Learning to Rank Using an Ensemble of Lambda-Gradient Models

Christopher J. C. Burges CBURGES@MICROSOFT.COM
Krysta M. Svore KSVORE@MICROSOFT.COM
Paul N. Bennett PAUBEN@MICROSOFT.COM
Andrzej Pastusiak ANDRZEJP@MICROSOFT.COM
Qiang Wu QIANGWU@MICROSOFT.COM
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA

Editor: Olivier Chapelle, Yi Chang, Tie-Yan Liu

Abstract

We describe the system that won Track 1 of the Yahoo! Learning to Rank Challenge.
Keywords: Learning to Rank, Gradient Boosted Trees, Lambda Gradients, Web Search

1. Introduction and Summary

The Yahoo! Learning to Rank Challenge, Track 1, was a public competition on a Machine Learning for Information Retrieval task: given a set of queries, and given a set of retrieved documents for each query, train a system to maximize the Expected Reciprocal Rank (Chapelle et al., 2009) on a blind test set, where the training data takes the form of a feature vector $\mathbf{x} \in \mathcal{R}^d$ with label $y \in \mathcal{Y}$, $\mathcal{Y} \equiv \{0, 1, 2, 3, 4\}$ (a more positive number denoting higher relevance) for each query/document pair (the original, textual data was not made available). The Challenge setup, background information, and results have been extensively covered elsewhere and we refer to Chapelle and Chang (2011) for details. In this paper we summarize the work that resulted in the winning system.[1] We limit the work described in this paper to the work done specifically for the Challenge; the work was done over a four week period prior to the end of the Challenge.

Our approach used a linear combination of twelve ranking models, eight of which were bagged LambdaMART boosted tree models, two of which were LambdaRank neural nets, and two of which were MART models using a logistic regression cost. LambdaRank is a method for learning arbitrary information retrieval measures; it can be applied to any algorithm that learns through gradient descent (Burges et al., 2006). LambdaRank is a listwise method, in that the cost depends on the sorted order of the documents. We briefly summarize the ideas here, where the $\mathbf{x}_i, i = 1, \ldots, m_q$, are taken to be the set of labeled feature vectors for a given query q and corresponding documents $d_i, i = 1, \ldots, m_q$. The key LambdaRank insight is to define the gradient of the cost with respect to the score that

1. Our team was named Ca3Si2O7, the chemical formula for Rankinite. We donated half of the $8000 prize money to the NIPS Foundation and half to the International Machine Learning Society (the organizers of ICML) for student scholarships.

the model assigns to a given $\mathbf{x}_i$ *after* all of the $\mathbf{x}_i, i = 1, ..., m_q$, have been sorted by their scores s_i; thus the gradients take into account the rank order of the documents, as defined by the current model. LambdaRank is an empirical algorithm, in that the form that the gradients take was chosen empirically: the λ's are those gradients, and the contribution to a given feature vector $\mathbf{x}_i$'s λ_i from a pair $\mathbf{x}_i, \mathbf{x}_j$, $y(\mathbf{x}_i) \neq y(\mathbf{x}_j)$, is just the gradient of the logistic regression loss (viewed as a function of $s_i - s_j$) multiplied by the change in Z caused by swapping the rank positions of the two documents while keeping all other documents fixed, where Z is the information retrieval measure being learned (Burges et al., 2006; Donmez et al., 2009). λ_i is then the sum of contributions for all such pairs. Remarkably, it has been shown that a LambdaRank model trained on Z, for Z equal to Normalized Cumulative Discounted Gain (NDCG) (Jarvelin and Kekalainen, 2000), Mean Reciprocal Rank, or Mean Average Precision (three commonly used IR measures), given sufficient training data, consistently finds a local optimum of that IR measure (in the space of the measure viewed as a function of the model parameters) (Donmez et al., 2009).

While LambdaRank was originally instantiated using neural nets, it was found that a boosted tree multiclass classifier ("McRank") gave improved performance (Li et al., 2007); combining these ideas led to LambdaMART, which instantiates the LambdaRank idea using gradient boosted decision trees (Friedman, 2001; Wu et al., 2009). This work showed that McRank's improved performance over LambdaRank (instantiated in a neural net) is due to the difference in the expressiveness of the underlying models (boosted decision trees versus neural nets) rather than being due to an inherent limitation of the lambda-gradient idea.[2] For a self-contained description of these algorithms we refer the reader to Burges (2010).

Regarding our ensemble approach, four of the LambdaMART rankers (and one of the nets) were trained by optimizing for the Expected Reciprocal Rank (ERR) measure (Chapelle et al., 2009), and four (and the other net) were trained by optimizing for NDCG. For the LambdaMART models, we also generated extended training sets by randomly deleting feature vectors for each query in the training data, and concatenating the resulting data into a new training set; four of the eight LambdaMART models were trained using such data (see below for details). We performed parameter sweeps to find the best parameters for the boosted trees (such as the number of leaves and the learning rate): the parameter sweeps resulted in variations in accuracy of up to 0.08 in the absolute value of NDCG. Once the best parameters were found, we performed 10-fold bagging without replacement for each LambdaMART model, thus using all available training data for that model. Note that this bagging step was done after all parameters had been fixed. The logistic regression models optimized for the graded relevance probabilities that ERR assigns (see below), and their outputs were also used as features in two of the LambdaMART models.

We explored various approaches to linearly combine the 12 resulting rankers, using the provided training set. For the combination method, we investigated (1) linear LambdaRank, (2) a method to optimally combine any pair of rankers given any IR metric, and (3) simple averaging. We found that (3) — simply adding the normalized model scores — worked as well as the other approaches, and somewhat surprisingly, that including the less accurate models (which were the logistic regression and neural net models) also helped.

2. In fact we also trained McRank models to include in our ensemble for the Challenge, but we found the performance to be sufficiently low that they were not included; this was likely a bug, but time did not permit pursuit.

Section 2 describes how we split the training data into local train, test and validation sets, and how we augmented the training data using random sampling. Section 3 describes how the computation of the lambda gradients for ERR can be computed in quadratic time, and provides a simple theorem as to the consistency of ERR. An experiment showing that lambda gradients can be used to directly learn ERR, our overall experimental protocol, and our results, are given in Section 4. While primarily this paper serves as a snapshot of the work we did towards the Challenge, the work also highlights some open questions about ensembles and lambda gradients, which we mention in Section 5.

2. Data

The Track 1 collection consisted of 19,944 train, 2,994 valid and 6,983 test queries, with 519 features. The mean number of documents per query was 24. All three sets were made available to contestants but only the training data had labels: to determine performance on the validation set, a model had to be submitted to the competition web page, and at the end of the competition, competitors had to assign which single one of their models would be tested on the test set. We did not make use of the validation or test feature vectors for training (e.g., for use in semi-supervised learning). To create local, labeled train/validation/test data, we shuffled the given training data (by query) and split it into ten equally sized sets. For the combining experiments below, we used 6 of those sets for training, 2 to compute the optimal combiner parameters, and 2 for test.

Since 19,944 queries is a rather small amount of data to train a web search ranker, we augmented the data to create two new training sets: in the first (Aug70), for each query, we randomly sampled 70% of the documents, and repeated this five times to create an augmented training set with 119,305 queries, and in the second (Aug50), for each query, we randomly sampled 50% of the documents, and repeated this ten times to create an augmented training set with 217,175 queries. (The concatenated augmented sets included the original data). The sampling was without replacement. Not only does augmenting the data in this manner provide added regularization, it also provides new information to the ranker, in the following sense: for example, if a given query has documents A, B, C with labels $4, 3, 2$, respectively, then one way in which the model could overfit is to learn that A should be ranked higher than C, conditioned on B lying between them. Removing B and adding the resulting data to the training set removes the possibility of such learned conditioning. However, the sampling also changes the distribution, and so overly aggressive sampling can be expected to hurt performance. We first tried the 70% sampling, and then given the encouraging results, also tried the more aggressive 50% sampling. Time limitations forbade further experimentation along this axis, but it is intriguing that such aggressive pruning still gave better individual model performance than just using the given training data.

3. Information Retrieval Measures

The competition used *Expected Reciprocal Rank (ERR)* (Chapelle et al., 2009) as the measure by which entries were evaluated. Experiments exploring how well lambda-gradient methods can be used to learn ERR are described in Section 4.1. In order to build a set of

diversely trained models, and because we know that our models perform well on NDCG, we also trained models using NDCG (Jarvelin and Kekalainen, 2000); see Burges (2010) for a detailed description of training lambda-gradient models on NDCG. In this section, we describe how the lambda gradients can be computed efficiently for ERR, and we also give a consistency proof for ERR.

We assume throughout this section that the documents have been sorted by score. ERR for a set of n documents for a given query is defined by

$$ERR \equiv \sum_{r=1}^{n} \frac{1}{r} R_r \prod_{i=1}^{r-1} (1 - R_i) \tag{1}$$

where, if $y(\mathbf{x}_i) \in \mathcal{Y}$ is the relevance label for the document at rank i (where the top ranked document is assigned rank 1), the

$$R_i = \frac{2^{y(\mathbf{x}_i)} - 1}{2^{y_m}} \tag{2}$$

model the probability of relevance conditioned on the label (where y_m is the maximum label value, namely 4). The R_i are in $[0, 1]$, in fact, $R_i \in \{0, 0.0625, 0.1875, 0.4375, 0.9375\}$. In order to compute the lambda gradients, for a given query and a given pair of (differently labeled) documents d_i and d_j, we need to compute the change in ERR when that pair of documents exchanges positions, with all other documents fixed. Let Δ denote the initial minus the final ERR values. Naively one might expect that the cost of computing all the Δ's for a given query would be $O(n^4)$, since Equation (1) appears to be quadratic in the number of documents, and since computing the Δ's requires computing a different ERR gain for every pair of documents. However as Chapelle et al. (2009) showed, the ERR can be computed in $O(n)$ time, and we can use a similar trick here to compute the Δ's for a given query in $O(n^2)$ time. In order to motivate the computation below, consider the following example. Letting $T_i \equiv 1 - R_i$, suppose that the 2nd and 6th documents are exchanged. From Equation (1) we see that

$$\begin{aligned} \Delta &= \frac{1}{2} T_1 (R_2 - R_6) + \frac{1}{3} T_1 (T_2 - T_6) R_3 + \frac{1}{4} T_1 (T_2 - T_6) T_3 R_4 \\ &+ \frac{1}{5} T_1 (T_2 - T_6) T_3 T_4 R_5 + \frac{1}{6} T_1 T_3 T_5 (T_2 R_6 - T_6 R_2) \end{aligned} \tag{3}$$

Examining Eq. (3), note that the terms take three forms, as exemplified by the first term above, the last, and the rest. Note also that differences corresponding to $i < 2$ and to $i > 6$ in Eq. (1) cancel. Thus creating an array A whose ith component is the ERR computed only to level i, we can write the general form of the above computation (for the case $i < j$) as

1. Compute $\pi_i \equiv \prod_{k=1}^{i} T_k$ for $i > 0$, and set $\pi_0 = 1$.
2. Set $\Delta = \pi_{i-1}(R_i - R_j)/r_i$.

3. Increment Δ by

$$\begin{aligned} & (T_i - T_j)\left(\frac{R_{i+1}}{r_{i+1}} + \frac{T_{i+1}R_{i+2}}{r_{i+2}} + \cdots + \frac{T_{i+1}\ldots T_{j-2}R_{j-1}}{r_{j-1}}\right)\pi_{i-1} \\ = \quad & (T_i - T_j)\left(\frac{A_{j-1} - A_i}{T_i}\right). \end{aligned}$$

4. Further increment Δ by

$$\frac{\pi_{i-1}}{r_j} T_{i+1}T_{i+2}\ldots T_{j-1}\left(T_i R_j - T_j R_i\right) \quad = \quad \frac{\pi_{j-1}}{r_j}\left(R_j - \frac{T_j R_i}{T_i}\right).$$

Note that the denominators never vanish since the T_i never vanish. Since computing A has linear complexity, and since A and π_i can be computed once per query, we see that structuring the computation in this way results in quadratic complexity.

Note that unlike NDCG, the change in ERR induced by swapping two documents d_i and d_j, while fixing the rank of all other documents, depends also on the labels and ranks of the documents with ranks between i and j. This raises the question as to whether ERR is consistent, in the sense that, given a pair of documents d_i, d_j with $i < j$ and with $y(\mathbf{x}_i) > y(\mathbf{x}_j)$, does swapping the rank positions of d_i and d_j necessarily decrease ERR? Calling this desired property 'pairwise consistency', we have

Theorem 1 *ERR is pairwise consistent for any choice of strictly increasing discounts r_i.*

Proof Following the above analysis, we just need to show that Δ is nonnegative when $R_i > R_j$. Using $T_i - T_j = R_j - R_i$ and collecting terms, we have that

$$\Delta = \frac{\pi_i(R_i - R_j)}{T_i}\left(\frac{1}{r_i} - \frac{R_{i+1}}{r_{i+1}} - \frac{T_{i+1}R_{i+2}}{r_{i+2}} - \cdots - \frac{T_{i+1}\ldots T_{j-2}R_{j-1}}{r_{j-1}} - \frac{T_{i+1}\ldots T_{j-1}}{r_j}\right)$$

Since $r_{i+1} > r_i$, this is certainly nonnegative if

$$\left(\frac{1}{r_i} - \frac{R_{i+1}}{r_i} - \frac{T_{i+1}R_{i+2}}{r_i} - \cdots - \frac{T_{i+1}\ldots T_{j-2}R_{j-1}}{r_i} - \frac{T_{i+1}\ldots T_{j-1}}{r_i}\right) \geq 0 \tag{4}$$

and hence if

$$R_{i+1} + T_{i+1}R_{i+2} + \cdots + T_{i+1}\cdots T_{j-2}R_{j-1} + T_{i+1}\cdots T_{j-1} \leq 1.$$

But since $R_i + T_i = 1$, the left-hand side just collapses to 1. ■

4. Experiments

4.1. Linear LambdaRank: A Sanity Check

We began by checking that modifying the LambdaRank gradients to model ERR, as had previously been done for MRR and MAP (Donmez et al., 2009), resulted in a system that

indeed learned ERR. Figure 1 shows the result of training a linear LambdaRank model on one tenth of the training data and testing on a validation set of the same size. We see that while training on ERR produces a small drop in the validation NDCG, training on ERR and training on NDCG gave essentially identical results, when testing on ERR. We thus verified that we could train using ERR directly; however the results on this small set were no better than those found by training on NDCG. Our hope was then that our more sophisticated models would be able to take advantage of being trained directly on ERR, and that the diversity added by training on two measures that gave similar test ERR accuracies would also help in the ensemble.

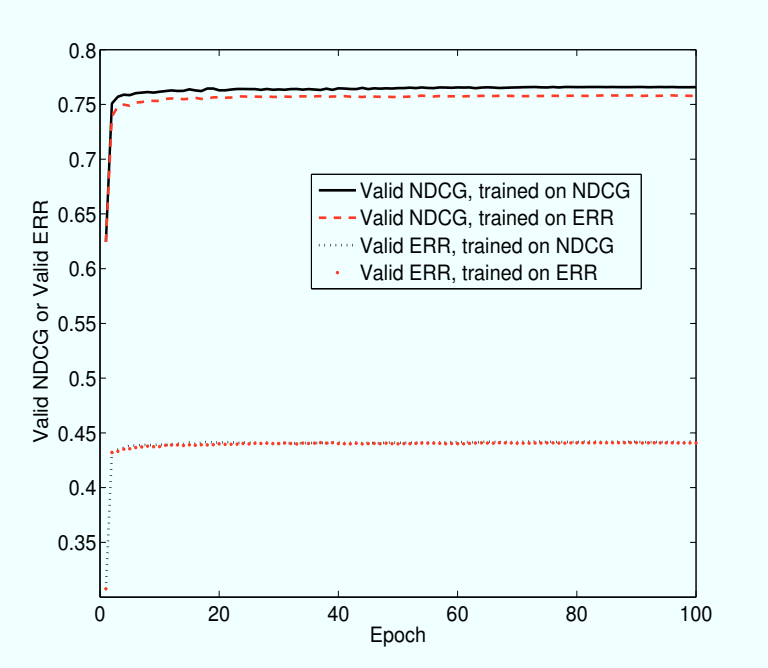

Figure 1: Training and testing a linear LambdaRank model using the NDCG and ERR measures.

4.2. The Models

Here we describe the models we used in our ensemble. Eight of the models were LambdaMART: four of these were trained to optimize NDCG and four, ERR. Within each set of four, we trained a model on the given training data, one on Aug50, one on Aug70, and one on the given training data, but using four additional features derived from two logistic regression models described below. We also trained two, two-layer LambdaRank neural nets, one optimized for NDCG, and one for ERR. We further trained two MART logistic regression models to model the conditional probabilities given in Equation (2) directly. The two MART models were very similar, and differed only in the normalization of their output scores: one had an added per-query score normalization to reduce the variance of the scores of low-scoring documents (letting $P_{d_{max}}$ denote the maximum document score for a

given query, the score for document d for that query is taken to be $P_{d_{max}}(1 + P_d - P_{d_{max}}))$. Finally, two of the LambdaMART models (one optimized for ERR, one for NDCG) were trained using additional features computed from the maximum and mean MART logistic regression scores, computed over all the documents, and also computed over the top three ranked documents. Note that these features do not depend directly on document features. The intuition behind this was mainly to increase the diversity of the ensemble.

For each LambdaMART and LambdaRank model, we performed extensive parameter sweeps on a 122-node MPI cluster, running Microsoft HPC Server 2008. The sweeps showed absolute variations of up to 0.03 in NDCG and 0.01 in ERR for LambdaRank, and up to 0.08 in NDCG and 0.02 in ERR for LambdaMART. For the neural nets (LambdaRank), the number of hidden nodes was varied from 10 to 30 in steps of 5, and the learning rate was varied from 10^{-4} to 10^{-2} (the optimal values were found to be 25 hidden nodes and a learning rate of 10^{-3}). The two-layer LambdaRank models were trained for 500 iterations using random restarts (a restart occurs when progress on the validation set slows sufficiently). For the LambdaMART models, we swept through learning rates taking values in $\{0.01, 0.05, 0.1, 0.3, 0.5\}$ and we swept through numbers of leaves from 50 to 400 in steps of 30. The LambdaMART models were trained for 3000 boosting iterations, where typically the number of trees kept was within 200 of the maximum (3000). We used 10-fold cross validation on the provided training data to determine the best parameters for both LambdaRank and LambdaMART.

4.3. Combining Rankers

As mentioned, we split the data into 10 equal parts, by query, after shuffling. Here we will refer to the datasets by their components: for example, 01_06 refers to the data constructed by concatenating the first 6 of the 10 splits together.

We ran two sets of experiments. In the first, we used 01_06 for training, 07_08 to compute the optimal combiner parameters, and 09_10 for test. For the individual models, in order to get an apples-to-apples comparison with the ensembles, we trained on 01_06 and tested on 09_10. This initial set of experiments, which used only 6 of the final set of 12 models we trained,[3] showed us that simply averaging model scores, after first performing an overall normalization to unit variance, gave as good results as any other combination method. In light of this, to get more accurate estimates of test accuracy, we used 07_10 for comparing ensembles. In the second set of experiments we retrained each model on 01_10, using 10-fold cross validation to determine the number of trees for the boosted tree models. For a given model, the resulting 10 sets of test scores (on the blind test set) were simply averaged. These sets of averaged, blind test scores were then combined using the best combination technique (simple averaging of normalized scores) and the best subset of models to combine, as found in the first set of experiments. In this way, we effectively trained on all available training data, despite having used some training data to find the best combination method and to find other parameters.

The first 6 rows of Table 1 list the ERR and NDCG accuracies on 09_10 for the set of 6 individual models we used to find the best combination technique. In Table 1, "optimal combiner" refers to the method for finding the optimal linear combination of a pair of

3. The others weren't ready yet!

Table 1: ERR and NDCG accuracies for various combination methods. Accuracies are reported on 09_10.

Model	ERR	NDCG
1. LambdaMART optimized for ERR	0.455	0.774
2. LambdaMART optimized for ERR trained on Aug70	0.456	0.781
3. LambdaMART optimized for NDCG	0.455	0.778
4. LambdaMART optimized for NDCG trained on Aug70	0.457	0.784
5. LambdaRank optimized for ERR	0.447	0.749
6. LambdaRank optimized for NDCG	0.447	0.757
Combine all 6 models, normalized scores, with weight 1	0.459	0.783
Combine Tree 1, with optimal combiner	0.458	0.784
Combine Tree 2, with optimal combiner	0.458	0.785
Combine all 6 models with Linear LambdaRank (NDCG)	0.457	0.785
Combine all 6 models with Linear LambdaRank (ERR)	0.457	0.785
Combine all 6 models with 2-layer LambdaRank (NDCG)	0.458	0.785
Combine all 6 models with 2-layer LambdaRank (ERR)	0.458	0.785
Combine models 2 and 4 with weight 1	0.457	0.785
Combine models 1, 2, 3 and 4 with weight 1	0.458	0.784

rankers (for any IR measure) described in (Wu et al., 2009; Burges, 2010). The optimal combiner is most easily applied to a pair of rankers at a time, so Trees 1 and 2 in Table 1 refer to a binary combination sequence: Tree 1 is the combination of models (((13)(24))(56)) (where the integers refer to the model indexes given in the first 6 rows of the table, and where the parentheses denote the order of combination). Thus Tree 1 combines pairs of algorithms whose only difference is the measure they are optimized for (ERR or NDCG). Tree 2 is (((45)(16))(23)), and combines the least algorithmically similar models first.

We concluded from Table 1 that the more sophisticated combination techniques provided no clear advantage over simple averaging, and so in honor of Occam, we chose the latter. This freed us up to use all of 07_10 as a test set to identify the best subset of models to form the ensemble. In order to do this, we first computed the single model performance on 07_10.[4] The results are shown in Table 2.

Table 3 shows the ensembles we tested using weight 1 combinations (after score normalization). We wrote a tool to combine any given set of scores in this way, and to output final NDCG and ERR accuracies. The results are reported on 07_10. From this table, we concluded that the best combination was simply an average of normalized model scores, over all 12 models, and this is the model we submitted as our final model. Specifically, all of the final models were trained on the whole training set, where the MART and LambdaRank models were trained using the fixed parameters found during the sweeps, and using 1/10 of the data for validation; on the other hand each LambdaMART model was averaged over 10 trained models, where each of the 10 were trained using a randomly chosen validation set of

4. By now, all 12 models were ready.

Table 2: The 12 models used in the final ensemble, with corresponding ERR and NDCG accuracies. All models were trained on 01_06 (see text), and accuracies are reported on 07_10.

Model	Description	ERR	NDCG
M1	LambdaMART optimized for ERR	0.461	0.774
M2	LambdaMART optimized for ERR trained on Aug50	0.464	0.786
M3	LambdaMART optimized for ERR trained on Aug70	0.462	0.780
M4	LambdaMART trained for ERR with MART scores as features	0.460	0.775
M5	LambdaMART optimized for NDCG	0.462	0.779
M6	LambdaMART optimized for NDCG trained on Aug50	0.464	0.787
M7	LambdaMART optimized for NDCG trained on Aug70	0.463	0.783
M8	LambdaMART trained for NDCG with MART scores as features	0.461	0.781
M9	LambdaRank optimized for ERR	0.453	0.750
M10	LambdaRank optimized for NDCG	0.453	0.757
M11	MART	0.455	0.772
M12	MART with output scores normalized to unit variance per query	0.455	0.772

Table 3: ERR and NDCG accuracies for ensembles of models trained on 01_06 and combined with simple averaging. Results are reported on 07_10.

Ensemble	ERR	NDCG
All: M1–M12	0.4657	0.7878
All minus MARTs: M1–M10	0.4657	0.7875
LambdaMART only: M1–M8	0.4652	0.7874
All minus LambdaRanks: M1–M8, M11, M12	0.4653	0.7879
Augmented only: M2, M3, M6, M7	0.4641	0.7864
Non-augmented: M1, M4, M5, M8–M12	0.4648	0.7850
Just ERR: M1–M3, M4, M9	0.4657	0.7860
Just NDCG: M5–M8, M10	0.4652	0.7869

1/10 of the training data; and the scores of each of the resulting 12 models were scaled to unit variance.[5] Table 4 contains the final results on the held-out validation and test data.

5. Future Work

We end by mentioning three intriguing questions. First, ensembles are clearly a powerful way to achieve state-of-the-art accuracies. Why is it that we do not seem able to train a single model to achieve the same accuracies, directly? Note that a large ensemble can itself be modeled by training a much smaller model using arbitrarily large amounts of unlabeled

5. In fact, unit variance and zero mean, although the translation to zero mean will not affect the ranking results.

Table 4: ERR and NDCG accuracies on the final validation and test data.

Rank	Team	Test ERR	Test NDCG	Valid ERR	Valid NDCG
1	Ca3Si2O7	0.468605	0.8041	0.4611	0.7995
2	catonakeyboardinspace	0.467857	0.8060	0.4609	0.8011
3	MLG	0.466954	0.8026	0.4600	0.7960
4	Joker	0.466776	0.8053	0.4607	0.8011
5	AG	0.466157	0.8018	0.4606	0.8010

data to give almost the same accuracy (Bucila et al., 2006), so it is not the case that such single models simply cannot be found. Relatedly, how does the diversity of the trained models contribute to the overall increase in accuracy? Simply training a variety of models and then combining them is a powerful technique, but such approaches are in need of more principled foundations to guide the search for the best ensemble. Second, in previous work, we showed that IR measures such as NDCG, MAP, and MRR can be optimized directly using lambda-gradient models, but that test accuracy on a given measure is not always highest when that model was trained on that measure (Donmez et al., 2009). For some pairs of IR measures, the train/test measures appear to match only if a sufficiently large training set is used, and we have found a similar effect here (the ERR test accuracies were similar, whether training on NDCG or on ERR). Why this occurs is an open question: it could be because the measure simply extracts more information from the training data (for example, pair-based lambda gradients such as those used for NDCG, versus absolute rank-based lambda gradients, will provide many more instances to learn from); or it could be a regularization effect. Finally, we note that although our system achieved the highest test ERR score, it did not achieve the highest test NDCG. To the best of our knowledge, we were the only team that built models that directly learned ERR, and this result is an indication that such training was effective.

Acknowledgments

We thank our other team members, Ofer Dekel and John Platt: Ofer for providing the highly optimized version of the LambdaMART code that we used, and John for a discussion leading to the data augmentation idea. We would also like to thank the organizers of the Yahoo! Learning to Rank Challenge for their work in running the competition.

References

C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proc. Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

C.J.C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with non-smooth cost functions. In *Advances in Neural Information Processing Systems 18*, 2006.

O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *JMLR Workshop and Conference Proceedings*, 14:1–24, 2011.

O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *International Conference on Information and Knowledge Management (CIKM)*, 2009.

P. Donmez, K. Svore, and C.J.C. Burges. On the local optimality of lambdarank. In *Special Interest Group on Information Retrieval (SIGIR)*, 2009.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 25(5):1189–1232, 2001.

K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 41–48, 2000.

P. Li, C.J.C. Burges, and Q. Wu. Learning to rank using classification and gradient boosting. In *Advances in Neural Information Processing Systems 19*, 2007.

Q. Wu, C.J.C. Burges, K. Svore, and J. Gao. Adapting Boosting for Information Retrieval Measures. *Information Retrieval*, 2009.