RankDE: Learning a Ranking Function for Information Retrieval using Differential Evolution

Danushka Bollegala
The University of Tokyo
Hongo 7-3-1, Tokyo
113-8656, Japan
danushka@iba.t.u-tokyo.ac.jp

Nasimul Noman
The University of Tokyo
Hongo 7-3-1, Tokyo
113-8656, Japan
noman@iba.t.u-tokyo.ac.jp

Hitoshi Iba
The University of Tokyo
Hongo 7-3-1, Tokyo
113-8656, Japan
iba@iba.k.u-tokyo.ac.jp

ABSTRACT

Learning a ranking function is important for numerous tasks such as information retrieval (IR), question answering, and product recommendation. For example, in information retrieval, a Web search engine is required to rank and return a set of documents relevant to a query issued by a user. We propose *RankDE*, a ranking method that uses differential evolution (DE) to learn a ranking function to rank a list of documents retrieved by a Web search engine. To the best of our knowledge, the proposed method is the first DE-based approach to learn a ranking function for IR. We evaluate the proposed method using LETOR dataset, a standard benchmark dataset for training and evaluating ranking functions for IR. In our experiments, the proposed method significantly outperforms previously proposed rank learning methods that use evolutionary computation algorithms such as Particle Swam Optimization (PSO) and Genetic Programming (GP), achieving a statistically significant mean average precision (MAP) of 0.339 on TD2003 dataset and 0.430 on the TD2004 dataset. Moreover, the proposed method shows comparable results to the state-of-the-art non-evolutionary computational approaches on this benchmark dataset. We analyze the feature weights learnt by the proposed method to better understand the salient features for the task of learning to rank for information retrieval.

Track: Real-World Applications.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms

Keywords

Differential Evolution, Learning to rank, Information Retrieval, Web Search

GECCO'11, July 12–16, 2011, Dublin, Ireland.

1. INTRODUCTION

The World Wide Web has grown into a huge collection of Web pages containing information regarding numerous concepts. Web search engines have gained popularity as one of the most important access methods to the Web [1]. Web search engine users formalize their information need in the form of a search query and retrieve the information that they seek for. As a result of both the vast amount of Web pages and the ambiguity in queries, a search engine often retrieves numerous documents for a single query. To reduce the burden of users to go through the entire list of retrieved documents to find the information they are searching for, search engines rank the retrieved set of documents according to the *relevancy* of a document to a query issued by the user and displays a ranked list of documents. Accurate ranking of the retrieved set of documents is therefore an important component in a search engine that enables users to quickly find the information that they need. On the other hand, if a search engine often ranks documents that are irrelevant to user queries as the top hits, then the users lose confidence in that search engine. Consequently, learning to rank documents retrieved for a user query has gained much attention and several initiatives have taken place such as the Yahoo!'s Learning to Rank Challenge[1], and Microsoft's LETOR project[2].

Learning an accurate ranking function to rank a set of documents retrieved for a given user query remains a difficult problem because of several challenges. First, there are many factors a ranking function must take into consideration in the Web when determining the rank of a document such as the content of the page (i.e. whether the document contains the words in the query or not), link structure (i.e. the number of in-bound and out-bound links to the document), novelty (i.e. how regularly the content of the document is revised?), authority (i.e. encyclopedic resources such as Wikipedia vs. a blog) etc. Integrating those heterogenous factors to construct a single ranking function is a non-trivial task. Second, the learning algorithm must be fast and scalable to be used in a Web search engine. For example, LETOR dataset contains over 25 million documents with rank information annotated that must be used to learn a ranking function. Moreover, speed at test (retrieval) time is also important because a search engine must rank and return a large number of documents for a single user query. Third, the measures that are used to evaluate a ranking algorithm in information retrieval such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG) are non-convex and difficult to directly optimize by conventional optimization tools. Most existing approaches for learning to rank resort to some sort of a convex

[1] http://learningtorankchallenge.yahoo.com/
[2] http://research.microsoft.com/en-us/um/beijing/projects/letor/

approximation to those measures and then use standard machine learning algorithms.

In this paper, we propose a rank learning approach using differential evolution (DE) [27, 26] that we designate *RankDE*. The proposed method directly optimizes over the evaluation metrics used in information retrieval such as MAP and NCDG, without requiring any approximations. The main contributions of our work can be summarized as follows.

- We propose RankDE, a rank learning method for information retrieval using differential evolution.
- We evaluate the proposed method on the LETOR dataset – a benchmark dataset developed by Microsoft Research to systematically evaluate different rank learning methods. Our method outperforms numerous baselines that are popularly used to rank documents in Web search such as BM25, as well as previously proposed rank learning algorithms that use evolutionary computation (EC) methods such as Genetic Programming (GP) [32] and Particle Swam Optimization (PSO) [6]. Moreover, the proposed method outperforms non-EC methods such as RankSVM (based on Support Vector Machines) and RankBoost (based on AdaBoost).
- We analyze the weights learnt by the proposed method for numerous popular features that are used to learn ranking functions in IR to gain an insight into the importance of different features for the task of learning to rank for IR.

The remainder of this paper is organized as follows. We first introduce the learning to rank problem in Section 2.1 followed up by a brief overview of differential evolution in Section 2.2. Next, our proposed rank learning method, *RankDE*, is presented in Section 3. We compare the proposed method against numerous baselines as well as previously proposed EC approaches and non-EC approaches in Section 4. We analyze the weights learnt by the proposed method for the different features used in learning ranking functions. We review related previous work on this topic in Section 5 and conclude the paper.

2. BACKGROUND

In this Section, we describe the problem of learning to rank for information retrieval and present a brief overview of differential evolution. This background information will be helpful to better understand the proposed ranking method, which is presented in detail in Section 4.

2.1 Learning to Rank Problem

The learning to rank problem in the context of information retrieval consists of two main phases: the training phase where we learn a ranking function from a set of annotated training data, and the test (i.e. retrieval) phase where we apply the learnt ranking function to rank a set of documents retrieved by a search engine for a user query. Specifically, in the training phase, a learning algorithm is presented with a collection of queries and their corresponding retrieved documents. Moreover, the documents are assigned with some labels that indicate the relevance judgements of those documents to their corresponding query. The relevance judgements are assigned by human annotators. For example, an annotator might annotate a set of documents by assigning some ranking score to each document depending on its relevance to the query. A higher ranking score indicates that a document with such a score is more relevant to the user query and must be ranked at the top.

The objective of learning is to construct a ranking model (e.g. a ranking function) that achieves the best *agreement* with the ranking induced by the scores assigned by the human annotators. The agreement between a ranking produced by an algorithm and that of a human annotator for a set of documents can be measured using numerous rank evaluation metrics such as Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG), Kendall's rank correlation coefficient and Spearman's rank correlation coefficient. Traditionally, both MAP and NDCG [1] have been used in the information retrieval community to evaluate rankings in Web search engines because those measures are shown to be correlating well with the document relevance in the Web search settings [12].

Once a ranking function is learnt using a set of training data, it can be used to rank a set of documents retrieved for a user query at test (i.e. retrieval) phase. Let us denote the set of queries by $Q = \{q_1, \ldots, q_{|Q|}\}$, where we use the notation, $|Q|$ to represent the number of element in set Q. Likewise, let $D = \{d_1, \ldots, d_{|D|}\}$ be the set of documents. Then, the training dataset is created as a set of query-document pairs, $(q_i, d_j) \in Q \times D$, in which each element is assigned with a relevance judgement (e.g. a label) $y(q_i, d_j)$ indicating the relationship between q_i and d_j by a human annotator. For example, the relevance judgement can be simply a binary relevance indicating whether the document d_j is relevant (i.e. $y(q_i, d_j) = 1$) or non-relevant ((i.e. $y(q_i, d_j) = 0$) for the query q_i, or it can be some ranking score (i.e. (i.e. $y(q_i, d_j) \in \mathbb{R}$) that induces a total ordering among all the documents d_j retrieved for the query q_i. A query-document pair (q, d) is represented using a feature vector $\boldsymbol{\phi}(q, d)$. The actual features used for training are discussed later in Section 4.1. The ranking model is defined as a real valued function $f(q, d)$ over features as follows,

$$f(q, d) = \boldsymbol{w}^\top \boldsymbol{\phi}(q, d). \tag{1}$$

Here, $\boldsymbol{w}$ denotes a weight vector indicating the importance of each feature towards the ranking score returned by the ranking function. To rank the documents retrieved for a query q_i, we compute $f(q_i, d_j)$ for each retrieved document d_j using Eq.1 and sort the documents in the descending order of their ranking score. Limiting the ranking functions to linear combinations of features as defined in Eq.1 is common to all previous work in learning to rank for information retrieval. Web search engines must rank and present a large number of documents to a single user query within few milliseconds. Consequently, the linear ranking function defined in Eq.1 only requires the inner product between two vectors, which can be computed quickly and makes it attractive as a ranking function for large-scale information retrieval systems.

2.2 Differential Evolution

Differential evolution (DE), proposed by Storn and Price [27], is a simple yet powerful population-based stochastic search technique for solving global optimization problems. DE has been used successfully in numerous fields such as pattern recognition [27], communication [15], and mechanical engineering [25], to optimize non-convex, non-differentiable and multi-modal objective functions. DE has many attractive properties compared to other evolutionary algorithms such as, implementation simplicity, the small number of control parameters, fast convergence rate, and robust performance.

DE has only a few control variables which remain fixed throughout the optimization process, which makes it easy to implement. Moreover, DE can be implemented in a parallel processing framework, which enables it to process a large number of training instances efficiently. These properties of DE makes it an ideal candidate for the current task of learning a ranking function for in-

formation retrieval, where we must optimize non-convex objective functions such as MAP and NDCG measures over large datasets. Next, we briefly outline the main steps in DE. For further details of DE and its comparison to other evolutionary computational approaches refer [27]. Without a loss of generality, we will consider the problem of maximizing a given objective function. (A similar approach can be followed for minimization.)

DE is a parallel direct search method which utilizes P number of L-dimensional parameter vectors, $\boldsymbol{x}_{i,G}$, where $i = 1, \ldots, P$. Here, we denote the population size by P and the i-th individual in the G-th generation is represented by an L-dimensional vector $\boldsymbol{x}_{i,G}$. The population size, P, does not change during the optimization process and the initial population is created by randomly generating the vectors such that they cover the entire range of values of the parameter space. In DE, we do not use selection pressure for selecting parents. Instead, in each generation, every individual $\boldsymbol{x}_{i,G}$, once becomes the principal parent to breed its own offsprings mating with other parents.

DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector. This *differential* operation is often referred to as mutation in DE. Using the above notation, we can write the mutation operation as follows,

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F(\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}). \quad (2)$$

Here, $\boldsymbol{v}_{i,G+1}$ is a mutant vector and r_1, r_2, r_3 are three mutually different random indexes selected from the set $\{1, 2, 3, \ldots, P\}$. The value, $F(> 0)$, is a real number typically less than one.

To increase the diversity of the mutated parameter vectors, a crossover operation is performed. A trial vector (i.e. the child), $\boldsymbol{u}_{i,G+1} = (u_{1i,G+1}, \ldots, u_{Li,G+1})$ is constructed from the mutated vector, $\boldsymbol{v}_{i,G+1}$, and the original population vector (i.e. the parent), $\boldsymbol{x}_{i,G}$, according to the following criterion,

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{IF } (randb(j) \leq CR) \text{ OR } (j = rnbr(i)), \\ x_{ji,G} & \text{IF } (randb(j) > CR) \text{ AND } (j \neq rnbr(i)). \end{cases} \quad (3)$$

Here, $randb(j)$ is the j-th evaluation of a uniform random number generator with outcomes in the range $[0, 1]$, $CR \in [0, 1]$ is a user-specified crossover constant, and $rnbr(i)$ is a randomly chosen index from the set $\{1, 2, \ldots, L\}$, which ensures that $\boldsymbol{u}_{i,G+1}$ gets at least one parameter from $\boldsymbol{v}_{i,G+1}$.

The selection scheme used in DE is also known as *parent-child competition*. Specifically, once the child $\boldsymbol{u}_{i,G+1}$ is computed according to the above-mentioned procedure the objective function is evaluated for the child $\boldsymbol{u}_{i,G+1}$ and the parent $\boldsymbol{x}_{i,G}$. If $\boldsymbol{u}_{i,G+1}$ yields a higher objective function value than that by $\boldsymbol{x}_{i,G}$, then $\boldsymbol{x}_{i,G+1}$ is set to $\boldsymbol{u}_{i,G+1}$. Otherwise, the parent (i.e. $\boldsymbol{x}_{i,G}$) is retained for the $(G + 1)$-th generation.

3. PROPOSED RANK LEARNING METHOD FOR IR: RANKDE

Motivated by the success of DE in a wide range of tasks, we propose *RankDE*, a learning to rank method for information retrieval using DE. The goal of RankDE is to discover a good ranking functions adapted to the properties of a given query-document collection, which are also able to generalize to unseen new queries and documents retrieved at test times. Main steps of RankDE are summarized in Algorithm 1.

RankDE takes as input a training dataset T that contain query-document pairs (q, d) with their corresponding feature vectors $\boldsymbol{\phi}(q, d)$. As an instance of DE, RankDE learns a weight vector $\boldsymbol{x}_{i,G}$ that optimizes some fitness function, which evaluates how well the rank scores assigned by the learnt ranking function agree with those assigned by the human annotators. The final output of Algorithm 1 is the parameter vector that maximizes the fitness function. We use Mean Average Precision (MAP) [1] (defined in Section 4.2) as the fitness function. MAP is frequently used to measure the accuracy of a document ranking algorithm in information retrieval systems.

Algorithm 1 RankDE: A DE-based rank learning algorithm.

Input: A training set T of query-document pairs with their feature vectors.

Output: A ranking function $f(q, d)$ that assigns a score to a document d indicating its relevancy to a query q.

```
for each generation G do
  for each parent i in generation G do
    select distinct r1, r2, r3 randomly from current population.
    compute v_{i,G+1} using Eq.2.
    compute u_{i,G+1} using Eq.3.
    if Fitness(u_{i,G+1}, T) > Fitness(x_{i,G}, T) then
      x_{i,G+1} = u_{i,G+1}
    end if
  end for
end for
return f(q, d) = w^T φ(q, d), where w is the individual with the highest fitness value in the final population.
```

The fitness function, $\mathrm{Fitness}(\boldsymbol{w}, T)$, used by RankDE is defined as follows,

$$\mathrm{Fitness}(\boldsymbol{w}, T) = \mathrm{MAP}(\boldsymbol{w}, T). \quad (4)$$

Here, $\mathrm{MAP}(\boldsymbol{w}, T)$ denotes the mean average precision value that we would obtain on training data if we use the weight vector $\boldsymbol{w}$ in ranking function $f(q, d)$ (Eq.1) to assign a score to each document d according to its relevancy to a query q and then sort those documents in the descending order of the assigned scores to obtain a ranked list of documents.

4. EXPERIMENTS

In this Section, we first describe the LETOR dataset and then present the experimental results on this benchmark dataset.

4.1 Datasets

We use the LETOR (version 2.0) benchmark dataset [20], which is used in much previous work investigating the problem of learning to rank for information retrieval. By using this dataset, we can directly compare the performance of the proposed method against previously proposed learning to rank algorithms for information retrieval. The LETOR version 2.0 consists of TD2003 and TD2004 datasets, which were part of the topic distillation task of the Text REtrieval Conference (TREC) in year 2003 and 2004. TD2003 dataset contains 50 queries and TD2004 dataset contains 75 queries. The document collection contains $1,053,110$ documents together with $11,164,829$ hyperlinks and is based on a January, 2002 crawl of the .gov domain. Topic distillation aims to find a list of documents relevant to a particular topic. The TREC committee provides judgements for the topic distillation task. For each query in TD2003 and TD2004 datasets, there are about $1,000$ documents listed. Each query-document pair is given a binary judgement indicating whether a document is relevant or non-relevant for a particular query.

A query-document pair in the LETOR dataset is represented using a 44 dimensional feature vector. The numerous features used in the LETOR dataset are shown in Table 1. The features include

Table 1: Features in the LETOR TD2003 and TD2004 datasets.

Category	Feature	No. of features
Content (low-level)	tf [1]	4
	idf [1]	4
	dl [1]	4
	tfidf [1]	4
Content (high-level)	BM25 [23]	4
	LMIR [33]	9
Hyperlink	PageRank [19]	1
	Topical PageRank [18]	1
	HITS [16]	2
	Topical HITS [18]	2
	HostRank [31]	1
Hybrid	Hyperlink-base relevance propagation [24]	6
	Sitemap-based relevance propagation [21]	2
Total		44

numerous ranking heuristics popularly used in the information retrieval community to rank a list of retrieved documents. The set of features used in LETOR includes low-level features such as, term frequency (tf), inverse document frequency (idf), document length (dl) combinations of low-level features such as tf*idf [1], as well as high-level features such as BM25 [23] and LMIR [33]. Hyperlink structure provides useful clues about the relevancy of a web page. Consequently, several features are computed using the hyperlink information in LETOR datasets such as PageRank [19], HITS [16], HostRank [31], topical PageRank and topical HITS [18].

It is noteworthy that the values of features extracted for documents retrieved for different queries are not comparable. Therefore, we first normalize the values of each feature across all documents retrieved for a particular query. Let us denote the set of documents retrieved for query q_i by $D(q_i)$ and a document in this set by d_j (i.e. $d_j \in D(q_i)$). Moreover, let us denote the k-th feature in the feature vector $\boldsymbol{\phi}(q_i, d_j)$ representing a query-document pair (q_i, d_j) by $\phi_k(q_i, d_j)$. Then, the normalized value of $\phi_k(q_i, d_j) \in [0, 1]$ is calculated as follows,

$$\phi_k(q_i, d_j) = \frac{\phi_k(q_i, d_j) - \min\{\phi_k(q_i, d_j)\}}{\max\{\phi_k(q_i, d_j)\} - \min\{\phi_k(q_i, d_j)\}}. \quad (5)$$

4.2 Evaluation Measures

To evaluate a ranking produced by an algorithm for a set of documents retrieved for a particular query, we can compare it against the ranking induced by the scores assigned by a human annotator for those documents. Precision at position n ($P@n$), Mean Average Precision (MAP), and normalized discounted cumulative gain (NDCG) are three widely used rank evaluation measures in the information retrieval community. Both those evaluation measures are in the range $[0, 1]$, where a method that produces the exact ranking as in the gold standard achieves the score of 1. Next, we describe each of those evaluation measures in detail.

Precision at rank n ($P@n$) [1] measure is defined as the proportion of the relevant documents among the top n-ranked documents,

$$P@n = \frac{\text{No. of relevant docs in top } n \text{ results}}{n}. \quad (6)$$

Table 2: Parameter settings for RankDE.

Parameter	Value
Population size (P)	50
maximum no. of generations	$10,000$
no. of dimensions (L)	44
F value	0.5
crossover rate (CR)	0.5

Average precision averages the $P@n$ at over different n values to produce a single measure for a given query as follows,

$$AP = \frac{\sum_{n=1}^{N}(P@n \times rel(n))}{\text{No. of relevant docs for this query}}. \quad (7)$$

Here, N is the number of retrieved documents, and $rel(n)$ is a binary function that returns the value 1 if the n-th ranked document is relevant to the query under consideration and 0 otherwise. Mean average precision (MAP) is computed as the average of AP over all queries in the dataset.

NDCG considers the reciprocal of the logarithm of the rank assigned to relevant documents. For a ranked list of documents retrieved for a query, NDCG value at position n, $NDCG@n$, is computed as follows,

$$NDCG@n = Z_n \sum_{j=1}^{n} \frac{2^{r(j)} - 1}{\log(1 + j)}. \quad (8)$$

Here, $r(j)$ is the rating of the j-th document in the ranked list, and the normalization constant Z_n is chosen such that a perfectly ranked list would obtain an NDCG@n score of 1. Specifically, it is given by,

$$Z_n = \frac{1}{\sum_{j=1}^{n} \frac{1}{\log(1+j)}}. \quad (9)$$

For the TD2003 and TD2004 datasets, we define two values of ratings 0 and 1 respectively corresponding to relevant and non-relevant documents in order to compute NDCG scores. In our evaluations, we report the average values taken over all the queries in a dataset as $P@n$ and $NDCG@n$.

4.3 Parameter Settings

The parameters in RankDE are set to the values shown in Table 2 by measuring the performance on the validation data provided in the LETOR datasets. Each individual is represented by a 44 dimensional real-valued vector in which, each dimension corresponds to some feature found in the LETOR datasets. The initial population is generated randomly by selecting the parameter values from the range $[-1, 1]$.

4.4 Evaluation Procedure and Baselines

We compare our DE-based learning to rank algorithm, RankDE, with several baselines and previously proposed EC-based rank learning algorithms using TD2003 and TD2004 datasets. Next, we briefly describe each of those algorithms.

BM25: BM25 [23] is a non-learning ranking function that combines numerous statistics to compute a ranking score that reflects the relevancy of a document to a given query. It utilizes information such as the number of times the query occur in a document (i.e. term frequency), the number of documents that contains the query (i.e. document frequency), the length

of the document in words, and the average length of a document (i.e. the average number of words contained in any document in the collection). BM25 was successfully used in the Okapi information retrieval system and is popularly known as Okapi BM25. This baseline demonstrates the performance that we would obtain if we did not use any training data to learn a ranking function.

RankSVM: Ranking Support Vector Machine [13] is an extension to the standard binary support vector classifier [29] that performs ordinal regression. Specifically, RankSVMs learn a large margin classifier that minimizes the number of discordant pairs between two sets of ranks. It is a pairwise learning algorithm, which indirectly optimizes the rank evaluation measures such as the MAP, via minimizing the number of discordant pairs between a human-made ranking and a system-made ranking.

RankBoost: Freund et al. [10] proposed RankBoost to combine multiple rankings using the AdaBoost [11] algorithm. RankBoost works by combining multiple weak rank scores of a given set of training instances. The weak rank scores might be only weakly correlated with the target (human assigned) ranking scores. By combining such a set of weak rank scores via boosting, RankBoost is able to learn an accurate ranking function. The features exist in the LETOR datasets such as term-frequency, PageRank, BM25, etc. are used to create the weak rankings by the RankBoost algorithm.

SwamRank: This is a particle swam optimization (PSO)-based ranking algorithm that attempts to learn a linear combination of numerous ranking functions in the form of Eq.1. SwamRank [6] directly optimizes the MAP for a given training dataset.

GPRank: This is the genetic programming-based rank learning algorithm proposed by Yeh et al. [32] for information retrieval. Similar to SwamRank, GPRank directly optimizes the MAP for a given training dataset. Both SwamRank and GPRank are further detailed in the Related Work Section (i.e. Section 5).

RankDE: This is the DE-based rank learning algorithm proposed in this paper.

We conduct 5-fold cross-validation using the LETOR datasets, following the official guidelines and data partitions as described in the LETOR project [20]. For each fold, we use three subsets as training data, one subset as validation data, and the remaining subset for testing. To report the performance of the ranking function learnt by RankDE, we compute the evaluation measures MAP, $P@n$, and $NDCG@n$ on the test. The reported performance in this paper is the average over the five folds.

4.5 Results

Tables 3 and 4 compare the performance of the proposed RankDE against the methods described in Section 4.4 respectively using the LETOR2 TD2003 and TD2004 datasets. Except for RankDE, all other results reported in Tables 3 and 4 are obtained from published work and we do not implement those methods here by ourselves. All results reported in Tables 3 and 4 use the officially released LETOR2 datasets and evaluation tools, which enable us to make a direct and a fair comparison.

From Tables 3 and 4 we see that the proposed RankDE has the best performance in terms of MAP scores among the different methods compared. The improvements reported by RankDE over all other methods are statistically significant (paired t-test at probability level 0.05). RankDE has the highest $NDCG@1$ and $P@1$ values in both TD2003 and TD2004 datasets. This implies that the proposed DE-based rank learning method (RankDE) is able to rank relevant documents as the first hit, which is a desirable quality of a ranking function for a search engine. It is interesting to note that on the TD2004 dataset, none of the previously proposed EC-based ranking algorithms (i.e. SwamRank and GPRank) were able to outperform RankBoost, a non-EC algorithm. However, the proposed DE-based rank learning algorithm (RankDE) outperforms RankBoost as well as all the other algorithms in this dataset as well.

4.6 Feature Analysis

Recall that the ranking function we learn (defined in Eq.1) using the proposed method can be interpreted as a weighted linear combination of 44 different features. By inspecting the final weight vector returned by Algorithm 1 we can gain some insight into which features are important for determining the relevance of a document retrieved for a query. The weights learnt by RankDE using the TD2003 dataset for different features are shown in Table 5. We have sorted the features in the descending order of their weights. Although not shown here for the limited availability of space, a similar ranking among features is observed for the weight vector learnt using the TD2004 dataset. From Table 5, we see that heuristics that are known to produce better document rankings such as the HostRank, inverse document frequency (idf) of the body text, inbound hyperlinks and HITS are assigned high positive weights by RankDE. The ability of the proposed method to detect salient features for ranking is important if we want to use a large number of features in an information retrieval system. For example, we can prune the trained model based on the weights learnt for different features to improve the speed during test (retrieval) phase, which can be critical for online Web search engines.

5. RELATED WORK

Learning to rank has received much attention lately as a result of the popularity of Web information retrieval and recommender systems. A Web search engine must rank a set of retrieved documents for a user query according to their relevance, whereas a recommender system must rank a list of product items for each user according to their preferences. Three different approaches exist to learn a ranking function for information retrieval: pointwise approach, pairwise approach and listwise approach.

In the pointwise approach [14, 4] each query-document pair is considered separately during training to learn an ordinal regression function that outputs a rank. The pointwise approach does not consider the relative preferences between two documents retrieved for the same query. Consequently, it has shown poor performance in learning to rank.

In contrast, the pairwise approach [13, 10, 2, 28] considers two documents d_j and d_k retrieved for the query q_i and forms pairwise preferential constraints. For example, if the human annotators prefer a document d_j to d_k as being relevant for q_i, then the constraint $f(q_i, d_j) > f(q_i, d_k)$ is formed. The weight vector $\boldsymbol{w}$ is optimized such that the majority of the pairwise preference constraints are satisfied. Pairwise rank learning is closely related to the problem of binary classification and numerous algorithms have been employed successfully under the pairwise approach to learn a ranking function such as Ranking Support Vector Machines (RankSVM) [13], RankBoost [10], and RankNet [2] respectively based on support vector machines [29], AdaBoost [11], and neural networks. However, one fundamental problem associated with the pairwise approach is that it only considers two documents at a time and ignore

Table 3: Ranking performance on the TD2003 dataset.

Method	BM25	RankSVM	RankBoost	SwamRank	GPRank	RankDE
MAP	0.126	0.256	0.212	0.209	0.283	0.339
P@1	0.120	0.420	0.260	0.453	0.520	0.600
P@2	0.130	0.350	0.270	0.330	0.420	0.400
P@3	0.160	0.340	0.240	0.269	0.370	0.333
P@4	0.145	0.300	0.230	0.223	0.330	0.300
P@5	0.148	0.264	0.220	0.207	0.280	0.280
P@6	0.140	0.243	0.210	0.188	0.270	0.250
P@7	0.129	0.234	0.211	0.185	0.250	0.243
P@8	0.120	0.233	0.193	0.173	0.240	0.237
P@9	0.116	0.218	0.182	0.164	0.230	0.222
P@10	0.109	0.206	0.178	0.151	0.220	0.210
NDCG@1	0.120	0.420	0.260	0.453	0.520	0.600
NDCG@2	0.140	0.370	0.280	0.343	0.450	0.445
NDCG@3	0.176	0.379	0.270	0.307	0.420	0.388
NDCG@4	0.174	0.363	0.272	0.284	0.390	0.356
NDCG@5	0.183	0.347	0.279	0.278	0.380	0.336
NDCG@6	0.184	0.341	0.280	0.271	0.370	0.310
NDCG@7	0.184	0.340	0.287	0.273	0.360	0.300
NDCG@8	0.185	0.345	0.282	0.270	0.350	0.292
NDCG@9	0.186	0.342	0.282	0.267	0.350	0.279
NDCG@10	0.186	0.341	0.285	0.263	0.350	0.267

Table 4: Ranking performance on the TD2004 dataset.

Method	BM25	RankSVM	RankBoost	SwamRank	GPRank	RankDE
MAP	0.282	0.350	0.384	0.314	0.362	0.430
P@1	0.307	0.440	0.480	0.400	0.450	0.692
P@2	0.293	0.407	0.447	0.380	0.420	0.500
P@3	0.258	0.351	0.404	0.351	0.380	0.436
P@4	0.243	0.327	0.347	0.317	0.330	0.404
P@5	0.229	0.291	0.323	0.296	0.320	0.385
P@6	0.224	0.273	0.304	0.278	0.300	0.333
P@7	0.210	0.261	0.293	0.253	0.280	0.308
P@8	0.192	0.247	0.277	0.235	0.260	0.308
P@9	0.182	0.236	0.262	0.221	0.250	0.282
P@10	0.175	0.225	0.253	0.215	0.240	0.254
NDCG@1	0.307	0.440	0.480	0.400	0.450	0.692
NDCG@2	0.327	0.433	0.473	0.413	0.440	0.544
NDCG@3	0.314	0.409	0.464	0.404	0.430	0.488
NDCG@4	0.315	0.406	0.439	0.393	0.440	0.458
NDCG@5	0.319	0.939	0.437	0.391	0.440	0.438
NDCG@6	0.325	0.397	0.448	0.394	0.450	0.399
NDCG@7	0.326	0.406	0.457	0.392	0.460	0.377
NDCG@8	0.324	0.410	0.461	0.396	0.470	0.371
NDCG@9	0.332	0.414	0.464	0.397	0.470	0.350
NDCG@10	0.335	0.420	0.472	0.402	0.470	0.328

Table 5: Weights learnt for the different features by RankDE.

Feature	Weight
HostRank	5.9932
idf(body)	3.938
HyperlinkScore(weighted-in-link)	3.798
HITS(hub)	3.548
tfidf(anchor)	2.571
LMIR.ABS(title)	2.299
LMIR.JM(anchor)	2.091
BM25(anchor)	1.870
TopicalHITS(authority)	1.727
HyperlinkFeature(weighted-in-link)	1.653
TopicalPageRank	1.460
LMIR.DIR(anchor)	1.440
sitemap2	1.341
BM25(title)	1.168
BM25(exttitle)	1.106
BM25	0.964
tf(url)	0.894
tfidf(url)	0.794
HyperlinkScore(weighted-out-link)	0.693
tfidf(title)	0.613
DL(body)	0.590
LMIR.JM(title)	0.491
tf(title)	0.453
LMIR.ABS(anchor)	0.280
tf(anchor)	0.248
LMIR.JM(title)	0.183
HyperlinkFeature(uniform-out-link)	−0.053
HITS(authority)	−0.416
tfidf(body)	−0.504
sitemap1	−0.517
HyperlinkScore(uniform-out-link)	−0.571
TopicalHITS(hub)	−0.611
LMIR.ABS(exttitle)	−0.841
PageRank	−0.989
LMIR.JM(exttitle)	−1.020
DL(title)	−1.082
DL(anchor)	−1.122
HyperlinkFeature(weighted-out-link)	−1.152
tf(body)	−1.371
LMIR.DIR(exttitle)	−2.047
idf(anchor)	−4.365
idf(title)	−7.841
DL(url)	−8.210
idf(url)	−11.305

the other documents retrieved for a query. This is particularly severe in the case of learning to rank for information retrieval because at test time we must decide the total ordering of a list of documents and not partial orderings between two documents.

The listwise approach [22, 3, 17, 30] consider the entire set of documents retrieved for a query during the training phase, there by overcoming the above-mentioned disfluencies in the pointwise and pairwise approaches. Because in information retrieval we must apply the learnt ranking function to induce a total ordering for a set of documents retrieved for a user query, the listwise approach models this setting closely compared to the pointwise or the pairwise approaches. Therefore, we follow the listwise approach in this paper to learn a ranking function from a given set of training data. Different loss functions have been employed in prior work on listwise rank learning leading to numerous algorithms such as ListNet [3] using cross entropy, RankCosine [22] using cosine loss, and ListMLE [30] using likelihood loss. However, these methods do not directly optimize the evaluation criteria used in information retrieval such as MAP or NDCG, instead approximate them via the above-mentioned loss functions. In contrast, our proposed method can directly optimize those evaluation criteria without requiring any approximations.

Genetic programming (GP) has been used to learn ranking functions in literature. Fan et al. [8, 9, 7] proposed a GP-based approach to learn a term-weighting formula by combining different parameters. First, they use an expression tree data structure to represent a term-weighting formula and then apply genetic programming to select the best performing function. Numerous operators such as addition, subtraction, multiplication, division, square root, logarithm etc. are considered. Almeida et al. [5] propose *Combined Component Approach* (CCA), a GP-based ranking function, that combines several term-weighting components such as term frequency, collection frequency, etc. to generate new ranking functions. Yeh et al. [32] propose a learning method that employs GP to learn a ranking function for information retrieval using the LETOR dataset. In this paper, we refer this GP-based rank learning method as GPRank. Diaz-Aviles et al. [6] propose SwamRank, a ranking method that uses particle swam optimization (PSO). They use the LETOR benchmark dataset and learn a linear combination of different features that represent a query-document pair to maximize average precision or NDCG measure on train data. As shown in our experiments on the LETOR benchmark dataset (Section 4), our proposed method (RankDE) which uses differential evolution outperforms both GP (GPRank) and PSO (SwamRank) based previously proposed methods.

6. CONCLUSION

In this paper, we proposed a differential evolution-based rank learning method for information retrieval. The proposed method (named as RankDE) directly optimizes the mean average precision (MAP) over a set of queries, without requiring any convex approximations as required by most of the previously proposed rank learning algorithms for information retrieval. We evaluated the proposed method using the LETOR benchmark datasets. In our experiments, the proposed method significantly outperformed numerous other rank learning methods such as BM25, RankSVM, RankBoost, SwamRank and GPRank. Moreover, a close investigation into the weights learnt by the proposed method for different features used for learning reveals that the proposed method can accurately detect salient features for ranking. In our future work, we plan to study different parameter settings and local search techniques to further improve the performance of the proposed method.

7. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML 2005*, pages 89–96, 2005.

[3] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *ICML 2007*, pages 129–136, 2007.

[4] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS'01*, 2001.

[5] H. M. de Almeida, M. A. Goncalves, M. Cristo, and P. Calado. A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *SIGIR 2007*, pages 399–406, 2007.

[6] E. Diaz-Aviles, W. Nejdl, and L. Schmidt-Thieme. Swarming to rank for information retrieval. In *GECCO 2009*, pages 9–15, 2009.

[7] W. Fan, M. D. Gordon, and P. Pathak. Personalization of search engine services for effective retrieval and knowledge management. In *twenty first international conference on information systems (ICIS'00)*, pages 20 – 34, 2000.

[8] W. Fan, M. D. Gordon, and P. Pathak. Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):523–527, 2004.

[9] W. Fan, M. D. Gordon, and P. Pathak. Genetic programming-based discovery of ranking functions for effective web search. *Journal of Management Information Systems*, 21(4):37–56, 2005.

[10] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933 – 969, 2003.

[11] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[12] B. He, C. Macdonald, and I. Ounis. Retrieval sensitivity under training using different measures. In *SIGIR'08*, pages 67 – 74, 2008.

[13] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN'99*, pages 97 – 102, 1999.

[14] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, pages 115 – 132, 2000.

[15] J. Ilonen, J.-K. Kamarainen, and J. Lampinen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17:93 – 105, 2003.

[16] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604 – 632, 1999.

[17] Y. Lan, T.-Y. Liu, Z. Ma, and H. Li. Generalization analysis of listwise learning-to-rank algorithms. In *ICML 2009*, pages 557–584, 2009.

[18] L. Nie, B. D. Davison, and X. Qi. Topical link analysis for web search. In *SIGIR'06*, pages 91 – 98, 2006.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report SIDL-WP-1999-0120, Stanford InfoLab, November 1999.

[20] T. Qin, T.-Y. Liu, J. Xu, W. Xiong, and H. Li. Letor: A benchmark collection for learning to rank for information retrieval. Technical report, Microsoft Research Asia, 2007.

[21] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. In *SIGIR'05*, pages 408 – 415, 2005.

[22] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information Processing and Management*, 2007.

[23] S. E. Robertson. Overview of the okapi projects. *Journal of Documentation*, 53(1):3 – 7, 1997.

[24] A. Shakery and C. Zhai. Relevance propagation for topic distillation uiuc trec 2003 web track experiments. In *TREC'03*, 2003.

[25] R. Storn. Differential evolution design of an iir-filter. In *IEEE International Conference on Evolutionary Computation*, pages 268 – 273, 1996.

[26] R. Storn and K. V. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, 1995.

[27] R. Storn and K. V. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):314 – 359, December 1997.

[28] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *SIGIR'07*, pages 383 – 390, 2007.

[29] V. Vapnik. *Statistical Learning Theory*. Wiley, Chichester, GB, 1998.

[30] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *ICML 2008*, pages 1192–1199, 2008.

[31] G.-R. Xue, Q. Yang, H.-J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *SIGIR'05*, pages 186 – 193, 2005.

[32] J.-Y. Yeh, J.-Y. Lin, H.-R. Ke, and W.-P. Yang. Learning to rank for information retrieval using genetic programming. In *SIGIR Workshop on Learning to rank for Information Retrieval (LR4IR)*, 2007.

[33] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179 – 214, 2004.