LambdaMART learning-to-rank using different sets of features.

Abhiram Eswaran*
College of Information and Computer Sciences
University of Massachusetts, Amherst
MA, United States of America
aeswaran@umass.edu

Shivani Gupta
College of Information and Computer Sciences
University of Massachusetts, Amherst
MA, United States of America
shivanigupta@umass.edu

ABSTRACT

Learning to rank for Information Retrieval (IR) is a task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance. The objective of this report is to compare the effectiveness of Learning-to-rank algorithms especially LamdaMART with traditional baselines like RM1, RM3, DL2, PL2 and Cluster Based Retrieval Model. We do this empirical evaluation on 3 data sets Robust04, Trec1-3 and WT10G. Finally we show that the biggest advantage of learning to rank is the ability to combine new features with ease.

CCS Concepts

•**Information Retrieval** → *Learning to Rank;*

Keywords

Empirical Study

1. INTRODUCTION

Information retrieval is a mature technology. Through commercial search engines, anyone can query billions of web pages in milliseconds for free. But ranking these results is the most critical aspect when it comes to retrieving results from billions of possible result for a query. In this report we discuss one of the famous ranking functions, LambdaMART.

Designing effective ranking functions for free text retrieval has proved notoriously difficult. Retrieval functions designed for one collection and application often do not work well on other collections without additional time consuming modifications. This has led to interest in using methods for automatically learning ranked retrieval functions. We call these methods, which learn how to combine multiple features by means of discriminative learning as "learning to rank" methods.

*Purely alphabetical order.

DOI: 10.475/123_4

The Learning to Rank Model is a feature based approach. All the documents under investigation are represented by multiple features reflecting the relevance of the documents to the query. Typical features used in learning to rank include the frequencies of the query terms in the document, the BM25 and Query Language Model scores with different smoothing. We discuss more about the features in Section 3.

The capability of combining a large number of features is a very important advantage of learning to rank. It is easy to incorporate any new progress on the retrieval model by including the output of the model as one dimension of the features. For example, topical features can be easily added with contextual features to include user behaviour into the ranking model. [7] Such a capability is highly demanding for real search engines, since it is almost impossible to use only a few factors to satisfy complex information needs of Web users.

The rest of the report is organized as follows. We discuss the related work in learning to rank for information retrieval in section 2. A discussion of the features that we used in our experiments and methods of data preparation are presented in section 3. We then describe, in section 4, the experimental methods. Empirical results are discussed in section 5. Section 6 gives the final conclusion of the experiments.

2. RELATED WORK

The concept of Learning To Rank came from the need for commercial search engines to incorporate multiple features for ranking. Few such features include: 1. Log frequency of query word in anchor text. 2. Query word in color on page. 3. Number of images on page. 4. Number of external links on page. 5. PageRank of the page. 6. URL length. 7. Page length. As of 2008, Google was said to be using 200 such features. [10]

Most work in learning to rank using machine learning models is divided into 3 broad categories. We discuss the 3 approaches and their related work below.

Point Wise Approach: The most straightforward way is to check whether existing learning methods can be directly applied. The function is based on features of a single object. And hence we predict the exact relevance degree of each document. The polynomial regression function by Fuhr[6], McRank using multi-class classification to solve the problem of ranking by Li [9] and PRanking[5] on ordinal regression are few famous work in Point Wise Ranking.

Pair Wise Ranking: The pairwise approach does not focus on accurately predicting the relevance degree of each

document. Instead, it cares about the relative order between two documents. One good example of this approach would be predicting partial ranking given two documents. This can be thought of as a classification problem where the problem is to identify the non-relevant documents that are classified as relevant. RankNet[11] which uses the approach of List Wise Ranking is one of the first learning-to-rank algorithm used by commercial search engines.

List Wise Ranking:In the list wise approach, each query is represented by a group of documents and their ground truth labels. The disadvantages of List Wise Ranking are that it is expensive and also has no advantage over the much used Pair Wise Ranking.

3. FEATURES AND DATASETS

Each document is represented by a 125-dimensional vector as proposed by [3]. The details of the individual features are given in Table 1. Each feature group of 5 consists of body, anchor, url, title and the whole document.

3.1 Dataset Descriptions

The datasets are lucene indices for various collections like TREC1-3, Robust04 and WT10G. The scores of all feature vectors are computed from the datasets along with relevance judgment labels:

(1) The relevance judgments are obtained from available qrel file provided by TREC, which take 2 values between 0 (irrelevant) and 1(relevant).

(2) The features are basically extracted by us, and are those widely used in the research community.

In the data files, each row corresponds to a query-document pair. The first column is relevance label of the pair, the second column is query id, and the following columns are features. A query-document pair is represented by a 125-dimensional feature vector. Below are 2 lines from TREC 1-3 data file which we computed from the data set.

```
==================================
0 qid:301 1:0 2:0 3:0 4:0 ... 124:0 125:-5.099
1 qid:301 1:0 2:3 3:0 4:0 ... 124:0 125:-5.573
==================================
```

3.2 Dataset Partition

We used five-fold cross validation with each data set. In each fold, we used three parts for training, one part for validation, and the remaining part for test. The training set is used to learn ranking models. The validation set is used to tune the hyper parameters of the learning algorithms, such as the number of leaf and tree values for LamdaMART. The test set is used to evaluate the performance of the learned ranking models. We also partitioned our queries into training, validate and test and used them correspondingly with the data sets with 50% for training, 25% for validation and 25%for testing.

3.3 Data Extraction

Data set WT10G is composed of HTML pages from the web. To extract and compute feature values we utilized available HTML web scraper in Java. We used Jsoup[2] to assist our experiments. This allowed us to extract different parts of the HTML like body, title, url etc from the html files. We then computed the feature scores after extracting the text from each required fields.

4. EXPERIMENTAL METHODS

We are using Query Language with Dirichlet Smoothing, Divergence from Randomness Models (DL2, PL2), CBDM as our baseline methods.

4.1 Query Likelihood Model with Dirichlet Smoothing

In query likelihood language modelling, we estimate a language model M_d for every document D in the collection and then rank each document by the probability of "generating" the query.

$$P(q \mid M_d) = \prod_{t \varepsilon q} P(t \mid D)$$

Using Maximum Likelihood Estimate,

$$P_{MLE}(q \mid M_d) = c(t,d) \div \mid D \mid$$

But here unseen words gets zero probability and if any one query term doesn't exist in the document, it gets a zero probability for that query. To solve this, we can apply smoothing where unseen words get non-zero probability by interpolating it from an MLE model of the whole corpus. One such smoothing method is Dirichlet Smoothing, in which smoothing depends on the document size. Longer documents provide better estimates and thus its own MLE is more reliable.

$$P(t \mid \theta_D) = \frac{c(t,d) + \mu * P(t \mid Corpus)}{\mid D \mid + \mu}$$

MLE weight: $1 - \lambda = \frac{|D|}{|D|+\mu}$ Corpus Weight: $\lambda = \frac{\mu}{|D|+\mu}$

4.2 Divergence From Randomness

The Divergence from Randomness (DFR) [4] paradigm is a generalisation of one of the very first models of Information Retrieval, Harter's 2-Poisson indexing-model. These models are based on the simple idea: "The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in the document d". In other words the term-weight is inversely related to the probability of term-frequency within the document d obtained by a model M of randomness:

$$weight(t \mid d) \propto -logProb_M(t \varepsilon d \mid collection)$$

where the subscript M stands for the type of model of randomness employed to compute the probability.

4.2.1 PL2

Here, we take Poisson's distribution as the model of randomness and do two normalizations on top of that. First normalization is regarding the amount of information gained by a term. Here we use Laplace L model to for computing the information gqin for a term in the document. Second normalization for is for term frequency. The document length is normalized to a standard length and hence term frequency is also recomputed with respect to the standard length. Hence,

$$weight(t \mid d) = Inf_(1) * Inf_(2)$$

where,

$$Inf_1(tf) = -log_2[{}^{F}_{tf} p^{tf} q^{F-tf}]$$

$$Inf_2(tf) = \frac{1}{tf+1}$$

Table 1: List of Learning Feature

Feature Ids	Feature Description
1-5	Covered query term number
6-10	Covered query term ratio
11-15	Stream Length
16-20	IDF(Inverse document frequency)
21-25	Sum of term frequency
26-30	Min of term frequency
31-35	Max of term frequency
36-40	Mean of term frequency
41-45	Variance of term frequency
46-50	Sum of stream length normalized term frequency
51-55	Min of stream length normalized term frequency
56-60	Max of stream length normalized term frequency
61-65	Mean of stream length normalized term frequency
66-70	Variance of stream length normalized term frequency
71-75	Sum of tf * idf
76-80	Min of tf * idf
81-85	Max of tf * idf
86-90	Mean of tf * idf
91-95	Variance of tf * idf
96-100	Boolean model
101-105	Vector space model
106-110	BM25
116-120	Language model approach for information retrieval (IR) with dirichlet smoothing
121-125	Language model approach for IR with Jelinek-Mercer smoothing

tf is normalized to,

$$tf = tf * \frac{avg_l}{l(d)}$$

where, avg_l is average length of document in the corpus and $l(d)$ is the length of document d

4.2.2 DL2

Here, we take divergence model as the randomness model M. First normalization is Laplace Model and second normalization for term frequency is applied. So,

$$Inf_1 = F * D(\phi, p) + 0.5 * log_2(2\pi * tf(1-\phi))$$

where, $\phi = tf/F$, $p = 1/N$ and $D(\phi, p) = \phi * log_2\phi/p + (1-\phi) * log_2((1-\phi)(1-p))$

4.3 CBDM

It is a cluster based retrieval model [8] in which we build language models for clusters and then retrieve/rank clusters based on the likelihood of generating the query. In CBDM, we use Dirichlet Smoothing to smoothen the cluster with the collection model and in second step the document model is smoothed using the smoothed cluster model. More formally,

$$P(w \mid d) = \lambda P_{ML}(w \mid D) + (1-\lambda)P(w \mid Clstr$$

$$= \lambda P_{ML}(w \mid D) + (1-\lambda)[\beta P_{ML}(w \mid Clstr) + (1-\beta)P_{ML}(w \mid Coll)]$$

5. RESULTS AND DISCUSSIONS

In this section, we introduce the experiments on LETOR to evaluate LamdaMART, and make discussions on the experimental results on the TREC1-3, WT10G and Robust04.

Table 2: Results on Trec1-3

Method	MAP	ERR	NDCG@10
LETOR LamdaMART	**0.2656**	0.0735	0.4581
PL2	0.204	0.226	0.642
DL2	0.208	0.264	**0.696**
CBDM	0.218	0.214	0.675
QL Dirichlet Smoothing	0.227	0.265	0.691
RM3	0.262	**0.272**	0.695

Table 3: Results on Robust04

Method	MAP	ERR	NDCG@10
LETOR LamdaMART	**0.2748**	0.0705	0.4248
PL2	0.210	0.262	0.611
DL2	0.226	0.322	0.680
CBDM	0.223	0.312	0.62
QL Dirichlet Smoothing	0.257	**0.333**	**0.700**
RM3	0.270	0.33	0.685

Three widely used measures are adopted for the evaluation: ERR, MAP, and NDCG@10.

5.1 *Overall* Performance

The ranking performances of learning to rank models with different data sets are listed in Table 2, 3 and 4. Here we have compared the LambdaMART LETOR with the five baseline methods for MAP, ERR and nDCG. The best performing method for each metric is marked **bold**.

According to these experimental results,we found that LamdaMART gives better MAP than other baselines for all the data sets.The results demonstrate the power of Learning to Rank to improve the metric values for the model it is

Table 4: Results on WT10G

Method	MAP	ERR	NDCG@10
LETOR LamdaMART	**0.2204**	0.0612	0.443
PL2	0.210	0.213	0.623
DL2	0.208	0.203	0.654
CBDM	0.209	0.215	0.665
QL Dirichlet Smoothing	0.201	0.254	**0.675**
RM3	0.216	**0.243**	0.632

trained. Since we use handpicked features which are known to give a good estimation of a document's similarity to a query, the combined performance of these features give better results than individual baselines. This can be attributed to the concept of feature engineering [1] in machine learning.

We see that for Robust04, QL with Dirichlet Smoothing is more effective while for Trec1-3, DL2 gave the best NDCG value and QL with Dirichlet Smoothing gave the best ERR value. The baselines outperform LamdaMART for ERR and NDCG@10 due to the fact that LamdaMART models were tuned for MAP and not ERR and NDCG@10. We discuss more about this in section 6.

5.2 Performance across data sets

While Trec1-3 and Robust04 considers only the content of the document, the data set WT10G considers the body,title and others. As a result, it can be noted that the margin of difference between MAP for LamdaMART and other baselines is reduced for WT10G set in Table 4. We attribute this fact to the increased number of features. Increased number of features gives a better estimate of the evaluation metrics.

6. CONCLUSION AND FUTURE WORK

In this report, we have demonstrated that use of LamdaMART learning to rank in Information Retrieval and evaluated its performance against 5 strong baselines. We showed that LamdaMART fared particularly well when we used MAP as an evaluation metric, independent of the data sets. We also demonstrated that the biggest advantage of Learning to Rank was to incorporate new features as a vector easily. We did this by using multiple field features like body, title and others in WT10G dataset.

The choice of evaluation measure is very important, because it plays a critical role in the performance of a retrieval model. The main point of the decision should be the purpose and use of ranking model. We also noted that the partitioning and selection of tree and leaf parameters plays an important role in the values for the selected evaluation metrics. Selecting the right parameters and features gives LamdaMART a substantial improvement over classical retrieval models and baselines.

As part of the future work we want to tune our LamdaMART model to specific evaluation metrics like NDCG@10 and ERR and compare its performance with the scores of the above baselines.

7. REFERENCES

[1] Feature engineering.

[2] Jsoup for java.

[3] Learning to rank for information retrieval by microsoft research.

[4] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.*, 20(4):357–389, Oct. 2002.

[5] K. Crammer and Y. Singer. Pranking with ranking. 2002.

[6] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. volume 7, 1989.

[7] A. G. Julia Kiseleva. Applying learning to rank techniques to contextual suggestions. 2014.

[8] X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 186–193, New York, NY, USA, 2004. ACM.

[9] C. B. P. Li and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. 2007.

[10] N. Y. Times. Google keeps tweaking its search engine. 2008.

[11] R. E. S. Y. Freund, R. Iyer and Y. Singer. An efficient boosting algorithm for combining preferences. volume 4 of *Journal of Machine Learning Research*, 2003.