

Tune and mix: learning to rank using ensembles of calibrated multi-class classifiers

Róbert Busa-Fekete · Balázs Kégl · Tamás Élétető ·
György Szarvas

Received: 10 January 2012 / Accepted: 13 April 2013
© The Author(s) 2013

Abstract In *subset ranking*, the goal is to learn a ranking function that approximates a gold standard partial ordering of a set of objects (in our case, a set of documents retrieved for the same query). The partial ordering is given by relevance labels representing the relevance of documents with respect to the query on an absolute scale. Our approach consists of three simple steps. First, we train standard multi-class classifiers (AdaBoost.MH and multi-class SVM) to discriminate between the relevance labels. Second, the posteriors of multi-class classifiers are calibrated using probabilistic and regression losses in order to estimate the Bayes-scoring function which optimizes the Normalized Discounted Cumulative Gain (NDCG). In the third step, instead of selecting the best multi-class hyperparameters and the best calibration, we mix all the learned models in a simple ensemble scheme.

Editors: Eyke Hüllermeier and Johannes Fürnkranz.

Most of G. Szarvas's work was done while working at UKP Lab, TU Darmstadt & RGAI of the Hungarian Acad. Sci. and University of Szeged.

R. Busa-Fekete (✉)

Linear Accelerator Laboratory (LAL), University of Paris-Sud, CNRS, Orsay, 91898, France
e-mail: busarobi@gmail.com

R. Busa-Fekete

Research Group on Artificial Intelligence (RGAI) of the Hungarian Academy of Sciences and University of Szeged, Aradi vértanúk tere 1., 6720 Szeged, Hungary

B. Kégl

Linear Accelerator Laboratory (LAL) and Computer Science Laboratory (LRI), University of Paris-Sud, CNRS, Orsay, 91898, France
e-mail: balazs.kegl@gmail.com

T. Élétető

Ericsson Hungary, Könyves Kálmán krt. 11.B, 1097 Budapest, Hungary
e-mail: tamas.elteto@ericsson.com

G. Szarvas

Nuance Communications Germany GmbH, Kackertstrasse 10, 52072 Aachen, Germany
e-mail: gyorgy.szarvas@gmail.com

Our extensive experimental study is itself a substantial contribution. We compare most of the existing learning-to-rank techniques on all of the available large-scale benchmark data sets using a standardized implementation of the NDCG score. We show that our approach is competitive with conceptually more complex listwise and pairwise methods, and clearly outperforms them as the data size grows. As a technical contribution, we clarify some of the confusing results related to the ambiguities of the evaluation tools, and propose guidelines for future studies.

Keywords Learning-to-rank · Multi-class classification · Class Probability Calibration · Regression Based Calibration · Ensemble methods

1 Introduction

In the past, lists of results obtained by information retrieval (IR) systems were ranked by probabilistic models, such as the BM25 measure (Robertson and Zaragoza 2009), based on a small number of attributes (the frequency of query terms in the document, in the collection, etc.). The parameters of these models were usually tuned by trial and error. As the number of useful features increased, these manually crafted models became increasingly laborious to configure. Alternatively, one can use as many (possibly redundant) attributes as possible, and employ machine learning (ML) techniques to induce a ranking model. This approach alleviates the human effort needed to design the ranking function and also provides a natural way to directly optimize the retrieval performance for any particular application and evaluation metric. As a result, learning to rank has gained considerable academic interest in the past decade.

ML-based ranking systems are traditionally classified into three categories. In the simplest *pointwise* approach, the instances are first assigned a relevance score using classical regression or classification techniques and then ranked by posterior scores obtained using the trained model (Li et al. 2007). In the *pairwise* approach, the order of pairs of instances is treated as a binary label and learned by a classification method (Freund et al. 2003). Finally, in the most complex *listwise* approach, the fully ranked lists are learned by a tailor-made learning method which seeks to optimize a ranking-specific evaluation metric during the learning process (Valizadegan et al. 2009).

In web page ranking or subset ranking (Cossock and Zhang 2008) the training data is given in the form of (query, document, relevance label) triplets. The relevance label of a training instance indicates the usefulness of the document to its corresponding query, and the ranking for a particular query is usually evaluated via the (normalized) Discounted Cumulative Gain ((N)DCG) or the Expected Reciprocal Rank (ERR) (Chapelle et al. 2009) measure. It is rather difficult to extend classical learning methods to directly optimize these evaluation metrics. Nevertheless, since the DCG can be bounded by the zero-one-loss (Li et al. 2007), traditional classification error can be regarded as a surrogate function for DCG.

Calibrating the output of a classifier is crucial in applications with quality measures different from the zero-one-error (Niculescu-Mizil and Caruana 2005). Our approach is based on the calibration of a multi-class classifier learned by ADABOOST.MH (Freund and Schapire 1997) or a multi-class Support Vector Machine (Crammer and Singer 2001) (SVM). The class labels are assumed to be random variables and the goal is to estimate the probability distribution of the class labels given a feature vector representing the (query, document) pair. A key novelty in our approach is that, instead of using a single calibration technique, we apply several methods to estimate the same probability distribution and we

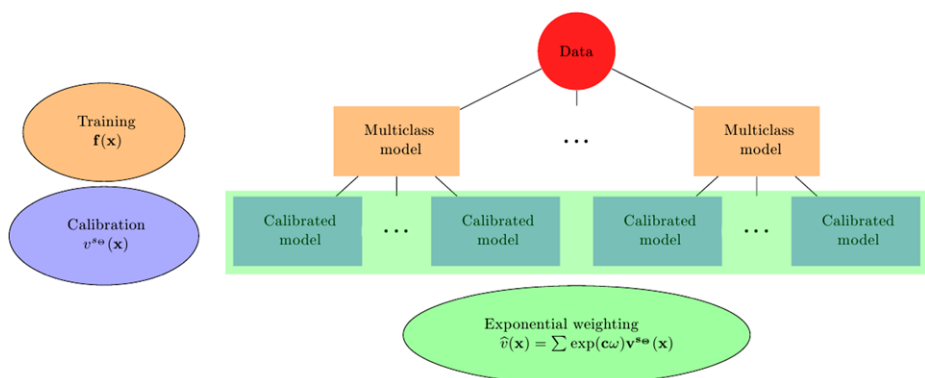


Fig. 1 The schematic overview of our approach. On the first level, a multi-class method (ADABOOST.MH) is trained using different hyperparameter settings. Then we calibrate the multi-class models in several ways to obtain diverse scoring functions. In the last step, we aggregate the scoring functions using an exponential weighting

combine these estimates in a final step. Our approach is motivated by the Bayesian paradigm and the Minimum Description Length principle (Rissanen 1983), both of which suggest that it is usually more efficient to mix different conditional distributions according to a prior than to select one “optimal” distribution.

We use both *regression-based calibration* (RBC) and *class-probability-based calibration* (CPC) to transform the output scores of multi-class classifiers into relevance label estimates. In RBC the real-valued scores are obtained by regressing the relevance grades against the output score vector, whereas in CPC the posterior probability distribution is used to approximate the so-called Bayes-scoring function (Cossock and Zhang 2008), which is shown to optimize the expected DCG in a probabilistic setup.

The proper choice of the weighting of the set of conditional distributions obtained by the calibrated classifiers is an important decision in practice. In this paper, we use an exponential scheme based on the quality of the rankings implied by the conditional distributions (via their corresponding conditional ranking functions) which is theoretically better-founded than the uniformly weighted aggregation used by MCRANK (Li et al. 2007).

Figure 1 offers a structural overview of our system. It is based on a set of standard techniques of (i) multiclass classification, (ii) output score calibration, and (iii) an exponentially weighted forecaster that is used to combine the various hypotheses. The computationally expensive first two steps belong to the simplest, pointwise category of learning-to-rank models, whereas the final mixing step optimizes a listwise objective function.

Most of the learning-to-rank methods had been tested on (and tuned to) the relatively small LETOR data sets, published by Microsoft. Recently, two larger benchmark data sets have been published by Yahoo and Microsoft. In addition to the mandatory comparison of our approach with the state of the art of learning to rank, we carried out rigorous and exhaustive experiments to compare the methods with each other on these new sets. To our knowledge, this is the first large-scale study of this kind. Reproducibility was an important goal, so we give all the algorithmic details necessary to repeat the experiments. For the same reason, we only tested methods that can be implemented easily (without ambiguity) or for which an open source implementation is available. Our most important finding is that pointwise methods are more competitive on large data sets than had been previously thought, and that they scale better as the data sets grow.

The paper is organized as follows. In Sect. 2, we review some approaches that are the most similar to ours. In Sect. 3, we define the ranking problem formally and introduce our notation. Section 4 contains theoretical results that motivated the calibration techniques described in Sect. 5. We explain the final ensemble step in Sect. 6. Section 7 then contains our experimental results. In Sect. 8 we draw some pertinent conclusions and briefly suggest some ideas for future study.

2 Related work

Among the plethora of ranking algorithms, our approach is probably the closest to the MCRANK algorithm (Li et al. 2007). We both use a multi-class classification algorithm at the core (they use gradient boosting, whereas we apply ADABOOST.MH and multi-class SVM). The chief novelties of our approach are that we use decision product base classifiers besides the popular decision trees and that we apply several different calibration approaches. Both elements add more diversity to our models that we exploit by using a final meta-ensemble technique. In addition, MCRANK's implementation is inefficient in the sense that the number of decision trees trained in each boosting iteration is as large as the number of different classes in the data set.

Even though MCRANK is not regarded as a state-of-the-art method itself, its importance is unquestionable. It can be viewed as a milestone that proved the *raison d'être* of classification-based learning-to-rank methods. It attracted the attention of researchers working on learning-to-rank to classification-based ranking algorithms. The most remarkable method motivated by MCRANK is LAMBDA MART (Wu et al. 2010), which adapts the MART algorithm to the subset ranking problem. The winning entry of Track 1 in the Yahoo! Learning-to-rank Challenge (Chapelle et al. 2011) was largely based on this method.

In the Yahoo! challenge (Chapelle et al. 2011), the general conclusion was that listwise and pairwise methods achieved the best scores in general, but tailor-made pointwise approaches also achieved very competitive results. In particular, the approach presented here is based on the system we devised when we participated in Yahoo! Learning-To-Rank Challenge (Busa-Fekete et al. 2011a). A preliminary version of this study appeared in a conference paper (Busa-Fekete et al. 2011b), but the results we present here are more general. The main contributions of this paper compared to Busa-Fekete et al. (2011a, 2011b) are that (1) we evaluate our approach on all publicly available benchmark data sets and investigate several issues experimentally; (2) we present a novel calibration approach, namely the sigmoid-based class probability calibration (CPC), which is theoretically better grounded than regression-based calibration; (3) we rigorously and exhaustively compare state-of-the-art open source rankers with a special emphasis on reproducibility; and (4) we test multi-class SVM as an alternative to ADABOOST.MH. As a theoretical motivation, we also provide an upper bound on the difference between the DCG value of the Bayes optimal scoring function and the DCG value achieved by its estimate using CPC in terms of the Kullback-Leibler divergence.

In a recent article (Kotłowski et al. 2011), it was shown that the bipartite ranking problem can be cast as a binary classification problem and that the rank regret of a classifier can be upper bounded by its regret for exponential loss and logistic loss. This result explains why many classifiers optimizing the exponential or logistic losses also perform well as rankers. For example, in Cortes and Mohri (2005) the authors showed experimentally that ADABOOST works well in the bipartite ranking problem.

3 Definition of the ranking problem

In this section we formally define the learning-to-rank problem and introduce the notation that will be used in the rest of this paper. First, let us assume that we are given a set of *query objects* $\mathbf{D} = \{\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(M)}\}$. Each query object $\mathcal{D}^{(k)}$ consists of a set of $n^{(k)}$ pairs

$$\mathcal{D}^{(k)} = \{(\mathbf{x}_1^{(k)}, \ell_1^{(k)}), \dots, (\mathbf{x}_{n^{(k)}}^{(k)}, \ell_{n^{(k)}}^{(k)})\}.$$

The real-valued feature vectors $\mathbf{x}_i^{(k)}$ represent the k th query and the i th document received as a potential hit for the query.¹ The *label index* $\ell_i^{(k)}$ of the query-document pair $\mathbf{x}_i^{(k)}$ is an integer between 1 and K . We assume that we are given a set of numerical *relevance grades*

$$\mathcal{Z} = \{z_1, \dots, z_K\}.$$

The relevance grade $z_{\ell_i^{(k)}}^{(k)} = z_{\ell_i^{(k)}}^{(k)}$ expresses the relevance of the i th document to the k th query on a numerical scale. A popular choice for the numerical relevance grades is

$$z_\ell = 2^{\ell-1} - 1 \quad (1)$$

for all $\ell = 1, \dots, K$.

The goal of the ranker is to output a permutation $\mathbf{j}^{(k)} = (j_1, \dots, j_{n^{(k)}})$ over the integers $(1, \dots, n^{(k)})$ for each query object $\mathcal{D}^{(k)}$. A widely used empirical measure of the quality of the permutation $\mathbf{j}^{(k)}$ is the Discounted Cumulative Gain (DCG)

$$\text{DCG}(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}) = \sum_{i=1}^{n^{(k)}} c_i z_{j_i}^{(k)}, \quad (2)$$

where c_i is the *discount factor* of the i th document in the permutation. The most common discount factor applied is

$$c_i = \frac{1}{\log(1+i)}. \quad (3)$$

The rationale behind this formula is that a user will be increasingly happy when he/she finds relevant documents early in the permutation. To normalize the DCG between 0 and 1, (2) is usually divided by the DCG score of the best permutation (NDCG). It is also a common practice to truncate the sum (2) at n_{\max} , defining the $\text{DCG}_{n_{\max}}$ and $\text{NDCG}_{n_{\max}}$ scores. The reason for this is that a user should rarely browse beyond the first page of search results containing the first n_{\max} hits.

We will treat the label index $\ell_i^{(k)}$ as a random variable with a conditional discrete probability distribution

$$p^*(\ell | \mathbf{x}_i^{(k)}) = P(\ell_i^{(k)} = \ell | \mathbf{x}_i^{(k)})$$

over the label indices ℓ for document i of query k . The *Bayes-scoring function*

$$v^*(\mathbf{x}_i^{(k)}) = \mathbb{E}\{z | \mathbf{x}_i^{(k)}\} = \sum_{\ell=1}^K z_\ell p^*(\ell | \mathbf{x}_i^{(k)})$$

¹When there is no possible confusion, we will omit the query index and simply write \mathbf{x}_i for the i th document of a given query.

is the conditional expectation of the relevance grade given the (query, document) pair $\mathbf{x}_i^{(k)}$. Then the *expected* DCG for any permutation $\mathbf{j}^{(k)}$ is

$$\text{EDCG}(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}) = \sum_{i=1}^{n^{(k)}} c_i \mathbb{E}\{z | \mathbf{x}_{j_i}^{(k)}\} = \sum_{i=1}^{n^{(k)}} c_i v^*(\mathbf{x}_{j_i}^{(k)}).$$

Let the *Bayes optimal* permutation $\mathbf{j}^{(k)*} = (j_1^{(k)*}, \dots, j_{n^{(k)}}^{(k)*})$ over $\mathcal{D}^{(k)}$ be the one which maximizes the expected DCG; that is,

$$\mathbf{j}^{(k)*} = \arg \max_{\mathbf{j}^{(k)}} \text{EDCG}(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}).$$

According to Theorem 1 stated in Cossock and Zhang (2008), $\mathbf{j}^{(k)*}$ has the property that if $c_i > c_{i'}$, then for the Bayes-scoring function we have $v^*(\mathbf{x}_{j_i^{(k)*}}) > v^*(\mathbf{x}_{j_{i'}^{(k)*}})$. This means that the optimal $\mathbf{j}^{(k)*}$ can be easily obtained from the Bayes-optimal scoring function v^* by ordering the query-document pairs $\mathbf{x}_i^{(k)}$ according to $v^*(\mathbf{x}_i^{(k)})$. This result justifies the *pointwise* approach that estimates v^* in a *regression* setup, since having a regressor function that approximates v^* well, one can readily obtain the Bayes optimal permutation. In this paper, we will also use the pointwise approach in a *discrete density estimation* setup: our goal is to estimate $p^*(\ell | \mathbf{x}_j^{(k)})$ by $p^{\mathcal{A}}(\ell | \mathbf{x}_j^{(k)})$, where the label \mathcal{A} will refer to the method that generates the probability estimates. For the scoring function generated by $p^{\mathcal{A}}$, we will use the notation

$$v^{\mathcal{A}}(\mathbf{x}_i^{(k)}) = \sum_{\ell=1}^K z_{\ell} p^{\mathcal{A}}(\ell | \mathbf{x}_i^{(k)}). \quad (4)$$

In the pointwise approach, the scoring function v induces the permutation \mathbf{j}^v for which

$$v(\mathbf{x}_{j_1^v}^{(k)}) \geq \dots \geq v(\mathbf{x}_{j_{n^{(k)}}^v}^{(k)}). \quad (5)$$

In Sect. 4 we will show that the excess EDCG with respect to the optimal EDCG can be upper bounded by the L_q distance between $p^{\mathcal{A}}(\ell | \mathbf{x}_j^{(k)})$ and $p^*(\ell | \mathbf{x}_j^{(k)})$, motivating the density-estimation-based calibration.

4 Bounds for the excess of EDCG

The main goal of calibrating a multi-class classifier is to get more accurate class conditional probability estimates. Even if a classifier has a good classification performance, its probability estimates can be very poor, as pointed out in Mease et al. (2007). To motivate our approach whose backbone is calibrated multi-class classification, we will show that if the class conditional probability distribution is estimated well in the sense that the Kullback-Leibler divergence between the original and the estimated distribution is small, then we can obtain a close-to-optimal expected DCG in our probabilistic setup.

The results are similar in spirit to the bounds derived in Cossock and Zhang (2008), where the excess of the DCG is bounded in terms of the L_p error of a *regressor*. Our bounds motivate the multi-class classification setup and the class-probability-based calibration techniques (Sect. 5.2), whereas the results of Cossock and Zhang (2008) motivate the regression setup and the regression-based pointwise calibration approach (Sect. 5.4).

4.1 Excess of EDCG in terms of L_q

In (4), we described a way to obtain a scoring function v based on the estimate of the probability distribution of relevance grades. Based on the estimated scoring function v , ranking the set of query-document pairs² \mathcal{D} is straightforward by using (5).

The following proposition gives an upper bound for the difference between the expected DCG values of the Bayes optimal scoring function and its estimate in terms of the quality of the relevance probability estimate.

Proposition 1 *Let $p, q \in [1, \infty]$ and $1/p + 1/q = 1$. Then*

$$\begin{aligned} \text{EDCG}(\mathbf{j}^*, \mathcal{D}) - \text{EDCG}(\mathbf{j}^v, \mathcal{D}) \\ \leq \left(\sum_{i=1}^n \sum_{\ell=1}^K |c_{\tilde{j}_i^v} - c_{\tilde{j}_i^*}| z_{\ell} \right)^{\frac{1}{p}} \left(\sum_{i=1}^n \sum_{\ell=1}^K |p(\ell|\mathbf{x}_i) - p^*(\ell|\mathbf{x}_i)|^q \right)^{\frac{1}{q}}, \end{aligned} \quad (6)$$

where \tilde{j}_i^v and \tilde{j}_i^* are the inverse permutations of j_i^v and j_i^* . The relation between $p(\ell|\mathbf{x})$ and $v(\mathbf{x})$ is defined in (4).

Proof Following the lines of Theorem 2 stated in Cossock and Zhang (2008),

$$\begin{aligned} \text{EDCG}(\mathbf{j}^v, \mathcal{D}) &= \sum_{i=1}^n c_i v^*(\mathbf{x}_{j_i^v}) \\ &= \sum_{i=1}^n c_i v(\mathbf{x}_{j_i^v}) + \sum_{i=1}^n c_i (v^*(\mathbf{x}_{j_i^v}) - v(\mathbf{x}_{j_i^v})) \\ &\geq \sum_{i=1}^n c_i v(\mathbf{x}_{j_i^*}) + \sum_{i=1}^n c_i (v^*(\mathbf{x}_{j_i^v}) - v(\mathbf{x}_{j_i^v})) \\ &= \sum_{i=1}^n c_i v^*(\mathbf{x}_{j_i^*}) + \sum_{i=1}^n c_i (v^*(\mathbf{x}_{j_i^v}) - v(\mathbf{x}_{j_i^v})) + \sum_{i=1}^n c_i (v(\mathbf{x}_{j_i^*}) - v^*(\mathbf{x}_{j_i^*})) \\ &= \text{EDCG}(\mathbf{j}^*, \mathcal{D}) + \sum_{i=1}^n c_i (v^*(\mathbf{x}_{j_i^v}) - v(\mathbf{x}_{j_i^v})) + \sum_{i=1}^n c_i (v(\mathbf{x}_{j_i^*}) - v^*(\mathbf{x}_{j_i^*})). \end{aligned} \quad (7)$$

In (7), $\sum_{i=1}^n c_i v(\mathbf{x}_{j_i^v}) \geq \sum_{i=1}^n c_i v(\mathbf{x}_{j_i^*})$, because \mathbf{j}^v is an optimal permutation for the scoring function v , and so reordering the indices decreases the DCG value. Then, for the permutations j_i^v and j_i^* and for their respective inverse permutations \tilde{j}_i^v and \tilde{j}_i^* , we have

$$\begin{aligned} \text{EDCG}(\mathbf{j}^*, \mathcal{D}) - \text{EDCG}(\mathbf{j}^v, \mathcal{D}) \\ \leq \sum_{i=1}^n (c_{\tilde{j}_i^v} - c_{\tilde{j}_i^*}) (v(\mathbf{x}_i) - v^*(\mathbf{x}_i)) \end{aligned}$$

²In this section we will omit the indexing over the queries.

$$\begin{aligned}
&= \sum_{i=1}^n \sum_{\ell=1}^K (c_{j_i^v} - c_{j_i^*}) z_{\ell} (p(\ell|\mathbf{x}_i) - p^*(\ell|\mathbf{x}_i)) \\
&\leq \sum_{i=1}^n \sum_{\ell=1}^K |(c_{j_i^v} - c_{j_i^*}) z_{\ell}| (p(\ell|\mathbf{x}_i) - p^*(\ell|\mathbf{x}_i)) \\
&\leq \left(\sum_{i=1}^n \sum_{\ell=1}^K |(c_{j_i^v} - c_{j_i^*}) z_{\ell}|^p \right)^{\frac{1}{p}} \left(\sum_{i=1}^n \sum_{\ell=1}^K |p(\ell|\mathbf{x}_i) - p^*(\ell|\mathbf{x}_i)|^q \right)^{\frac{1}{q}}, \quad (8)
\end{aligned}$$

where (8) follows from Hölder's inequality. \square

Corollary 2

$$\text{EDCG}(\mathbf{j}^*, \mathcal{D}) - \text{EDCG}(\mathbf{j}^v, \mathcal{D}) \leq C_1 \cdot \left(\sum_{i=1}^n \sum_{\ell=1}^K |p(\ell_i|\mathbf{x}_i) - p^*(\ell_i|\mathbf{x}_i)|^q \right)^{\frac{1}{q}},$$

where

$$C_1 = \max_{\mathbf{j}, \mathbf{j}'} \left(\sum_{i=1}^n \sum_{\ell=1}^K |(c_{j_i} - c_{j'_i}) z_{\ell}|^p \right)^{\frac{1}{p}}, \quad (9)$$

and the maximum is taken over arbitrary permutations \mathbf{j} and \mathbf{j}' over $1, \dots, n$.

Corollary 2 shows that as the distance between the “exact” and the estimated conditional distributions over the relevance labels tends to 0, the difference in the expected DCG values also tends to 0.

4.2 Bounds for excess of the expected DCG in terms of relative entropy

So far, we have shown that if the estimated class probabilities are close to the conditional discrete probability distribution over the label indices in terms of an L_q -norm, then the Bayes-scoring function will be estimated well. In particular, we have shown that the L_q -norm gives an upper bound on the difference of the EDCG values of the best ranking and the predicted ranking. We will now show that the relative entropy also gives an upper bound on the loss of the EDCG value for predicted rankings. That is, if the entropy of the estimated conditional distribution function is small relative to the class conditional probabilities, then a close-to-optimal ranking is obtained. This finding motivates some of our particular calibration techniques that are related to entropy minimization (Sect. 5).

To simplify the notation, in this section we will denote the class probability vectors by $\mathbf{p}_i = (p_{i,1}, \dots, p_{i,K})$ and $\mathbf{p}_i^* = (p_{i,1}^*, \dots, p_{i,K}^*)$, where $p_{i,\ell} = p(\ell|\mathbf{x}_i)$ and $p_{i,\ell}^* = p^*(\ell|\mathbf{x}_i)$.

Proposition 3 Assume that all elements of \mathbf{p}^* and \mathbf{p}_i are positive. For all $0 < \epsilon \leq \frac{1}{2}$ there exists a $\delta > 0$ such that if $\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2 < \delta$ then

$$\text{EDCG}(\mathbf{j}^*, \mathcal{D}) - \text{EDCG}(\mathbf{j}^v, \mathcal{D}) \leq C_2 \sum_{i=1}^n D_{\text{KL}}(\mathbf{p}_i^* \|\mathbf{p}_i), \quad (10)$$

where $D_{\text{KL}}(\cdot\|\cdot)$ is the Kullback-Leibler divergence between multinomial distributions of one trial with parameters $\mathbf{p}_i = (p(1|\mathbf{x}_i), \dots, p(K|\mathbf{x}_i))$ and $\mathbf{p}_i^* = (p^*(1|\mathbf{x}_i), \dots, p^*(K|\mathbf{x}_i))$ and the constant C_2 is

$$C_2 = C_1 \max_{1 \leq i \leq n} \frac{1 + 2\epsilon}{\frac{1}{2} \min_{1 \leq \ell \leq K} \{1/p_{i,\ell}\}},$$

where C_1 is defined in (9).

Proof First note that the relevance labels come from a multinomial distribution with parameter \mathbf{p}_i^* in our setup. We know (e.g., Gruenwald 2007, pp. 120) that

$$\begin{aligned} D_{\text{KL}}(\mathbf{p}_i^* \|\mathbf{p}_i) &= \frac{1}{2} (\mathbf{p}_i - \mathbf{p}_i^*) \mathcal{I}(f(\zeta_1, \dots, \zeta_K; 1, \mathbf{p}_i)) (\mathbf{p}_i - \mathbf{p}_i^*)^\top + o(\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2), \end{aligned} \quad (11)$$

where $\mathcal{I}(f(\zeta_1, \dots, \zeta_K; 1, \mathbf{p}_i))$ is the Fisher Information Matrix (FIM) of the multinomial probability distribution $f(\zeta_1, \dots, \zeta_K; 1, \mathbf{p}_i)$ assuming one trial. In our special case, the FIM is diagonal with elements $1/p_{i,\ell}$ in the diagonal, thus we have

$$\frac{1}{2} (\mathbf{p}_i - \mathbf{p}_i^*) \mathcal{I}(f(\zeta_1, \dots, \zeta_K; 1, \mathbf{p}_i)) (\mathbf{p}_i - \mathbf{p}_i^*)^\top \leq \frac{1}{2} \max_{\ell} \{1/p_{i,\ell}\} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2$$

and

$$\frac{1}{2} (\mathbf{p}_i - \mathbf{p}_i^*) \mathcal{I}(f(\zeta_1, \dots, \zeta_K; 1, \mathbf{p}_i)) (\mathbf{p}_i - \mathbf{p}_i^*)^\top \geq \frac{1}{2} \min_{\ell} \{1/p_{i,\ell}\} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2.$$

For all $\epsilon > 0$ there exists a $\delta > 0$, such that if $\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2 < \delta$, then $\left| \frac{o(\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2)}{\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2} \right| < \epsilon \cdot \frac{1}{2} \min_{\ell} \{1/p_{i,\ell}\}$. Now, using this constant and the lower bound of the FIM matrix, we can rewrite (11) as

$$(1 - \epsilon) \frac{1}{2} \min_{\ell} \{1/p_{i,\ell}\} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2 \leq D_{\text{KL}}(\mathbf{p}_i^* \|\mathbf{p}_i).$$

Using the fact that $1/(1 - \epsilon) \leq 1 + 2\epsilon$ for $0 < \epsilon \leq \frac{1}{2}$, we have

$$\frac{1}{2} \min_{\ell} \{1/p_{i,\ell}\} \|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2 \leq (1 + 2\epsilon) D_{\text{KL}}(\mathbf{p}_i^* \|\mathbf{p}_i).$$

It then follows that

$$\|\mathbf{p}_i - \mathbf{p}_i^*\|_2^2 \leq \frac{1 + 2\epsilon}{\frac{1}{2} \min_{\ell} \{1/p_{i,\ell}\}} D_{\text{KL}}(\mathbf{p}_i^* \|\mathbf{p}_i).$$

We obtain the inequality of the proposition using Corollary 2 with $q = 2$. \square

The relevance labels are typically represented as integers and so they constitute an *ordered set*. It is arguably a promising approach to exploit this ordering when learning a ranking function. This is possible both in a classification setup (via the optimization of a customized loss that differentiates errors proportionally to the severity of the misclassification) and naturally in an ordinal regression setting (Chu and Keerthi 2005; Aioli and Sperduti 2010).

Our theoretical results justify using a classification based method which produces good posterior probability estimates in terms of the KL divergence or L^p norm. At the same time, a similar result also exists for the regression based setup where the excess of DCG is bounded in terms of the L^p error of the regressor (Cossock and Zhang 2008), and efficient regression-based ranking algorithms (Wu et al. 2010) also exist.

Intuitively, the main difference between the methods based on classification and regression is that in the latter case, the loss function is *strictly monotonic*.³ ADABOOST.MH provides a convenient way to use a strictly monotone loss function by applying an appropriate initial weighting that encodes the loss function itself. We do not explore this promising research direction in this paper, however, the instance weighting used here Appendix A also emphasizes accuracy on highly relevant classes and therefore has a somewhat similar effect. Another possibility, originally used in the MCRANK algorithm, is to convert the K -class ordinal regression problem into $K - 1$ binary classification problem with the goal of obtaining posterior probabilities for $p(\ell_i > \ell)$ for $\ell = 1, \dots, K - 1$. We found that our multi-class setup worked sufficiently well for obtaining state-of-the-art experimental results, nevertheless, trying the approach and working out the corresponding calibration methodology is definitely an interesting avenue to explore, especially if the goal is to further diversify the final ranking ensemble.

5 Calibration

We shall assume throughout the paper that a multi-class classification algorithm provides vector-valued multi-class discriminant functions of the form $\mathbf{f}: \mathcal{X} \rightarrow \mathbb{R}^K$, where \mathcal{X} is the input space (in our case the space of query-document pairs represented by a real-valued vector) and K is the number of classes (relevance levels). Elements of these vector-valued discriminant functions will be denoted by $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_K(\mathbf{x}))$. The goal of multi-class classification is to identify the correct class (or classes in the case of multi-label classification). For completeness, we will provide details on training multi-class ADABOOST.MH and multi-class SVM (see Appendix A and Appendix B, respectively).

In general, multi-class large-margin classification algorithms force discrimination by pulling the scores $f_\ell(\mathbf{x})$ away from zero. This means that direct (linear) conversion into class probabilities usually does not produce good estimates (Mease et al. 2007). This phenomenon is particularly pronounced in the case of ADABOOST due to the exponential loss which increases sharply with negative margins (Niculescu-Mizil and Caruana 2005). At the same time, the score vector usually represents the *order* of the probabilities rather well, so a simple nonlinear function can transform the scores into good probability estimates. The process of learning this nonlinear function from held-out data is called *calibration* (Platt 2000). In this section, we will describe several calibration techniques, some of them inspired by classical techniques tuned for squared error and cross-entropy (Niculescu-Mizil and Caruana 2005; Wu et al. 2004), and some of them motivated directly by the NDCG measure.

³Ordinal regression postulates *monotonicity* of the loss \mathcal{L} , that is, $\ell_1 \geq \ell_2 \geq \ell(\mathbf{x})$ implies $\mathcal{L}(\ell(\mathbf{x}), \ell_1) \geq \mathcal{L}(\ell(\mathbf{x}), \ell_2)$, without committing itself to any numerical assumption (such as the quadratic loss) on \mathcal{L} .

5.1 Obtaining posterior probabilities: the naive estimator

In classical multi-class classification the elements of $\mathbf{f}(\mathbf{x})$ are treated as posterior scores corresponding to the labels, and the predicted label is

$$\hat{\ell}(\mathbf{x}) = \arg \max_{\ell=1, \dots, K} f_{\ell}(\mathbf{x}),$$

where $f_{\ell}(\mathbf{x})$ is the ℓ th element of $\mathbf{f}(\mathbf{x})$. The scores $f_{\ell}(\mathbf{x})$ are usually not all positive and they do not sum to 1. Hence, if we need to estimate the posterior class probabilities $p(\ell|\mathbf{x})$, we have to transform and normalize the scores. The naive calibration of a classifier \mathbf{f} consists of a linear rescaling followed by a normalization. First, we make the scores positive by applying the transformation

$$\mathbf{f}'(\mathbf{x}) = 1 + \frac{\mathbf{f}(\mathbf{x})}{R},$$

where

$$R = \max_{\mathbf{x} \in \mathcal{X}, 1 \leq \ell \leq K} |f_{\ell}(\mathbf{x})|. \quad (12)$$

Then we normalize the shifted score to obtain

$$p^{\text{SHIFT}}(\ell|\mathbf{x}_i) = \frac{f'_{\ell}(\mathbf{x}_i)}{\sum_{\ell'=1}^K f'_{\ell'}(\mathbf{x}_i)}.$$

In the case of ADABOOST.MH, the classifier has the form

$$\mathbf{f}(\mathbf{x}) = \sum_{t=1}^T \alpha^{(t)} \mathbf{v}^{(t)} \varphi^{(t)}(\mathbf{x}),$$

where both the elements of the vote vector $\mathbf{v}^{(t)}$ and the scalar classifier $\varphi^{(t)}(\mathbf{x})$ are ± 1 -valued, so R can be replaced by $\sum_{t=1}^T \alpha^{(t)}$. In the case of multi-class SVMs there is no such “natural” upper bound, so we fall back on the explicit maximization (12).

5.2 Class-probability-based calibration (CPC) using a sigmoidal function

The common solution (Platt 2000) to transform the scores to probability estimates is to apply a *sigmoidal* function

$$s_{\theta}(f) = \frac{1}{1 + \exp(-a(f - b))},$$

where the parameters $\theta = (a, b)$ are to be tuned. The probability estimates are then of the form

$$p^{s_{\theta}}(\ell|\mathbf{x}) = \frac{s_{\theta}(f_{\ell}(\mathbf{x}))}{\sum_{\ell'=1}^K s_{\theta}(f_{\ell'}(\mathbf{x}))}. \quad (13)$$

The parameters of the sigmoid function can be tuned by minimizing a so-called *target calibration function* (TCP) $L^{\mathcal{A}}(\theta, \mathbf{f})$, where θ is the set of parameters to be tuned, \mathbf{f} is the score vector, and the upper index \mathcal{A} refers to the type of the particular TCF. $L^{\mathcal{A}}(\theta, \mathbf{f})$ is naturally a function of the validation data set, too, (which is not necessarily the same as the training set), but here we will omit this dependence to simplify the notation.

Given a TCF $L^{\mathcal{A}}$ and a multi-class classifier \mathbf{f} , our goal is to find the optimal calibration parameters

$$\theta^{\mathcal{A}, \mathbf{f}} = \arg \min_{\theta} L^{\mathcal{A}}(\theta, \mathbf{f}).$$

The output of this calibration step is a probability distribution $p^{\mathcal{A}, \mathbf{f}}(\ell|\mathbf{x})$ and a corresponding Bayes-scoring function $v^{\mathcal{A}, \mathbf{f}}(\mathbf{x})$ defined in (4). From now on, we will refer to this scheme as a *class-probability-based calibration* (CPC).

When several calibration functions \mathcal{A} are used on several scores \mathbf{f} (generated by, for example, multi-class classifiers with different hyperparameters), the result is an *ensemble* of probability distributions $p^{\mathcal{A}, \mathbf{f}}(\ell|\mathbf{x})$ indexed by \mathcal{A} and \mathbf{f} . To obtain a single combined score, we mix the ensemble using a linear combination

$$p^{\text{CPC}}(\ell|\mathbf{x}) = \sum_{\mathcal{A}, \mathbf{f}} \pi(\mathcal{A}, \mathbf{f}) p^{\mathcal{A}, \mathbf{f}}(\ell|\mathbf{x}),$$

where $\pi(\mathcal{A}, \mathbf{f})$ is an appropriately chosen weight. Then we obtain a combined Bayes-scoring function by noticing that

$$\begin{aligned} v^{\text{CPC}}(\mathbf{x}) &= \sum_{\ell=1}^K z_{\ell} p^{\text{CPC}}(\ell|\mathbf{x}) \\ &= \sum_{\ell=1}^K z_{\ell} \sum_{\mathcal{A}, \mathbf{f}} \pi(\mathcal{A}, \mathbf{f}) p^{\mathcal{A}, \mathbf{f}}(\ell|\mathbf{x}) \\ &= \sum_{\mathcal{A}, \mathbf{f}} \pi(\mathcal{A}, \mathbf{f}) \sum_{\ell=1}^K z_{\ell} p^{\mathcal{A}, \mathbf{f}}(\ell|\mathbf{x}) \\ &= \sum_{\mathcal{A}, \mathbf{f}} \pi(\mathcal{A}, \mathbf{f}) v^{\mathcal{A}, \mathbf{f}}(\mathbf{x}). \end{aligned} \quad (14)$$

The proper selection of $\pi(\cdot, \cdot)$ can further increase the quality of the estimation. In Sect. 6, we will describe a simple setup borrowed from the theory of experts.

5.3 Target calibration functions

In the simplest case, the TCF can be

$$L^{\text{LS}}(\theta, \mathbf{f}) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} -\log p^{\theta}(\ell_i^{(k)} | \mathbf{x}_i^{(k)}). \quad (15)$$

We refer to this function as the *log-sigmoid* TCF. The motivation for using the log-sigmoid TCF is that the resulting probability distribution minimizes the relative entropy between the Bayes optimal probability distribution p^* and $p^{\text{LS}, \mathbf{f}}$. According to Proposition 3, a small relative entropy implies that the expected DCG score of the resulting ranking is close to the minimum expected DCG score.

In practice, distributions being less (or more) uniform over the labels might work better. This degree of freedom can be expressed by introducing the entropy weighted version of the

log-sigmoid TCF

$$L_C^{\text{EWLS}}(\theta) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} -\log p^{s_\theta}(\ell_i^{(k)} | \mathbf{x}_i^{(k)}) \times H_M(p^{s_\theta}(\ell_1 | \mathbf{x}_i^{(k)}), \dots, p^{s_\theta}(\ell_K | \mathbf{x}_i^{(k)}))^C, \quad (16)$$

where

$$H_M(p_1, \dots, p_K) = -\sum_{\ell=1}^K p_\ell \log p_\ell,$$

and C is a hyperparameter.

We also use TCFs based on the *expected loss*

$$L^{\text{EL}}(\theta) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} \sum_{\ell=1}^K \mathcal{L}(\ell, \ell_i^{(k)}) p^{s_\theta}(\ell | \mathbf{x}_i^{(k)}), \quad (17)$$

where $\mathcal{L}(\ell, \ell_i^{(k)})$ represents the loss if ℓ is predicted instead of the correct label $\ell_i^{(k)}$. We use the standard squared loss $\mathcal{L}(\ell, \ell') = (\ell - \ell')^2$.

If the label indices have some structure (that is, they are ordinal as in our case), it is also possible to first compute the expected label

$$\bar{\ell}_i^{(k)} = \sum_{\ell=1}^K \ell p^{s_\theta}(\ell | \mathbf{x}_i^{(k)})$$

and then compute the *expected label loss* TCF

$$L^{\text{ELL}}(\theta) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} \mathcal{L}(\bar{\ell}_i^{(k)}, \ell_i^{(k)}). \quad (18)$$

Note that the definition of $\mathcal{L}(\cdot, \cdot)$ might need to be redefined for L^{ELL} since the weighted average of the label indices might not be a label index at all.

Finally, we can apply the idea of SMOOTHGRAD (Chapelle and Wu 2010) to obtain a TCF. In SMOOTHGRAD a smooth surrogate function is used to optimize the NDCG metric. In particular, using the normalized *soft indicator* (or similarity function)

$$h_{\theta, \sigma}(\mathbf{x}_i^{(k)}, \mathbf{x}_{i'}^{(k)}) = \frac{\exp(-\frac{1}{\sigma}(v^{s_\theta}(\mathbf{x}_i^{(k)}) - v^{s_\theta}(\mathbf{x}_{i'}^{(k)}))^2)}{\sum_{i''=1}^{n^{(k)}} \exp(-\frac{1}{\sigma}(v^{s_\theta}(\mathbf{x}_i^{(k)}) - v^{s_\theta}(\mathbf{x}_{i''}^{(k)}))^2)},$$

the *soft NDCG* TCF can be written as

$$L_\sigma^{\text{SNDG}}(\theta) = -\sum_{k=1}^M \sum_{i=1}^{n^{(k)}} \sum_{i'=1}^{n^{(k)}} z_i^{(k)} c_{i'} h_{\theta, \sigma}(\mathbf{x}_i^{(k)}, \mathbf{x}_{j_{i'}}^{(k)}), \quad (19)$$

where $\mathbf{j} = (j_1, \dots, j_{n^{(k)}})$ is the permutation defined by the scoring function $v^{s_\theta}(\cdot)$ (5). The parameter σ controls the smoothness of L_σ^{SNDG} ; that is, the higher the value of σ , the smoother the function will be, but also the bigger the difference will be between the NDCG value and the value of the surrogate function. If $\sigma \rightarrow 0$ then L_σ^{SNDG} tends to the NDCG value; but, at the same time, optimizing the surrogate function becomes harder.

5.4 Regression-based pointwise calibration (RBC)

The CPC calibration can be naturally replaced by a regression technique in which the relevance grades are predicted explicitly instead of estimating the discrete conditional probability distribution $p^*(\ell|\mathbf{x})$. We will denote a regression function by $g_\theta : \mathbb{R}^K \rightarrow \mathbb{R}$, where $\theta \in \mathbb{R}^p$ is the vector that parametrizes the regression function. A pointwise estimate of the relevance grade can be obtained using

$$\widehat{z}(\mathbf{x}) = g_\theta(\mathbf{f}(\mathbf{x})), \quad (20)$$

where $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^K$ is the score vector output by the multi-class classifier. Optimizing the NDCG score with respect to the regression parameters θ is computationally difficult. The common solution to this problem is to use a surrogate function. Finding good surrogate functions for hard-to-optimize IR metrics is an open research problem (Ravikumar et al. 2011; Chapelle and Wu 2010; Cossock and Zhang 2008). The simplest choice for the surrogate function is the square loss for which the regression model can be fitted in a standard L_2 setup by minimizing

$$L_2(\theta, \mathbf{f}) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} (z_i^{(k)} - g_\theta(\mathbf{f}(\mathbf{x}_i^{(k)})))^2.$$

The problem with this choice is that the square loss is not NDCG consistent (Ravikumar et al. 2011). In spite of this, it turns out that if the relevance grades are rescaled querywise by the DCG scores, that is,

$$L'_2(\theta, \mathbf{f}) = \sum_{k=1}^M \sum_{i=1}^{n^{(k)}} \left(\frac{z_i^{(k)}}{\text{DCG}(\mathbf{j}^{(k)*}, \mathcal{D}^{(k)})} - g_\theta(\mathbf{f}(\mathbf{x}_i^{(k)})) \right)^2, \quad (21)$$

then the objective function becomes NDCG-consistent. Since $\text{DCG}(\mathbf{j}^{(k)*}, \mathcal{D}^{(k)})$ is unknown for a new query, it is impossible to predict the relevance grades $z_i^{(k)}$ on an absolute scale. Nevertheless, the ordering of the documents within a query is not changed by this constant scaling, so the predictions $g_\theta(\mathbf{f}(\mathbf{x}_i^{(k)}))$ can be used for ranking scores without knowing $\text{DCG}(\mathbf{j}^{(k)*}, \mathcal{D}^{(k)})$.

In our experiments we applied four different regression methods; namely, logistic regression, linear regression, neural network regression, and polynomial regression of degree between 2 and 4, inclusive.

6 Ensemble of the calibrated models

Selecting the best hyperparameters for a multi-class learning algorithm and the best calibration function is normally done in a validation step. This procedure would “throw away” most of the diverse information represented by the different predictors. Instead of selecting the best, we use all the relevance predictions $v^{\mathcal{A}, \mathbf{f}}(\mathbf{x}, S)$ obtained by different multi-class classifiers \mathbf{f} using different TCFs \mathcal{A} . Each relevance prediction can be used as a scoring function to rank the set of documents $\mathbf{x}_i^{(k)}$ according to (5). Up until now, our method is an almost⁴ purely pointwise approach. To fine-tune the algorithm and to make use of the

⁴The SNDCG calibration (19) is listwise.

diversity of our models, we combine the scoring functions $v^{\mathcal{A},\mathbf{f}}(\mathbf{x}, S)$ using an exponentially weighted forecaster (Cesa-Bianchi and Lugosi 2006).

The weights of the models are tuned on the NDCG score, giving a slight listwise touch to the final step of our approach. Formally, the final scoring function is obtained by using the weights

$$\pi(\mathcal{A}, \mathbf{f}) = \exp(c\omega^{\mathcal{A},\mathbf{f}}),$$

where $\omega^{\mathcal{A},\mathbf{f}}$ is the NDCG₁₀ score of the ranking obtained by using $v^{\mathcal{A},\mathbf{f}}(\mathbf{x})$. Plugging it into (14), the combined scoring function is

$$v^{\text{ENSEMBLE}}(\mathbf{x}) = \sum_{\mathcal{A},\mathbf{f}} \exp(c\omega^{\mathcal{A},\mathbf{f}}) v^{\mathcal{A},\mathbf{f}}(\mathbf{x}). \quad (22)$$

The parameter c is also tuned on the held-out validation set. It controls the dependence of the weights on the NDCG₁₀ values. A large c means that we focus only on the good models, whereas a c close to zero represents a near-uniform weighting.

Our rationale for using this particular mixing scheme is as follows. First, it is simple and computationally efficient to tune, which is important when we have a large number of models. Our basic setup is to train the models *in parallel* using computationally cheap pointwise objectives, and combine these fixed models linearly to improve a computationally expensive listwise objective. Complex dynamic weighting schemes where model combination and model training are intertwined (such as boosting) are not suitable in this setup. Among linear schemes, we were looking for a technique that subsumes classical winner-takes-all validation (choosing the best model) and simple equal-weight voting, and which could be tuned with a single hyperparameter between these two extremes. Exponential weighting is arguably the simplest of such schemes. Simplicity is also crucial to prevent model-mixing from overfitting.

Our choice was also inspired by theoretical guarantees over the cumulative regret of a mixture of experts on individual (model-less) sequences (Cesa-Bianchi and Lugosi 2006), without, of course, claiming that these theoretical results apply directly to our setup. That said, we do not assert that it is the best possible model-mixing scheme. It may be that more sophisticated techniques would allow us to further improve the results. Our point is that model-mixing is important, and even a simple method can be significantly better than the classical winner-takes-all validation. This result seems to agree with the general consensus drawn from the experiences on recent large-scale learning challenges (Bennett and Lanning 2007; Dror et al. 2009; Chapelle et al. 2011).

6.1 Guidelines for building the models

According to the results of our experiments, there does not seem to be any *statistical* cost of including as many models as possible; that is, we know no instance when deleting models before mixing improved the results, mainly since bad models were discarded anyway by the weighting scheme. At the same time, increasing the number of models without limit does not make much sense, but the problem is computational rather than statistical.

Within our computational limits, our objective was to have the largest number of *diverse* models possible. Most of the computational time was spent on training the boosting models, so the first step was to “cover” the hyperparameter space (number of tree leaves, number of product terms, number of iterations, the regularization coefficient in SVM) quasi-uniformly in a region derived from our previous experiences and from some preliminary experiments.

Compared to training the models, calibration took almost no time, which explains why we added as many and as diverse calibration techniques as possible. In Busa-Fekete et al. (2011a) we conducted an empirical analysis to discover what the main source of diversity was and we tried some other techniques to increase diversity that we did not apply in this paper for simplifying the method. All we could conclude was that any reasonable “perturbation” of the problem or of the models helps to improve the overall result.

7 Experiments

In this section, we will present our experimental results. In Sect. 7.1, we will briefly discuss some of the issues related to the available evaluation tools. Section 7.2 describes the benchmark data sets used in the experiments. In Sect. 7.3, we summarize the state-of-the-art techniques we compare with our approach and with each other. To assure full reproducibility of our experiments, we also provide details of the experimental setup here. Section 7.4 contains the results of the comparative experiments. We also discuss the general conclusions that can be drawn from the results of our experiments. In Sect. 7.5, we investigate how the performance of the algorithm depends on the size of the data and on the quality of the training relevance grades. In Sect. 7.6, we assess the effect of query-wise normalization of relevance grades proposed by Ravikumar et al. (2011). In Sect. 7.7, we examine the diversity of the models that we mix in the ensemble step in a qualitative manner. Lastly, we will summarize and discuss the official results in the Yahoo! Learning-to-Rank Challenge (Chapelle et al. 2011) in Sect. 7.8.

7.1 Evaluation tools

Here, we will briefly describe and compare the various tools available for computing NDCG scores. The definition (2) is unambiguous; nevertheless, the tools can differ in the definition of the discount factor c_i (3). More importantly, there may be important differences in the way the DCG score is normalized either when there exist no relevant documents for a query ($z_i = 0$ for all i), or when the number of documents is less than the truncation level n_{\max} . Even though this seems to be a technical subtlety, it turns out that the confusion arising from using the different tools can significantly alter the numerical scores and in some case may even change the relative ordering of the algorithms on the data sets.

We compared six evaluation tools to compute the NDCG scores:

1. The LETOR 3.0 script implemented in Perl⁵
2. The LETOR 4.0 script implemented in Perl⁶
3. The MS script implemented in Perl⁷
4. The YAHOO script implemented in Python⁸
5. The RANKLIB package implemented in Java⁹
6. The TREC evaluation tool v8.1 implemented in C¹⁰

⁵<http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR3.0/EvaluationTool.zip>.

⁶<http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR4.0/Evaluation/Eval-Score-4.0.pl.txt>.

⁷<http://research.microsoft.com/en-us/projects/mslr/eval-score-mslr.pl.txt>.

⁸https://github.com/busarobi/YAHOO_EVALTOOL/blob/master/evaluate.py.txt.

⁹<http://www.cs.umass.edu/~vdang/ranklib.html>.

¹⁰http://trec.nist.gov/trec_eval/.

The evaluation tools can be divided into three groups. The tools of the first group compute $DCG_{n_{\max}}$ according to the definition (2) described in Sect. 3. The LETOR 3.0, RANKLIB, and TREC tools belong to this group. All of these tools assign a zero score to a query if it is *empty*; that is, $z_i = 0$ for all i , which means that there are no relevant documents. The TREC tool makes use of the labels of documents given in the input file as relevance grades by default. From this point of view, this is the most flexible implementation, since arbitrary relevance grades can be defined. For example, in the case of the MQ2008 dataset, the labels 0, 1, and 2 should be simply replaced by 0, 1, and 3, respectively, to have the commonly used exponential grades as given in (1).

The second group is comprised of the YAHOO tool alone. It also computes the $DCG_{n_{\max}}$ according to the definition (2), but it assigns 1.0 to the empty queries. This is a minor difference that generates an additive bias between the $NDCG_{n_{\max}}$ computed by YAHOO tool and the three tools of the first group.

The third group consists of the LETOR 4.0 and MS tools. Except for a small technical difference (the LETOR 4.0 tool can be applied for up to three relevance labels, whereas the MS tool can handle up to five relevance labels), they compute the same score. Like the RANKLIB and LETOR 3.0 tools, they assign a zero to a query where the ideal $DCG_{n_{\max}}$ is zero. Their rather strange feature is that they also assign a zero $DCG_{n_{\max}}$ score to a query with less than n_{\max} documents in it, even if these documents are highly relevant. So, formally, they compute the $DCG_{n_{\max}}$ score as

$$DCG_{n_{\max}}(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}) = \begin{cases} \sum_{i=1}^{n_{\max}} c_i z_{j_i}^{(k)} & \text{if } n_{\max} \leq n^{(k)} \\ 0 & \text{otherwise.} \end{cases} \quad (23)$$

This truncation not only distorts the *test* score, but it can also alter the *training* of such algorithms that depend directly on the NDCG score. Indeed, for example, in ADARANK (Xu and Li 2007), which optimizes the $NDCG_{10}$ evaluation metric, a query containing fewer than 10 documents does not influence the computation of the coefficient of the weak ranker at all, and the weight of such queries converge to zero over the successive boosting iterations.

To illustrate the effect of these differences, we compared the $NDCG_{n_{\max}}$ scores obtained by the YAHOO tool (Figs. 2(a) and 2(c)) with the LETOR 4.0 tool (Figs. 2(b) and 2(d)) on the LETOR 4.0 data sets (MQ₂₀₀₇ and MQ₂₀₀₈) using five state-of-the-art rankers.¹¹ First, note the striking absolute differences, especially for larger n_{\max} for which the effect of the truncation (23) is bigger.¹² Worse, on the MQ₂₀₀₇ data set even the *order* of the methods is altered: the RANKNET method is put at a serious disadvantage by the incorrectly implemented evaluation tool. This latter finding was the main reason why we included this technical section in the paper.

From now on all reported NDCG scores will be computed using the YAHOO tool (with a score of 1.0 for “empty” queries). Note that we also changed the code of RANKLIB based on this tool. Although our main evaluation metric is the $NDCG_{10}$ score, we will also report the Expected Reciprocal Rank (ERR) scores (Chapelle et al. 2009)

$$ERR(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}) = \sum_{i=1}^{n^{(k)}} \frac{1}{i} \frac{z_{j_i}^{(k)}}{Z} \prod_{i'=1}^{i-1} \left(1 - \frac{z_{j_{i'}}^{(k)}}{Z}\right), \quad (24)$$

¹¹ See Sect. 7.2 and Sect. 7.3 for a detailed description of the data sets and the methods.

¹² There are several queries in the data that have only 7 documents, so the $NDCG_{8-10}$ scores will be zero for all of these queries according to (23). This is the reason why the LETOR scores fall rapidly for $n_{\max} \geq 8$ on the MQ₂₀₀₈ data set (Fig. 2(d)).

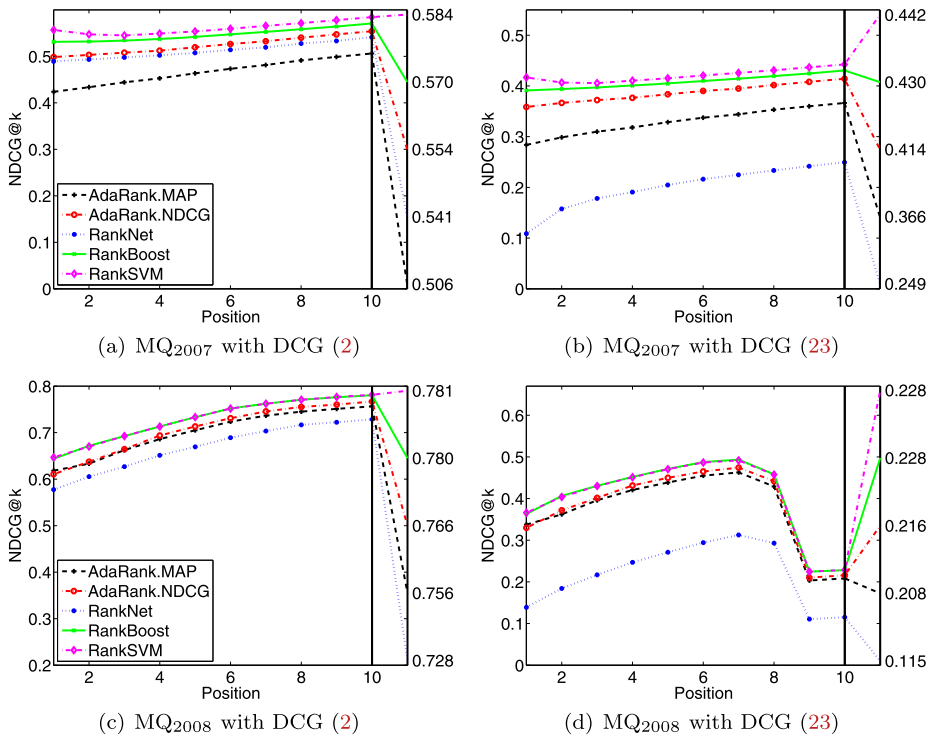


Fig. 2 The empirical NDCG_{n_{max}} scores on the Letor 4.0 test sets. The *left panels* ((a) and (c)) depict the NDCG_{n_{max}} scores computed by the YAHOO tool, and the *right panels* ((b) and (d)) show the results when we use the official LETOR 4.0 tool. Apart from the strikingly large absolute differences, the incorrectly implemented LETOR 4.0 tool also alters the order of the methods

where Z is the maximal relevance grade (that is, $Z = 2^{K-1} - 1$ in our case).

In summary, we propose the following guidelines for future research studies, with the aim of making numerical results comparable across studies carried out.

1. If possible, apply the YAHOO tool.
2. Always specify which tool is used to compute the numerical results.
3. If a new tool for computing the NDCG_{n_{max}} is implemented, always specify the default value for empty queries, and avoid the bug (23) when $n_{\max} > n^{(k)}$.

7.2 Data sets

We evaluated the ranking methods on the commonly used benchmark data sets summarized in Table 1. We were only interested in data sets with more than two relevance levels, firstly because the calibration for binary relevance labels does not make too much sense, and secondly, because the difference between various learning algorithms can be more pronounced in the multi-label case. Note also that the general consensus in the IR community is that graded relevance labels are superior to the binary setup when large document collections are involved (Järvelin and Kekäläinen 2002; Kekäläinen and Järvelin 2002; Sakai 2007).

Table 1 The benchmark data sets used in the experiments

	#documents	#queries	#features	#relevance levels	avg. #docs. per query
LETOR data sets (5-fold cuts)					
OHSUMED	16140	106	45	3	152
MQ ₂₀₀₇	69623	1692	46	3	41
MQ ₂₀₀₈	15211	784	46	3	19
YAHOO data sets (simple cut)					
YAHOO ₁	709877	29921	700	5	24
YAHOO ₂	172870	6330	700	5	27
MS data sets (5-fold cuts)					
MS ₁	1200192	10000	136	5	120
MS ₂	3771125	31531	136	5	120

The features were normalized querywise in the LETOR data sets (OHSUMED, MQ₂₀₀₇, MQ₂₀₀₈) so we did not preprocess them. In the case of YAHOO and MS data sets, we augmented the feature sets. Besides the original features, we added querywise standardized features, which means that we rescaled the feature values for each query separately so to have zero mean and a standard deviation of one. The idea behind querywise normalization is that features used in a learning-to-rank tasks often represent some kind of count. Some of these quantities are not comparable in an absolute way: for example, the number of times a query term occurs in a document is not comparable between common query terms (e.g., “dog”) and rare query terms (e.g., “AdaBoost”).

We used the official train/valid/test cut for each data set. We divided the official training sets by a random 80–20 % split into training and calibration sets. The latter was used to adjust the parameters of the different calibration methods and to tune the hyperparameter c of the exponential weighting scheme.

7.3 Methods and experimental setup

We compared our algorithms with five state-of-the-art ranking methods and with the ranker that used the simple best feature (described below). Here, we will briefly summarize them.

1. **BESTFEATURE:** As a baseline, we will report the performance of the ranker based on the single best feature. Each feature can be used as a ranker function since values of a given feature determine a ranking on an individual query (for example, in the early years of IR, the BM25 feature alone was used as a ranking score). In the training phase, the ranker chooses the feature that achieves the highest performance in terms of the evaluation metric of interest on the training data. For a test query, the ranker then simply returns the values of this single feature as a score. Since the feature values are given in our experiments, there is no hyperparameter to be validated for this method. We will refer to this simple approach as $\text{BF.}\{\text{ERR}, \text{NDCG}\}$, according to the evaluation metric used.
2. **ADARANK** (Xu and Li 2007) is a listwise boosting approach that seeks to optimize an arbitrary listwise IR metric, such as the Mean Average Precision (MAP), ERR, or NDCG. Inspired by ADABOOST, it uses a stepwise greedy optimization technique to maximize the chosen IR metric. In every boosting iteration, ADARANK re-weights the

- queries based on their scores obtained by the evaluation metric: it up-weights the query that has a lower score and down-weights high-scoring queries. The weak learner is chosen by optimizing the listwise evaluation metric of interest, which is usually hard to optimize except for very simple weak classifiers. This may be viewed as a handicap of this method. According to the original implementation of ADARANK, we used the best feature ranker (BF) described above as the base ranker taking into account the weighting of queries. The only hyperparameter of ADARANK is the number of boosting iterations, which we optimized by using early-stopping on the validation set. We refer to this method as ADARANK. $\{\text{MAP,ERR,NDCG}\}$, depending on which evaluation metric is used.
3. RANKNET (Burges et al. 2005) is a neural-net-based method which employs a loss based on pairwise cross entropy as its objective function. The neural net with one output node is trained to optimize directly the differentiable probabilistic pairwise loss instead of the common squared loss. We validated the number of hidden layers ranging from 1 to 3 and the number of neurons in the hidden layers ranging from 10 to 500. For the number of training epochs we applied early stopping.
 4. RANKBOOST (Freund et al. 2003) is a pairwise boosting approach. The objective function is the rank loss (as opposed to ADABOOST, which optimizes the exponential loss). In each boosting iteration the weak classifier is chosen by maximizing the weighted rank loss. For the weak learner, we used decision stumps and a variant of the single decision stump described in Freund et al. (2003), which is able to optimize the rank loss in an efficient way.
 5. RANKSVM (Joachims 2006) is a pairwise method based on SVM, which formulates the ranking task as a binary classification. We used a linear kernel because the optimization using non-linear kernels cannot be carried out in a reasonable time. The tolerance level of the optimization was set to 0.001 and the regularization parameter was validated in the interval $[10^{-6}, 10^4]$ with a logarithmically increasing step size.
 6. COORDINATEASCENT (CA) (Metzler and Croft 2007) is a linear listwise model, where the scores of the query-document pairs are calculated as weighted combinations of the feature values. The weights are tuned by using a coordinate ascent optimization method, where the objective function is an arbitrary evaluation metric given by the user. The coordinate ascent optimization method itself has two hyperparameters to be tuned: the number of restarts R from random initial weights, and the number of iterations T taken after each restart. We used $R = 30$ and $T = 100$. We did not validate these hyperparameters, but using the validation set we evaluated every model obtained due to restarting the optimization process, and we kept the one that had the highest performance score.

In our approach, we applied two multi-class learning methods whose outputs were then used in a calibration procedure: ADABOOST.MH (Schapire and Singer 1999) and the multi-class Support Vector Machine (MC-SVM) (Crammer and Singer 2001). To train ADABOOST.MH, we used our open source implementation (Benbouzid et al. 2012b) available at <http://multiboost.org>. We employed decision trees with 8 and 64 leaves and decision products with 3 and 10 terms. We calibrated and mixed all the trained boosted models. The training was performed on the EGI grid,¹³ which allowed us to perform the training process in parallel, and thus it took less than one day to get all the strong classifiers. Further details on training ADABOOST.MH can be found in Appendix A.

To train the MC-SVM (Crammer and Singer 2001), we used a linear kernel as the training time for non-linear kernel functions was prohibitively long. The tolerance of the opti-

¹³<http://www.egi.eu>.

mization was set to 0.001. We trained MC-SVM using different trade-off parameters ranging from 10^{-6} to 10. Further details on training MC-SVM can be found in Appendix B.

We only tuned the number of iterations T for ADABOOST.MH and the trade-off parameter C for SVM, and the base parameter c in the exponential weighting scheme (22) on the validation set. In the exponential weighting combination (22) we set the weights using the NDCG_{10} performance scores of the calibrated models, and c and T were selected based on the performance score of the combined scoring function $v^{\text{ENSEMBLE}}(\cdot)$ in terms of NDCG_{10} . The hyperparameter optimization was performed using a simple grid search, where c ranged from 0 (corresponding to a uniform weighting) to 200 and for T from 10 to 10000. On the larger YAHOO and MS data sets, the optimal numbers of boosting iterations were between 8000 and 10000 and about 2000, respectively. Interestingly, on the LETOR data sets the optimal T was much lower: for LETOR 3.0 the best number of iterations is $T = 100$ and for both LETOR 4.0 data sets it is $T = 50$. The best base parameter c is larger than 100 for all data sets. This value is relatively high considering that it is used in the exponent, but the performances of the best models were relatively close to each other so the weight distribution of these good models was not very far from being uniform. We used fixed parameters $C = 2$ in the TCN function L_C^{EWLS} (16), and $\sigma = 0.01$ in L_σ^{SNDG} (19).

To demonstrate the efficiency of our ensemble scheme described in Sect. 6, we will also present the test performance scores of the single best calibrated multi-class classifier. In this case, the scores for a single trained and calibrated model can be obtained using (4). The best ranker is then chosen based on its performance score on the validation set. We will refer to this ranker based on a single calibrated multi-class classifier as $\{\text{AB}, \text{SVM}\} + \text{ONEBEST}$, depending on which multi-class training method was used.

7.4 Comparative results

Tables 2 and 3 list the NDCG and ERR scores, respectively, for the different methods. The results reveal some clear general trends. First, the exponentially weighted ensemble of calibrated ADABOOST.MH (AB+EXP) and multi-class SVM classifier (SVM+EXP) outperform all of the baseline methods in terms of both evaluation metrics of interest, with the exception of the MQ_{2007} data set. On this single set the winner is the RANKSVM algorithm, which achieves an excellent NDCG_{10} score (the ERR score, however, is lower than the ERR score of our methods).

Second, it is interesting that the one best calibrated ADABOOST.MH model outperforms RANKBOOST; and, similarly, the single calibrated MC-SVM outperforms the RANKSVM in many cases. RANKBOOST can be thought of as the ranker counterpart of ADABOOST.MH: RANKBOOST optimizes the rank loss in a stepwise fashion similar to ADABOOST, which optimizes the exponential loss. RANKSVM can be viewed as the ranking counterpart of the calibrated MC-SVM, since the core of both algorithms is a quadratic optimization although with different loss functions. In other words, using an appropriate calibration function, classical multi-class classifiers can have state-of-the-art ranking performance. Note also that in RANKBOOST, it is not easy to design base learners that can optimize the weighted rank loss, so we could only use an updated version of the decision stump introduced in the original RANKBOOST paper (Freund et al. 2003). In contrast, in ADABOOST.MH we were able to use a larger variety of standard base learners.

We also report the scores we achieved when just using the class-probability-calibrated models (denoted by EXP.CPC) or just using the regression-based-calibration models (denoted by EXP.RBC) in the exponential weighted ensemble for both multi-class classifiers applied here. Both calibrations achieve similar scores, except for the CPC multi-class SVM

Table 2 The NDCG₁₀ values of the methods compared on benchmark data sets. The best scores for each data set and those within one sigma are marked in bold. Here, we used the standard deviation of the average NDCG₁₀. Pointwise, pairwise, and listwise approaches are with red (♣), green (♠), and blue (◇) backgrounds, respectively

Database	OHSUMED	MQ ₂₀₀₇	MQ ₂₀₀₈	YAHOO ₁	YAHOO ₂	MS ₁	MS ₂
♣ BF.NDCG	0.443	0.562	0.776	0.732	0.710	0.352	0.355
◇ ADARANK.ERR	0.441	0.554	0.770	0.733	0.719	0.321	-
◇ ADARANK.NDCG	0.445	0.554	0.767	0.723	0.731	0.353	-
♠ RANKBOOST	0.446	0.571	0.780	0.754	0.747	0.421	-
♠ RANKSVM	0.466	0.584	0.781	0.761	0.744	0.357	-
◇ RANKNET	0.452	0.541	0.729	0.653	0.686	-	-
◇ CA	0.448	0.580	0.783	0.754	0.737	0.451	-
♣ AB+EXP	0.469	0.578	0.790	0.789	0.769	0.484	0.484
♣ AB+EXP.CPC	0.472	0.583	0.791	0.787	0.768	0.478	0.464
♣ AB+EXP.RBC	0.464	0.579	0.788	0.786	0.755	0.482	0.474
♣ AB+ONEBEST	0.446	0.563	0.779	0.777	0.759	0.477	0.476
♣ MC-SVM+EXP	0.475	0.569	0.774	0.768	0.756	0.297	-
♣ MC-SVM+EXP.CPC	0.418	0.537	0.767	0.768	0.754	0.321	-
♣ MC-SVM+EXP.RBC	0.474	0.569	0.776	0.762	0.749	0.297	-
♣ MC-SVM+ONEBEST	0.455	0.561	0.761	0.768	0.755	0.297	-

Table 3 The ERR of the methods compared on benchmark data sets. The best scores for each data set and those within one sigma are marked in bold. We used the standard deviation of the average ERR. Pointwise, pairwise, and listwise approaches have red (♣), green (♠), and blue (◇) backgrounds, respectively

Database	OHSUMED	MQ ₂₀₀₇	MQ ₂₀₀₈	YAHOO ₁	YAHOO ₂	MS ₁	MS ₂
♣ BF.NDCG	0.534	0.296	0.297	0.429	0.421	0.299	0.303
◇ ADARANK.ERR	0.534	0.292	0.294	0.437	0.439	0.220	-
◇ ADARANK.NDCG	0.519	0.292	0.287	0.412	0.446	0.255	-
♠ RANKBOOST	0.544	0.314	0.303	0.432	0.442	0.309	-
♠ RANKSVM	0.570	0.318	0.301	0.438	0.432	0.248	-
◇ RANKNET	0.542	0.286	0.266	0.343	0.388	-	-
◇ CA	0.540	0.317	0.306	0.447	0.444	0.340	-
♣ AB+EXP	0.549	0.320	0.312	0.462	0.457	0.365	0.363
♣ AB+EXP.CPC	0.573	0.322	0.312	0.461	0.457	0.361	0.352
♣ AB+EXP.RBC	0.549	0.322	0.313	0.461	0.455	0.364	0.358
♣ AB+ONEBEST	0.506	0.294	0.294	0.457	0.454	0.360	0.357
♣ MC-SVM+EXP	0.579	0.315	0.299	0.451	0.451	0.242	-
♣ MC-SVM+EXP.CPC	0.505	0.297	0.295	0.450	0.451	0.226	-
♣ MC-SVM+EXP.RBC	0.577	0.314	0.300	0.450	0.450	0.241	-
♣ MC-SVM+ONEBEST	0.554	0.306	0.293	0.450	0.451	0.242	-

which gives below par performance scores on MQ₂₀₀₇ and OHSUMED. The reason for this is that in these data sets there are many queries where all of the feature values are zeros. In this case, an SVM-based method results in zero scores (unlike to ADABOOST.MH-based methods). SVM-based techniques also perform quite poorly on the MS data sets because the regularization penalty C cannot be set to its optimal value due to the excessive running time with large C .

The final ensemble step described in Sect. 6 has a significant impact when used on the classifiers produced by ADABOOST.MH. In the case of MC-SVM the improvement is marginal, but the exponentially weighted ensemble on linear models is computationally efficient both to train and to evaluate, so there is no reason why we cannot use the full ensemble.

The listwise algorithms (ADARANK, RANKNET and CA) perform well on the LETOR data sets (OHSUMED, MQ₂₀₀₇, MQ₂₀₀₈).¹⁴ However, pointwise and pairwise approaches significantly outperform the computationally expensive listwise algorithms on the YAHOO and MS data sets and slightly outperform them on the LETOR data sets. The only exception is the CA algorithm, which can achieve a state-of-the-art performance even on large-scale learning-to-rank benchmark data sets.

The methods using SVM engines are horrendously slow on large-scale data sets. For example, running RANKSVM and MC-SVM takes more than three weeks for a single fold of the MS data sets if the regularization coefficient C is larger than 10^{-4} . Running them with smaller coefficients, though, produces suboptimal results, which explains why we were not able to get state-of-the-art performance scores using these methods for MS data sets.

7.5 Learning curves

In this set of experiments we investigated how the performance scores of the methods depend on the size of the training set and on the quality of the queries. In the first experiment, we randomly divided the YAHOO₂ training set into ten equal parts, and trained the rankers on 10 %, 20 %, ..., 100 % of the available data. The YAHOO₂ data set was better suited to this experiment because, on the one hand, the test set consisted of almost 3800 queries, so small differences in NDCG scores could be still significant, and on the other hand, the training size of about 1300 queries was not prohibitively large, so all of the methods could be trained in a reasonable time.

Figure 3(a) shows the results we got. Although it is hard to draw general conclusions from one experiment, first, it seems that listwise methods (ADARANK, RANKNET and CA) are more competitive on small data sets and their learning curves “flatten” as we add more and more data. Still, pointwise and pairwise methods seem to improve steadily as the data size grows. Second, among our two classification-based techniques, the final ensemble step (Sect. 6) helps ADABOOST.MH in the full range of data sizes, whereas it helps MC-SVM only for smaller sets. Lastly, among the two pairwise techniques, RANKBOOST seems to work better for small data whereas the NDCG score of RANKSVM grows faster, which is somewhat unfortunate as we know that RANKSVM will be hard to optimize for larger data sets.

In a variant of this experiment we investigated how noise affects the performance of the methods. First, using the BESTFEATURE ranker we computed the NDCG score for each query. A low score on a query indicates that the features cannot capture the relevant non-trivial semantics between the query and the documents. This is one of the main sources of noise in a learning-to-rank task (besides the label noise coming from the disagreement of the annotators). We then trained the rankers on 10 %, 20 %, ..., 100 % of the available data as in the previous experiment, but this time we added the queries in (decreasing or increasing) order of their NDCG scores. Figures 3(b) and 3(c) show the results. As expected, the learning curves are significantly flatter when we keep bad queries till the end. Besides the same general trends observed in the previous experiment, it is interesting to see that listwise methods seem to work well only if trained using good queries, whereas pairwise and pointwise techniques improve even when bad queries are added to the training set. It is also noticeable that the best calibrated ADABOOST.MH model performs very well on the best

¹⁴Since we did not use the official evaluation scripts for of the reasons stated in Sect. 7.1, our scores cannot be compared to the official LETOR scores in an absolute sense, but we could reproduce the order of the methods available at the LETOR web site.

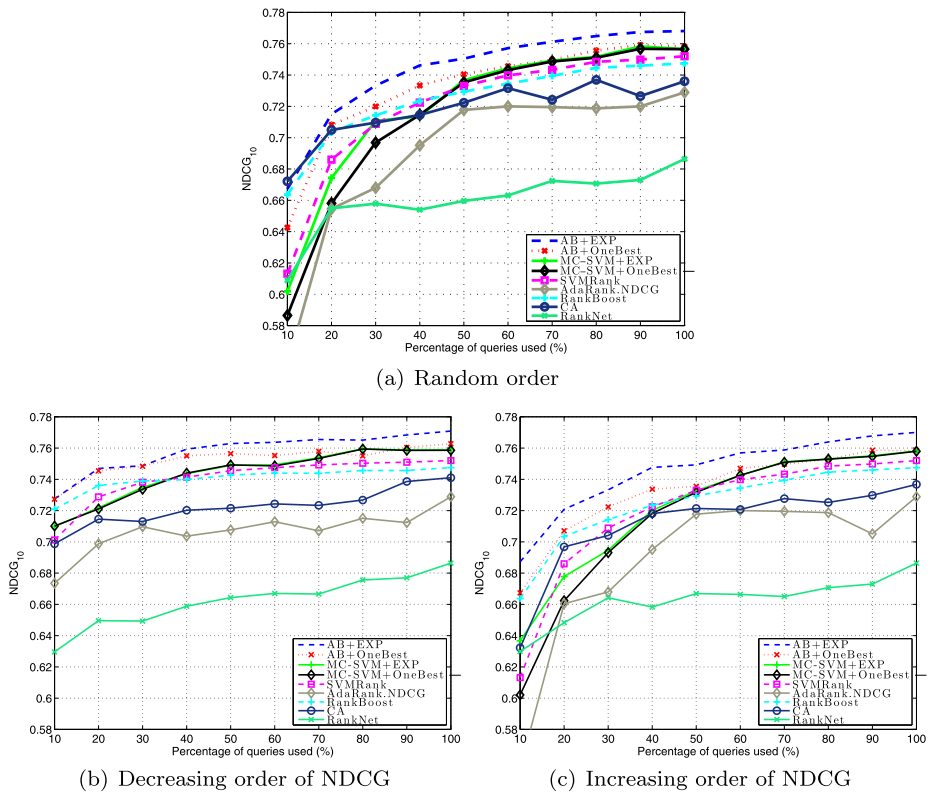


Fig. 3 The dependence of $NDCG_{10}$ scores on training data size. The queries were added gradually to the YAHOO₂ training data in 10 % portions. In the *top panel* the queries were ordered randomly, whereas in the *left and right panels* they were added in decreasing and increasing order, respectively, according to their $NDCG_{10}$ scores obtained by using the BESTFEATURE ranker

10 % to 30 % of the queries, but the ensemble step helps significantly when noisy queries start to accumulate.

7.6 Normalizing the relevance grades

We carried out some experiments to assess the effect of the query-wise normalization of relevance grades proposed in Ravikumar et al. (2011). In the regression-based calibration setup, we calibrated the trained multi-class classifiers in two ways: (1) using the relevance grades as target values according to (20) and (2) using the relevance grades *normalized query-wise* by the ideal DCG_{10} scores for each query according to (21).

We Applied ADABOOST.MH to get multi-class classifiers using the same set of base classifiers that described in Sect. 7.3, and we ran it for 10000 iterations for the YAHOO₁ and YAHOO₂ data sets. Then we calibrated the classifiers using linear regression, logit regression, neural network regression, and polynomial regression of degree 2. The $NDCG_{10}$ scores in Fig. 4 indicate that this step did indeed improve the individual rankers.

The intuitive rationale of relevance normalization is that normalizing the relevance scores by the ideal DCG scores will balance the contribution of the individual queries to the loss (in this case L_2) to be optimized. In particular, the normalization downweights queries having

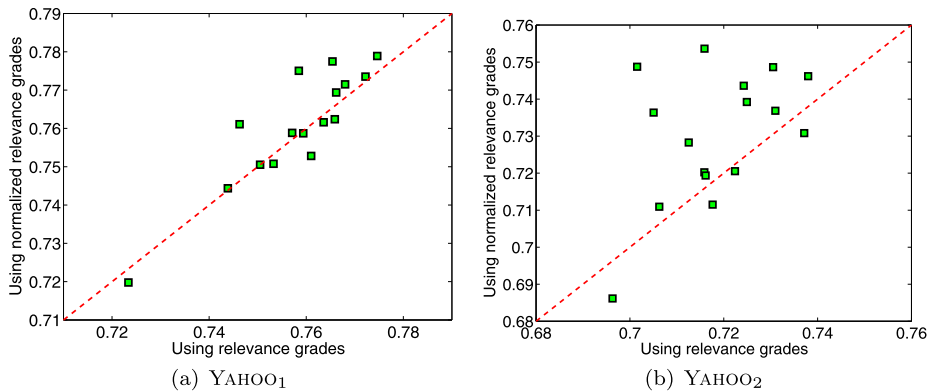
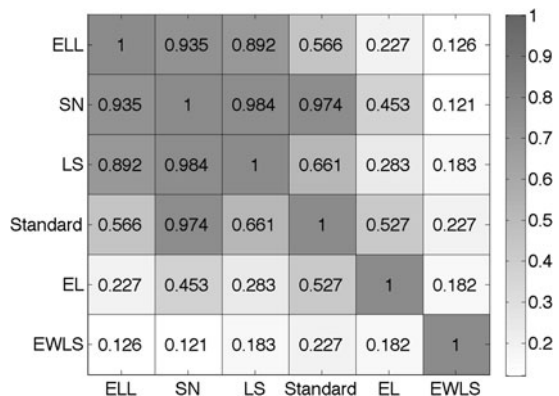


Fig. 4 The scatterplot of $NDCG_{10}$ scores computed by using the relevance grades normalized by the ideal DCG_{10} query-wise according to (21) (vertical axis) versus $NDCG_{10}$ scores computed by just using the original relevance grades ($r = 2^{y-1} - 1$) in the RBC calibration according to (20) (horizontal axis). The rectangles show the scores of the calibrated models

Fig. 5 The p -values for different calibrations obtained via Fischer’s method on foldwise p -values of the t -test, Letor 4.0/MQ2007. The class probability based calibration was calculated as described in Sect. 5.2



high DCG scores. These queries can be thought of as “easy” with a large number of relevant documents, so downweighting them means that the calibration will focus on harder queries that may further improve the generalization performance.

7.7 The diversity of class-probability-based calibration outputs

To investigate how diverse the score values of different class probability calibrated models are, we compared the scores obtained by the five CPC methods described in Sect. 5.2 using the t -test. We obtained five p -values for each CPC pair. Then we applied Fischer’s method to get one overall p -value, assuming that these five p -values came from independent statistical tests. Here, we just used the output of boosted trees with the number of tree leaves set to 8.

The results in Fig. 5 indicate that for a subset of TCFs, the estimated probability distributions were quite close to each other. Although the TCFs are rather different, it seems that they approximate a similar distribution with only small differences. We think that one reason for the observed efficiency of the proposed method is that these small differences within the cluster are due to the estimation noise, so by mixing them, the level of the noise decreases.

Table 4 The ERR and NDCG₁₀ scores on the official Yahoo! Learning-to-Rank Challenge entries. The entries are in order of their ERR scores. Scores within one sigma from the winner score are in bold. Teams with red (♣), green (♠), and blue (◇) backgrounds used pointwise, pairwise, and listwise approaches, respectively. We have no information on uncolored teams. The name of our team was LAL. For comparison, we also included the results of AB+EXP and RANKSVM from Tables 2 and 3 and the results of the Gradient Boosted Decision Tree algorithm (Friedman 2002) taken from (Chapelle and Chang 2011)

Rank	Track 1			Track 2		
		ERR	NDCG ₁₀		ERR	NDCG ₁₀
1	◇ Ca3Si2O7	0.469	0.804	♠ MN-U	0.463	0.786
2	♣ catona...	0.468	0.806	♣ arizona	0.463	0.788
3	MLG	0.467	0.803	♣ Joker	0.463	0.789
4	♣ Joker	0.467	0.805	♣ ULG-PG	0.462	0.782
5	♣ AG	0.466	0.802	VeryGoodSignal	0.462	0.785
6	♣ LAL	0.466	0.799	ya	0.461	0.783
7	HotStepper	0.465	0.797	♣ WashU	0.461	0.784
8	♣ WashU	0.464	0.798	♣ catona...	0.461	0.783
9	ya	0.464	0.798	CLTeam	0.461	0.781
10	♣ ULG-PG	0.464	0.797	yareg	0.461	0.778
11	yareg	0.463	0.796	♣ LAL	0.460	0.777
12	♠ MN-U	0.463	0.797	lily	0.460	0.769
	♠ RANKSVM	0.438	0.761	♠ RANKSVM	0.432	0.744
	♣ GBDT	0.462	0.790	♣ GBDT	—	—
	♣ AB+EXP	0.462	0.789	♣ AB+EXP	0.457	0.769

7.8 Yahoo! Learning-to-Rank Challenge

The YAHOO₁ and YAHOO₂ data sets were the official data sets of the Yahoo! Learning-To-Rank Challenge.¹⁵ This open challenge attracted scores of academic researchers as well as industrial practitioners, and drew a huge number of participants with over 300 teams coming from both industrial and academic areas. The data sets used in the challenge can be considered the first freely available large-scale data sets in learning-to-rank, allowing a more reliable benchmarking tests than the earlier ones based on the LETOR sets. The challenge revealed some important findings. (Table 4 shows the final scores achieved on the test set.) First, ensemble methods achieved the best scores. Almost without exception, the disseminated algorithms devised by the top teams were based on ensemble techniques. Second, the general consensus (coming mainly from benchmarks on the LETOR data sets (Cao et al. 2007; Valizadegan et al. 2009)) that pairwise and listwise techniques outperform pointwise approaches seems to be refuted. The majority of the best teams used pointwise approaches and they were competitive with the listwise and pointwise techniques. This finding agrees with our experimental results on benchmark data sets: some of the pairwise and listwise methods were on par with our approach on the LETOR data sets, but on the larger YAHOO₁₋₂ and MS₁, our pointwise technique performed significantly better than the pairwise and listwise methods.¹⁶

We also participated in the Yahoo! Learning-To-Rank Challenge with an earlier version of our approach (Busa-Fekete et al. 2011a). The algorithm we used there resembles the most

¹⁵<http://learningtorankchallenge.yahoo.com/>.

¹⁶To be fair, the winning method on the larger YAHOO₁ data set used a listwise technique (Burges et al. 2011). Unfortunately, their code is private so we could not include it in our benchmark tests on the other data sets.

to AB+EXP.RBC. The main difference is that we further diversified our ensemble by merging several relevance grades to create five different multi-class classification problems. This created further diversity in the ensemble and pushed our entry into the top 6. Label-grouping could be included in any classification-based ensemble technique. We decided to not to use it in the benchmark tests in this paper to keep the method simple and computationally less expensive.

8 Conclusions and future work

In this paper, we described a generic technique for learning to rank. The method consists of three steps: (1) training several multi-class classifiers to predict the relevance labels, (2) calibrating the output score vectors to predict either the posterior probabilities of the relevance labels or the real-valued relevance grades, and (3) combining the generated models using a simple exponential weighting scheme. The advantages of the method are its conceptual simplicity, its practical performance, and its computational efficiency. We also presented a theoretical analysis where we examined the link between the Kullback-Leibler divergence and the expected DCG. We showed that the better estimate of the conditional probability distribution of relevance labels in terms of Kullback-Leibler divergence results in a higher DCG score in our probabilistic setup.

In experiments we showed that our essentially pointwise approach is competitive with more complex methods including RANKSVM, ADARANK RANKBOOST, CA, or RANKNET on most of the available large learning-to-rank benchmark data sets. In a comparison of multi-class classifiers, we found that ADABOOST.MH is better suited for this task than MC-SVM. The main bottleneck of MC-SVM is that it slows down for larger trade-off parameters on large-scale data sets, making it difficult to achieve its optimal performance. To alleviate this problem, as a further study, we plan to test the algorithm described in Hazan and Kale (2011), where MC-SVM is trained in an online bandit setup.

We also investigated how the performance of different algorithms evolves as the size of the data grows. We found that pointwise and pairwise techniques in general, and our approach in particular, scale better for large data sets.

The results of our paper along with the findings of the Yahoo! Learning-To-Rank Challenge underscores the performance of ensemble rankers. Their applicability in practice is mainly limited by the fact that they have to evaluate many rankers at test time, and it is well known that the evaluation time is crucial in a real world learning-to-rank application. This motivates the development of a framework in which a controller can select the rankers to be evaluated based on the characteristics of individual queries (Cambazoglu et al. 2010). Our future goal here is to model the problem as a Markov decision process, and solve it using standard reinforcement learning techniques (Dulac-Arnold et al. 2011; Benbouzid et al. 2011, 2012a).

Acknowledgements We would like to thank Van Dong for distributing the RankLib package and for his useful comments. This work was supported by the ANR-2010-COSI-002 grant of the French National Research Agency.

Appendix A: Training ADABOOST.MH

Running a full search in each boosting iteration of ADABOOST.MH is prohibitively expensive, so we decided to run an accelerated version based on a multi-armed bandit (MAB) setup (Busa-Fekete and Kégl 2010). In this section we shall describe the algorithmic details.

```

ADABOOST.MH.BA( $\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(1)}, \text{BASE}(\cdot, \cdot, \cdot, \cdot), T, \mathcal{G} = \{\mathcal{H}_1, \dots, \mathcal{H}_M\}, \text{BANDITALGO}$ )
1 for  $t \leftrightarrow 1$  to  $T$ 
2    $j \leftrightarrow \text{BANDITALGO.getArm}()$ 
3    $\mathbf{h}^{(t)}(\cdot) \leftrightarrow \alpha^{(t)} \mathbf{v}^{(t)} \varphi^{(t)}(\cdot) \leftrightarrow \text{BASE}(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}, \mathcal{H}_j)$ 
4    $\gamma_{\mathcal{H}_j}^{(t)} \leftrightarrow \sum_{i=1}^n w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i \quad \triangleright \text{edge} = 1 - 2 \times \text{error}$ 
5    $r_j^{(t)} = -\log \sqrt{1 - \gamma_{\mathcal{H}_j}^{(t)}} \quad \triangleright \text{calculate reward based on edge}$ 
6    $\text{BANDITALGO.receiveReward}(j, r_j^{(t)})$ 
7   for  $i \leftrightarrow 1$  to  $n$  for  $\ell \leftrightarrow 1$  to  $K$ 
8      $w_{i,\ell}^{(t+1)} \leftrightarrow w_{i,\ell}^{(t)} \frac{\exp(-h_{\ell}^{(t)}(\mathbf{x}_i) y_{i,\ell})}{\sum_{i'=1}^n \sum_{\ell'=1}^K w_{i',\ell'}^{(t)} e^{-h_{\ell'}^{(t)}(\mathbf{x}_{i'}) y_{i',\ell'}}$ 
9 return  $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^T \mathbf{h}^{(t)}(\cdot)$ 

```

Fig. 6 The pseudocode of the bandit-accelerated version of ADABOOST.MH.BA algorithm. \mathbf{X} is the observation matrix, \mathbf{Y} is the label matrix, $\mathbf{W}^{(1)}$ is the initial weight matrix, $\text{BASE}(\cdot, \cdot, \cdot, \cdot)$ is the base learner algorithm, and T is the number of iterations, BANDITALGO is the bandit algorithm. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) classifier

For the formal description, let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ be the $n \times d$ observation matrix, where $x_i^{(j)}$ are the elements of the d -dimensional observation vectors $\mathbf{x}_i \in \mathbb{R}^d$. We are also given a label matrix $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ of dimension $n \times K$, where $\mathbf{y}_i \in \{+1, -1\}^K$. In multi-class classification one and only one of the elements of \mathbf{y}_i is set to $+1$. We will denote the index of the correct class by $\ell(\mathbf{x}_i)$ or simply ℓ_i .

The general idea of accelerating the base learner is to partition the base classifier space \mathcal{H} into (not necessarily disjoint) subsets $\mathcal{G} = \{\mathcal{H}_1, \dots, \mathcal{H}_M\}$ and use MABs to learn the usefulness of the subsets. Each arm represents a subset, so, in each iteration, the bandit algorithm selects a subset. The base learner then finds the best base classifier *in the subset* (instead of searching through the *whole space* \mathcal{H}), and returns a reward based on this optimal base learner. Based on the rewards, the MAB algorithm gradually learns the quality or usefulness of the subsets. We used an adversarial bandit algorithm (EXP3G (Auer et al. 2002)) because our earlier experiments showed that using this bandit algorithm, the strong learner obtained via ADABOOST.MH.BA has a slight but consistently superior accuracy, while training is also faster than when using stochastic bandit algorithms, such as UCB (Auer et al. 1995) and UCBV (Audibert et al. 2009). The schematic overview of ADABOOST.MH.BA can be seen in Fig. 6.

We used two well-boostable base learners, i.e. *decision trees* and *decision products* (Kégl and Busa-Fekete 2009). Our decision tree implementation is very similar to Quinlan's C4.5 algorithm (Quinlan 1993), except that we use the edge improvement (instead of Quinlan's entropy-based criterion) to select the next node and the threshold to split. The decision tree has one hyperparameter, the number of leaves N , which also appears as a linear factor in the running time. The second base learner we used is *decision products*. The goal of this learner is to optimize *products* $\mathbf{h}(\cdot) = \alpha \prod_{j=1}^m \mathbf{v}_j \varphi_j(\cdot)$ of simple base classifiers $\mathbf{v}_j \varphi_j(\cdot)$, where the vote vectors \mathbf{v}_j are multiplied element-wise. This base learner has also only one hyperparameter; namely, the number of terms m .

The idea behind ADABOOST.MH.BA cannot be applied directly with decision trees or decision products, since it is hard to determine an eligible partitioning of the space of decision trees or decision products, due to the enormous size of this space. Thus, we followed the setup proposed in Busa-Fekete and Kégl (2009) in which trees and products are modeled as sequences of decisions over the smaller and manageable partitioning used for stumps. For simple decision stumps, the most natural partitioning is to assign a subset to each feature: $\mathcal{H}_j = \{\varphi_{j,b}(\mathbf{x}) : b \in \mathbb{R}\}$, where j is the index of the selected feature and b is the decision threshold, and the base classifier is of the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise.} \end{cases} \quad (25)$$

Motivated by a one-against-all scheme for multi-class classification, the standard way to weight the ℓ th label of the i th instance in ADABOOST.MH is

$$w_{i,\ell}^{(1)} = \begin{cases} 1/(2n) & \text{if } \ell_i = \ell, \\ 1/(2n(K-1)) & \text{otherwise,} \end{cases}$$

where ℓ_i is the correct label of the i th query-document pair, n is the number of training instances, and K is the number of classes.¹⁷ Instead of this setup, we further up-weighted relevant instances exponentially proportionally to their relevance, so, for example, an instance \mathbf{x}_i with relevance $\ell_i = 4$ was twice as important in the global training cost as an instance with relevance $\ell_i = 3$, and four times as important than an instance with relevance $\ell_i = 2$. Formally, the initial (unnormalized) weight of ℓ th label of the i th instance is

$$w_{i,\ell}^{(1)} = \begin{cases} 2^{\ell_i} & \text{if } \ell_i = \ell, \\ 2^{\ell_i}/(K-1) & \text{otherwise.} \end{cases}$$

The weights are then normalized to sum to 1. This weighting scheme was motivated by the evaluation metric: the weight of an instance in the NDCG score is exponentially proportional to the relevance label of the instance itself.

We used our open source implementation (Benbouzid et al. 2012b) available at <http://multiboost.org>, in which this bandit-based approach is implemented along with the base learners mentioned above. In addition, our package allows the user to apply arbitrary initial weightings.

Appendix B: Multi-class Support Vector Machine

The Multi-Class Support Vector Machine (MC-SVM) was originally presented by Crammer and Singer (2001). We use only linear classifiers in our experiments because the kernelized version is computationally prohibitively expensive. Using the notation of the previous section, the linear MC-SVM classifier can be written as

$$\hat{\ell}_{\mathbf{M}}(\mathbf{x}) = \arg \max_{1 \leq \ell \leq K} \mathbf{m}_{\ell}^{\top} \mathbf{x},$$

¹⁷For a thorough explanation, see (2) in Kégl and Busa-Fekete (2009), Sect. 7.2 in Schapire and Singer (1999), or Sect. A.1 in the documentation at <http://www.multiboost.org/download/documentation.pdf>.

where $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_K)$ is a $d \times K$ real matrix. For a given matrix \mathbf{M} , the output scores can be calculated as $\mathbf{f}_{\mathbf{M}}(\mathbf{x}) = (\mathbf{m}_1^T \mathbf{x}, \dots, \mathbf{m}_K^T \mathbf{x})$. To bound the empirical multi-class error

$$R(\mathbf{M}) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}_{\{y_i, \hat{\ell}_{\mathbf{M}}(\mathbf{x}) < 0\}}$$

from above, the *multi-class hinge loss* is defined as

$$Q(\mathbf{M}, \mathbf{x}, \ell) = \max_{1 \leq \ell' \leq K} (\mathbf{m}_{\ell'}^T \mathbf{x}_i + 1 - \delta_{\ell, \ell'}) - \mathbf{m}_{\ell}^T \mathbf{x}_i,$$

where ℓ is the label index of the instance \mathbf{x} , and $\delta_{p,q}$ is Kronecker delta.¹⁸ The multi-class hinge loss $Q(\mathbf{M}, \mathbf{x}_i, \ell_i)$ is zero if the score of the correct class $\mathbf{m}_{\ell_i}^T \mathbf{x}_i$ beats the score of the second best class by a margin of at least one; otherwise it grows linearly with the difference between the highest score of the non-correct classes and the score of the correct class. Thus, it is not hard to see that

$$R(\mathbf{M}) \leq \frac{1}{n} \sum_{i=1}^n Q(\mathbf{M}, \mathbf{x}_i, \ell_i).$$

Introducing the slack variables ξ_i to deal with training points with a margin of less than one, the optimization task is

$$\min_{\mathbf{M}} \frac{1}{2} \|\mathbf{M}\|_2^2 + \frac{C}{n} \sum_{i=1}^n \xi_i$$

$$\text{subject to } \mathbf{m}_{\ell_i}^T \mathbf{x}_i + \delta_{\ell_i, \ell} - \mathbf{m}_{\ell}^T \mathbf{x}_i \geq 1 - \xi_i, \quad \text{for all } 1 \leq i \leq n, 1 \leq \ell \leq K,$$

where the parameter C controls the trade-off between the margin and the training error. In our experiments C was the only parameter to be tuned. We used an open source implementation of MC-SVM available at http://svmlight.joachims.org/svm_multiclass.html.

References

- Aioli, F., & Sperduti, A. (2010). A preference optimization based unifying framework for supervised learning problems. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 19–42). Berlin: Springer.
- Audibert, J. Y., Munos, R., & Szepesvári, C. (2009). Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19), 1876–1902.
- Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. (1995). Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th annual symposium on foundations of computer science* (pp. 322–331). Los Alamitos: IEEE Computer Society Press.
- Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. (2002). The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1), 48–77.
- Benbouzid, D., Busa-Fekete, R., & Kégl, B. (2011). MDDAG: learning deep decision DAGs in a Markov decision process setup. In *NIPS'11 workshop on deep learning and unsupervised feature learning*.
- Benbouzid, D., Busa-Fekete, R., & Kégl, B. (2012a). Fast classification using sparse decision DAGs. In *Proceedings of the 29th international conference on machine learning*.
- Benbouzid, D., Busa-Fekete, R., Casagrande, N., Collin, F. D., & Kégl, B. (2012b). MultiBoost: a multi-purpose boosting package. *Journal of Machine Learning Research*, 13, 549–553.

¹⁸ $\delta_{p,q}$ is equal to 1 if $p = q$ and zero otherwise.

- Bennett, J., & Lanning, S. (2007). The Netflix prize. In *Proceedings of KDD cup and workshop 2007* (pp. 3–6).
- 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 22th international conference on machine learning* (pp. 89–96).
- Burges, C., Svore, K., Bennett, P., Pastusiak, A., & Wu, Q. (2011). Learning to rank using an ensemble of lambda-gradient models. In *Yahoo! Learning-to-rank challenge (JMLR W&CP)* (Vol. 14, pp. 25–35).
- Busa-Fekete, R., & Kégl, B. (2009). Accelerating AdaBoost using UCB. In *KDDCup 2009 (JMLR W&CP)*, Paris, France (Vol. 7, pp. 111–122).
- Busa-Fekete, R., & Kégl, B. (2010). Fast boosting using adversarial bandits. In *International conference on machine learning* (Vol. 27, pp. 143–150).
- Busa-Fekete, R., Kégl, B., Éltes, T., & Szarvas, G. (2011a). Ranking by calibrated AdaBoost. In *JMLR W&CP* (Vol. 14, pp. 37–48).
- Busa-Fekete, R., Kégl, B., Éltes, T., & Szarvas, G. (2011b). A robust ranking methodology based on diverse calibration of AdaBoost. In *LNCS: Vol. 6911. European conference on machine learning* (pp. 263–279).
- Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., & Degenhardt, J. (2010). Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on web search and data mining* (pp. 411–420).
- Cao, Z., Qin, T., Liu, T., Tsai, M., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on machine learning* (pp. 129–136).
- Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. New York: Cambridge University Press.
- Chapelle, O., & Chang, Y. (2011). Yahoo! Learning-to-rank challenge overview. In *Yahoo! Learning-to-rank challenge (JMLR W&CP)* (Vol. 14, pp. 1–24).
- Chapelle, O., & Wu, M. (2010). Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13(3), 216–235.
- Chapelle, O., Metzler, D., Zhang, Y., & Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proceeding of the 18th ACM conference on information and knowledge management* (pp. 621–630). New York: ACM.
- Chapelle, O., Chang, Y., & Liu, T. (Eds.) (2011). *Yahoo! Learning-to-rank challenge. Journal of Machine Learning Research, Workshop and Conference Proceedings* (Vol. 14).
- Chu, W., & Keerthi, S. (2005). New approaches to support vector ordinal regression. In *Proceedings of the 22nd international conference on machine learning* (pp. 145–152).
- Cortes, C., & Mohri, M. (2005). Confidence intervals for the area under the ROC curve. In *Advances in neural information processing systems* (Vol. 18). Cambridge: MIT Press.
- Cossock, D., & Zhang, T. (2008). Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11), 5140–5154.
- Crammer, K., & Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2, 265–292.
- Dror, G., Boullé, M., Guyon, I., Lemaire, V., & Vogel, D. (Eds.) (2009). *Proceedings of KDD-cup 2009 competition. JMLR workshop and conference proceedings* (Vol. 7).
- Dulac-Arnold, G., Denoyer, L., Preux, P., & Gallinari, P. (2011). Datum-wise classification: a sequential approach to sparsity. In *Proceedings of the European conference on machine learning and knowledge discovery in databases (ECML-PKDD-11)*, Athens, Greece, (pp. 375–390), Part I.
- Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 119–139.
- 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, J. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4), 367–378.
- Gruenewald, P. (2007). *The minimum description length principle*. Cambridge: MIT Press.
- Hazan, E., & Kale, S. (2011). *Advances in neural information processing systems: Vol. 25. NEWTRON: an efficient bandit algorithm for online multiclass prediction*. Cambridge: MIT Press.
- Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20, 422–446.
- Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD)*.
- Kégl, B., & Busa-Fekete, R. (2009). Boosting products of base classifiers. In *International conference on machine learning*, Montreal, Canada (Vol. 26, pp. 497–504).
- Kekäläinen, J., & Järvelin, K. (2002). Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53(13), 1120–1129.

- Kotowski, W., Dembczynski, K., & Hüllermeier, E. (2011). Bipartite ranking through minimization of univariate loss. In *Proceedings of the 28th international conference on machine learning* (pp. 1113–1120).
- Li, P., Burges, C., & Wu, Q. (2007). McRank: learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems* (Vol. 19, pp. 897–904). Cambridge: MIT Press.
- Mease, D., Wyner, A., & Buja, A. (2007). Boosted classification trees and class probability/quantile estimation. *Journal of Machine Learning Research*, 8, 409–439.
- Metzler, D., & Croft, B. W. (2007). Linear feature-based models for information retrieval. *Information Retrieval*, 10, 257–274.
- Niculescu-Mizil, A., & Caruana, R. (2005). Obtaining calibrated probabilities from boosting. In *Proceedings of the 21st international conference on uncertainty in artificial intelligence* (pp. 413–420).
- Platt, J. (2000). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schoelkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers* (pp. 61–74). Cambridge: MIT Press.
- Quinlan, J. (1993). *C4.5: programs for machine learning*. San Mateo: Morgan Kaufmann.
- Ravikumar, P., Tewari, A., & Yang, E. (2011). On NDCG consistency of listwise ranking methods. In *JMLR workshop and conference proceedings, AISTATS*.
- Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11, 416–431.
- Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3, 333–389.
- Sakai, T. (2007). On the reliability of information retrieval metrics based on graded relevance. *Information Processing & Management*, 43(2), 531–548.
- Schapire, R., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3), 297–336.
- Valizadegan, H., Jin, R., Zhang, R., & Mao, J. (2009). Learning to rank by optimizing NDCG measure. In *Advances in neural information processing systems* (Vol. 22, pp. 1883–1891).
- Wu, Q., Burges, C. J. C., Svore, K. M., & Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3), 254–270.
- Wu, T., Lin, C., & Weng, R. (2004). Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5, 975–1005.
- Xu, J., & Li, H. (2007). AdaRank: a boosting algorithm for information retrieval. In *SIGIR '07: proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 391–398). New York: ACM.