Speeding up Document Ranking with Rank-based Features

Claudio Lucchese
ISTI–CNR, Pisa, Italy
c.lucchese@isti.cnr.it

Franco Maria Nardini
ISTI–CNR, Pisa, Italy
f.nardini@isti.cnr.it

Salvatore Orlando
Univ. di Venezia, Italy
orlando@unive.it

Raffaele Perego
ISTI–CNR, Pisa, Italy
r.perego@isti.cnr.it

Nicola Tonellotto
ISTI–CNR, Pisa, Italy
n.tonellotto@isti.cnr.it

ABSTRACT

Learning to Rank (LtR) is an effective machine learning methodology for inducing high-quality document ranking functions. Given a query and a candidate set of documents, where query-document pairs are represented by feature vectors, a machine-learned function is used to reorder this set. In this paper we propose a new family of *rank-based* features, which extend the original feature vector associated with each query-document pair. Indeed, since they are derived as a function of the query-document pair and the full set of candidate documents to score, rank-based features provide additional information to better rank documents and return the most relevant ones. We report a comprehensive evaluation showing that rank-based features allow us to achieve the desired effectiveness with ranking models being up to 3.5 times smaller than models not using them, with a scoring time reduction up to 70%.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

Keywords
Learning to Rank; Efficiency; Meta-Features

1. INTRODUCTION

The problem of ranking documents in response to a user's query is particularly challenging in Web search. Hence most of the search engines exploit document ranking pipelines based on *Learning-to-Rank* (LtR) techniques [4, 9]. In a LtR framework, a machine learning algorithm is used to derive a model from a training set of labeled documents [12]. Search engines usually exploit LtR models within a two-stage ranking architecture: (i) candidate retrieval and (ii) candidate re-ranking. The first stage retrieves from the inverted index a set $\mathcal{C}_q$ of (possibly relevant) candidate documents matching a user query q, where $|\mathcal{C}_q| \gg k$ and k is the number of relevant url's to include in the return page. This preliminary filtering is usually achieved by a simple and fast base ranker, e.g., BM25 plus some document features. In the second stage, the LtR model is used to score and re-rank the documents in $\mathcal{C}_q$. Finally, the top k documents in $\mathcal{C}_q$ are returned to the user.

In this scenario, the efficiency of the second LtR-based stage is crucial. It clearly impacts on the response time of the system. In addition, a more efficient ranker allows us to score more candidate documents: this is a very important property, since when larger sets of candidate documents are retrieved during the first step, we observe a better recall and a better overall quality of the final result list [10].

The efficiency of machine-learned rankers have recently attracted increasing interest, since state-of-the-art LtR algorithms induce models that are indeed very complex and expensive at scoring time, and may become a bottleneck in a two-stage ranking pipeline [4, 14, 16, 15, 1, 2]. Most of the work published so far focused primarily on the optimization techniques that can be adopted to make the second-stage ranker faster. Our approach is orthogonal to such optimization techniques. Indeed, we address the problem of improving the efficiency of a ranking model by *extending* the feature set used to represent a query-document pair with additional *rank-based features*. Since they are a function of the query-document pair and the full set of candidate documents $\mathcal{C}_q$, rank-based features provide additional information to better score documents. We show that these rank-based features, when used to extend the original feature vectors, are able to improve the learning process and achieve the desired ranking quality with learned models of reduced complexity.

Specifically, we propose to extend the feature set associated with each document in $\mathcal{C}_q$ as follows. For each original feature considered, we add four *rank-based* variants, named Dist-Min, Dist-Max, Rank, and Rev-Rank. The goal of rank-based features is to provide additional information about some ordering properties of a document compared with the others in $\mathcal{C}_q$. For example, given a feature $f \in \mathcal{F}$, where $\mathcal{F}$ is the feature set characterizing a generic document $c \in \mathcal{C}_q$, its Rank_f variant is equal to the *rank* of c in $\mathcal{C}_q$, where $\mathcal{C}_q$ is ordered in decreasing order of f. Indeed, each rank-based feature turns out to be a function of the query and the full candidate results set, rather than just of a query-document pair, thus providing a richer information.

We focus on the two state-of-the-art rankers, i.e., Gradient-Boosted Regression Trees (GBRT) [7] and λ-MART [3], as they have been proved to be the most effective in the recent Yahoo! learning to rank challenge [5]. We experimentally show that the use of *rank-based* features can reduce the number of regression trees generated by gradient boosting

SIGIR '15, August 09 - 13, 2015, Santiago, Chile

ACM 978-1-4503-3621-5/15/08 $15.00
DOI: http://dx.doi.org/10.1145/2766462.2767776.

Table 1: A small example of golden set for learning to rank.

q_1			q_2			q_3		
rel	BM25	PR	*rel*	BM25	PR	*rel*	BM25	PR
1	.80	.20	1	.60	.50	1	.65	.45
1	.75	.15	1	.60	.47	1	.67	.40
0	.65	.05	1	.50	.45	0	.60	.35
0	.65	.05	0	.45	.40	0	.40	.15

approaches up to 3.5 times and they provide a scoring time reduction up to 70%.

2. RANK-BASED FEATURES

To introduce the new set of *rank-based* features, let us consider the example in Table 1. The table illustrates a small training set of query-document feature vectors. It is made up of three queries with four candidate results each. For each document associated with a query, a binary relevance label *rel* and two well-known features – BM25 and PageRank (PR) – are listed. During learning, a tree-based algorithm should find the rules that best separate relevant from irrelevant results. A simple decision stump – i.e., a tree with one node and two leaves – is not sufficient in this case, since a "minimal" classification tree with perfect accuracy is the following:

```
if BM25 ≥ .75 then 1 else
    if PR ≥ .45 then 1 else
        if BM25 ≥ .67 then 1 else 0
```

Simpler but still effective trees can be obtained if we enrich the feature set associated with each pair (q, c), $c \in \mathcal{C}_q$, with new *rank-based* features. In our toy example, an optimal decision tree that achieves perfect accuracy is the one that classifies as relevant the documents with a PR score being at most 0.05 less than the best PR score in the same candidate set, that is:

```
if Dist-Max_PR ≤ 0.05 then 1 else 0
```

where $\mathsf{Dist\text{-}Max}_{\mathrm{PR}}$ measures the difference between each value of PR and the largest value assumed by PR in $\mathcal{C}_q$. This last classifier does not improve the quality of the first one. On the other hand, it is much simpler and its evaluation requires much less time if *rank-based* features are available.

We propose four construction strategies to build new *rank-based features* for a given feature $f \in \mathcal{F}$, occurring in the feature vector representing each pair (q, c), where $c \in \mathcal{C}_q$:

- Rank_f. Feature $\mathsf{Rank}_f \in \{1, 2, \ldots, |\mathcal{C}_q|\}$ corresponds to the *rank* of c after sorting $\mathcal{C}_q$ in *descending order* of f.
- $\mathsf{Rev\text{-}Rank}_f$. The same as Rank_f, but $\mathcal{C}_q$ is ordered in *ascending order* of f.
- $\mathsf{Dist\text{-}Min}_f$. Feature $\mathsf{Dist\text{-}Min}_f \in \mathbb{R}$ is given by the absolute value of the difference between the value of f and the *smallest value* of f in $\mathcal{C}_q$.
- $\mathsf{Dist\text{-}Max}_f$. The same as $\mathsf{Dist\text{-}Min}_f$, but we consider the difference with the *largest value* of f in $\mathcal{C}_q$.

We expect that these new features can improve the scoring ability of the learned models since they contribute to give information about the whole set of candidates C_q, while other features give information limited to the single pairs (q, c) with respect to the entire collection of indexed documents. We claim that they can capture *relatively better* or *relatively worse* concepts over the current candidate set. It is worth noting that Rank_f and $\mathsf{Rev\text{-}Rank}_f$ are not mutually exclusive. If higher values of f are better, the former could promote good documents, while the latter should demote bad documents. Moreover, the proposed *rank-based features* are orthogonal to other meta-level features and query-level normalization techniques, as some *rank-based features* can be built on top of normalized ones. We also compared *rank-based features* against query-level z-score normalization, with the latter showing very poor performance. Due to space constraints, these experiments are not reported.

Rank-based feature generation. It is possible to apply the four *rank-based* feature construction strategies to every feature $f \in \mathcal{F}$. This would multiply by five the size of the input dataset, making the learning process very expensive. In order to save space and time, we apply our feature construction methodology to a limited subset of the original features, using this subset to generate the new ranked-based ones. Specifically, we adopt a greedy approach exploiting a feature evaluation technique to choose which subset of features to enrich with the new rank-based ones:

1. We train a ranker on a given training dataset (golden set) by exploiting the full feature set $\mathcal{F}$.
2. We run a feature ranking method, aiming at identifying the 10 best performing features.
3. We create a new dataset, where the original features $\mathcal{F}$ are accompanied by 40 additional features, generated by applying the four *rank-based* construction strategies to the best 10 features. The resulting feature set is denoted by $\mathcal{F}^{+40}$.
4. A new ranker is trained on the new data, and the $\mathcal{F}^{+40}$ feature are evaluated and ranked as above.
5. Finally, we create an smaller dataset that contains all the original features $\mathcal{F}$, but only the best 10 *rank-based* features found in the previous step. The resulting dataset is denoted by $\mathcal{F}^{+10}$.

We show that the rankers derived using $\mathcal{F}^{+40}$ or $\mathcal{F}^{+10}$ can improve over the ranker built on the original feature set $\mathcal{F}$.

As previously stated, we focus on two state-of-the-art tree-based ranking algorithms: namely GBRT [7] and λ-MART [3]. Both generate large forests of additive regression trees. Our objective is thus to exploit the feature sets $\mathcal{F}^{+40}$ and $\mathcal{F}^{+10}$ to reduce the number of trees without hindering the ranking quality. Feature importance in these rankers can easily be estimated as described in the following section.

Feature importance evaluation. Several approaches have been proposed to evaluate the goodness of a feature in a ranking model [17, 11, 8]. Since both GBRT and λ-MART rankers are based on boosted regression trees, we borrow a method for feature evaluation from the original work on GBRTs by J. Friedman [7]. During the construction of each tree t in set of trees of the GBRT model T, for each feature we compute a measure similar to the *least square improvement measure* proposed in [7]. Since each tree split node improves the objective function, the *gain* g_i for each $f_i \in \mathcal{F}$ can be estimated by summing up the gains across all the split nodes $n \in t$ for all the trees $t \in T$ where feature f_i is used. We thus have:

$$g_i = \sum_{t \in T} \sum_{n \in t} \hat{i}_n^2 \, 1(v_n = f_i)$$

where v_n is the splitting feature used in node n, 1() is the indicator function, and $\hat{i}_n^2$ is the empirical improvement in

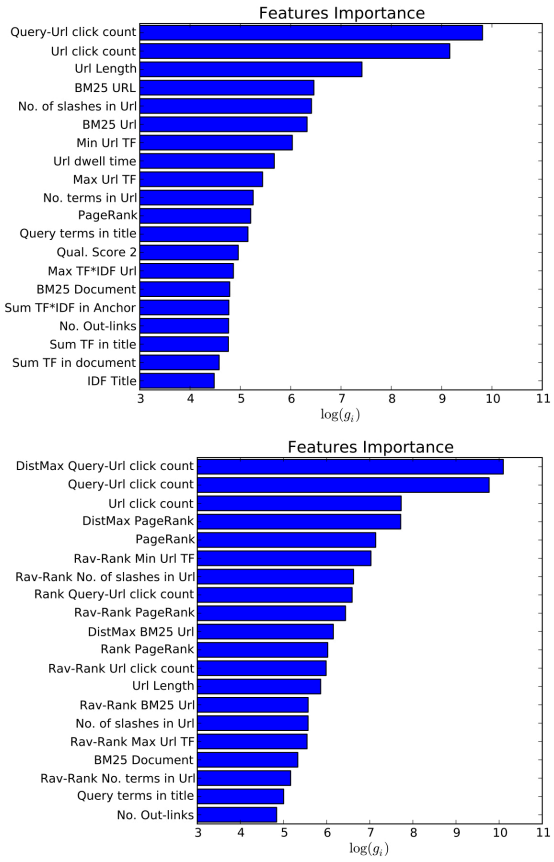

Figure 1: Features importance averaged over 5 folds of the MSN dataset: $\mathcal{F}$ (above) and $\mathcal{F}^{+40}$ (below).

the squared error as a result of the split. More specifically, according to [7], we have:

$$\hat{i}_n^2 = \frac{n_l n_r}{n_l + n_r} (\overline{y}_l - \overline{y}_r)^2$$

where n_l (n_r) is the number of instances in the left (right) child of the splitting node n, and $\overline{y}_l$ ($\overline{y}_r$) is the mean value assumed by the relevance label in the left (right) child of n.

Note that g_i measures exactly the improvement provided by f_i on the basis of the squared error reduction that was observed in the training set.

3. EXPERIMENTS

In our experiments we use the MSN learning to rank[1] datasets. It consists of vectors of 136 features extracted from query-url pairs. This dataset is partitioned into five subsets for five-fold cross validation. For each fold, we employ the validation set to fine-tune the number of trees of the generated models to avoid overfitting. In the following, we denote these folds by MSN/Fold i, $i \in \{1, 2, 3, 4, 5\}$. The MSN dataset provides relevance judgment labels ranging from 0 (irrelevant) to 4 (perfectly relevant). We use RankLib[2], an open-source implementation of the GBRT and λ-MART algorithms to learn our models. We then evaluate

[1] `http://research.microsoft.com/en-us/projects/mslr/`
[2] `http://sourceforge.net/p/lemur/wiki/RankLib/`

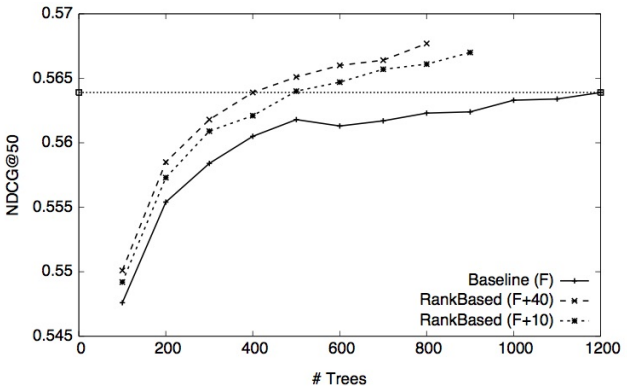

Figure 2: Effectiveness with features sets $\mathcal{F}^{+40}$, $\mathcal{F}^{+10}$, and $\mathcal{F}$ on MSN/Fold 1.

the effectiveness of the ranking models by using NDCG@k, $k \in \{10, 25, 50, 100\}$.

Assessment of feature importance. Feature importance is measured by gain g_i as defined above. We report results regarding the MSN dataset, averaged over 5 folds, on which we trained λ-MART rankers by optimizing NDCG@50. The results achieved at different NDCG cut-offs show a similar trend. Figure 1 plots the importance of the top-20 features according to gain g_i (log scale) for λ-MART trained on $\mathcal{F}^{+40}$. Our experiments confirm that a few features contribute to the largest part of the cost function optimization. This motivated our choice of applying the proposed feature construction strategies to the best 10 features of $\mathcal{F}$ only, thus obtaining the new feature set $\mathcal{F}^{+40}$. Figure 1 shows that the most important feature in $\mathcal{F}^{+40}$ is a rank-based one, and that 12 out of the top-20 features are rank-based.

We also notice that in most cases the rank-based features become more important than the original one. Consider for example feature f = PageRank: it was the 11^{th} most important feature in $\mathcal{F}$, but its variant Dist-Max$_f$ is the 4^{th} and precedes in $\mathcal{F}^{+40}$ the original feature f.

We conclude that GBRT and λ-MART models largely use the new rank-based features as they were considered more informative than others. We also find out that rank-based features are not a replacement of original features, but they rather provide complementary information.

Efficiency of learned models. Figure 2 illustrates the quality in terms of NDCG of the λ-MART models trained on MSN/Fold 1 by using $\mathcal{F}$, $\mathcal{F}^{+40}$, and $\mathcal{F}^{+10}$. We report the values of NDCG@50 obtained on the test set of MSN/Fold 1 as a function of the number of trees of the generated model. Note that the three curves stops in correspondence of a different number of trees: this is because the validation set is used to choose the number of trees in the final model.

The benefit of our rank-based features shows up very early in the learning process. First, we note that the new feature sets $\mathcal{F}^{+40}$ and $\mathcal{F}^{+10}$ always produce models that are more effective than the one obtained by using the original feature set $\mathcal{F}$ at smaller ensemble sizes. More importantly, the maximum quality of the model trained on $\mathcal{F}$ requires a number of trees that is about three times larger than the number of trees of the models trained on either $\mathcal{F}^{+40}$ or $\mathcal{F}^{+10}$ to obtain the same NDCG@50. This behaviour is very significant, since the number of trees directly impacts ranking

Table 2: Scoring time performance at the best NDCG@50 provided by $\mathcal{F}$.

Dataset	Features	λ-MART # Trees	λ-MART Time μs.	λ-MART Δ Time	λ-MART NDCG@50	GBRT # Trees	GBRT Time μs.	GBRT Δ Time	GBRT NDCG@50
MSN / Fold 1	$\mathcal{F}$	1163	29.337		0.5640	833	21.094		0.5581
	$\mathcal{F}^{+40}$	500	14.529	-50%	0.5651	300	9.533	-54%	0.5605
	$\mathcal{F}^{+10}$	500	13.464	-54%	0.5640	300	8.303	-60%	0.5608
MSN / Fold 2	$\mathcal{F}$	898	22.717		0.5645	708	17.971		0.561
	$\mathcal{F}^{+40}$	400	12.031	-47%	0.5656	400	12.031	-33%	0.5623
	$\mathcal{F}^{+10}$	400	11.468	-49%	0.5652	300	8.303	-53%	0.5614
MSN / Fold 3	$\mathcal{F}$	750	19.020		0.5659	915	23.142		0.5623
	$\mathcal{F}^{+40}$	400	12.031	-36%	0.5678	300	9.533	-58%	0.5646
	$\mathcal{F}^{+10}$	400	10.634	-44%	0.5675	300	8.303	-64%	0.5637
MSN / Fold 4	$\mathcal{F}$	1420	35.757		0.5648	942	23.816		0.5613
	$\mathcal{F}^{+40}$	400	12.031	-66%	0.5657	400	12.031	-49%	0.5629
	$\mathcal{F}^{+10}$	400	10.467	-70%	0.5657	300	8.469	-64%	0.5613
MSN / Fold 5	$\mathcal{F}$	581	14.799		0.5705	990	25.015		0.5696
	$\mathcal{F}^{+40}$	300	9.533	-35%	0.5715	400	12.031	-51%	0.5703
	$\mathcal{F}^{+10}$	300	8.304	-43%	0.5715	400	10.967	-56%	0.5717

efficiency. Document scoring requires in fact the traversal of all the trees of the model, and its cost is thus linearly proportional to their number.

In order to evaluate the benefits of the proposed models at scoring time, we have to consider both the time needed to generate the rank-based features, and the time for traversing the forest of trees. In fact, the use of rank-based features should reduce the number of trees at least as much as to cover the cost of generating them.

Table 2 reports the scoring time of the best GBRT and λ-MART model in terms of NDCG@50 built with $\mathcal{F}$. We also measured the quality of the models exploiting rank-based features by adding 100 trees at a time, and we selected the smallest forest of trees achieving at least the same quality as the best model build using the original feature set. For such model, we also measured and reported the per-document scoring time, which *includes* the feature generation time.

It is apparent that the rank-based features allows to achieve the same or better performance than using $\mathcal{F}$, with a much smaller number of trees, thus resulting in a significantly smaller scoring time. We observe that the scoring time can be reduced up to a 70% factor. The time needed to extract the rank-based features is thus rewarded with a strongly reduced scoring time. For the sake of completeness, we measured the quality of the complete forests (results are not reported here) and we observed a statistically-significant improvement in NDCG with both proposed features sets with a p-value always smaller than 0.01 under randomization test with $100,000$ permutations [13].

4. CONCLUSIONS AND FUTURE WORK

We proposed a novel feature construction strategy to enrich a feature set with new *rank-based* features in a learning-to-rank framework. The exploitation of *rank-based* features allows us to generate significantly faster rankers, i.e., with less trees, without any quality loss. Experiments proved that our proposed features can reduce the number of regression trees generated by gradient boosting approaches up to 3.5 times and they provide a speed-up in the scoring time up to 70%. In addition, results shown that *rank-based* features can provide statistically-significant improvements in the effectiveness of the generated ranking models.

We envision several open research directions. We apply the rank-based feature construction only to a subset of the whole set $\mathcal{F}$. We intend to study the impact of feature selection techniques for rank-based feature enhancement. Moreover, we will investigate the impact of our rank-based features on other, non tree-based (LtR) algorithms. We also intend to study the performance of a multi-stage ranking pipeline [6, 4] exploiting our rank-based features.

Acknowledgements. We acknowledge the support of Tiscali S.p.A. In particular, we wish to warmly thank Domenico Dato and Monica Mori (Istella) for fruitful discussions and valuable feedbacks helping us in concretizing this paper.

5. REFERENCES

[1] N. Asadi and J. Lin. Training efficient tree-based models for document ranking. In *Proc. ECIR*, pages 146–157, 2013.

[2] N. Asadi, J. Lin, and A. P. de Vries. Runtime optimizations for prediction with tree-based models. *IEEE TKDE*, 99:1, 2013.

[3] C. J. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. *Journal of Machine Learning*, 14:25–35, 2011.

[4] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proc. WSDM*, pages 411–420. ACM, 2010.

[5] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *J. of Machine Learning Research*, 14:1–24, 2011.

[6] V. Dang, M. Bendersky, and W. B. Croft. Two-stage learning to rank for information retrieval. In *Advances in Information Retrieval*, pages 423–434. Springer, 2013.

[7] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[8] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature selection for ranking. In *Proc. SIGIR'07*, ACM.

[9] T.-Y. Liu. Learning to rank for information retrieval. *Found. and Trends in Information Retrieval*, 3(3):225–331, 2009.

[10] C. Macdonald, R. L. Santos, and I. Ounis. The whens and hows of learning to rank for web search. *Information Retrieval*, 16(5):584–628, 2013.

[11] F. Pan, T. Converse, D. Ahn, F. Salvetti, and G. Donato. Feature selection for ranking using boosted trees. In *Proc. CIKM'09*, pages 2025–2028, ACM.

[12] I. Segalovich. Machine learning in search quality at yandex. *Invited Talk, SIGIR*, 2010.

[13] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proc. CIKM '07*, pages 623–632. ACM, 2007.

[14] L. Wang, J. J. Lin, and D. Metzler. Learning to efficiently rank. In *Proc. SIGIR*, pages 138–145. ACM, 2010.

[15] L. Wang, J. J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. SIGIR*, pages 105–114. ACM, 2011.

[16] L. Wang, D. Metzler, and J. J. Lin. Ranking under temporal constraints. In *Proc. CIKM*, pages 79–88. ACM, 2010.

[17] L. Yu, H. Liu, and I. Guyon. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224, 2004.