X., Bendersky, M., & Najork, M. (2020). Permutation equivariant document interaction network for neural learning to rank. In *Proceedings of the 2020 ACM SIGIR on international conference on theory of information retrieval* (pp. 145–148). ACM.

Pobrotyn, P., Bartczak, T., Synowiec, M., Białobrzeski, R., & Bojar, J. (2020). Context-aware learning to rank with self-attention. arXiv:2005.10084.

Qin, T. & Liu, T. (2013). Introducing letor 4.0 datasets. arXiv: 1306.2597.

Qin, T., Liu, T., & Li, H. (2010). A general approximation framework for direct optimization of information retrieval measures. *IR Journal*, *4*, 375–397.

Qin, T., Liu, T., Xu, J., & Li, H. (2010). Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, *13*, 346–374.

Ravana, S. D., & Moffat, A. (2009). Score aggregation techniques in retrieval experimentation. In *Proceedings of the 20th Australasian database conference*. Australian Computer Society.

Smucker, M., Kazai, G., & Lease, M. (2012). *Overview of the TREC 2012 crowdsourcing track*. Paper presented at the proceedings of TREC.

Smucker, M., Kazai, G., & Lease, M. (2013). *Overview of the TREC 2013 crowdsourcing track*. Paper presented at the proceedings of TREC.

Sun, S. & Duh, K. (2020). Modeling document interactions for learning to rank with regularized self-attention. arXiv: 2005.03932.

Tax, N., Bockting, S., & Hiemstra, D. (2015). A cross-benchmark comparison of 87 learning to rank methods. *IP&M*, *51*, 757–772.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *31st conference on neural information processing system*. Curran Associates, Inc.

Xia, F., Liu, T., Wang, J., Zhang, W., & Li, H. (2008). Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th international conference on machine learning* (pp. 1192–1199). ACM.

Zamani, H., & Croft, W. (2018). On the theory of weak supervision for information retrieval. In *Proceedings of the 2018 ACM SIGIR international conference on theory of information retrieval*. ACM.

Zhen, Q., Le, Y., Honglei, Z., Yi, T., Rama, K., Xuanhui, W., Michael, B., & Marc, N. (2021). Neural rankers are hitherto outperformed by gradient boosted decision trees. In *Proceedings of ICLR*. OpenReview.

Zhuang, H., Wang, X., Bendersky, M., Grushetsky, A., Wu, Y., Mitrichev, P., Sterling, E., Bell, N., Ravina, W., & Qian, H. (2020). Interpretable learning-to-rank with generalized additive models. arXiv:2005.02553.

Zhuang, H., Wang, X., Bendersky, M., & Najork, M. (2020). Feature transformation for neural ranking models. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. ACM.

How to cite this article: Purpura, A., Silvello, G., & Susto, G. A. (2022). Learning to rank from relevance judgments distributions. *Journal of the Association for Information Science and Technology*, 1–17. https://doi.org/10.1002/asi.24629