Learning to Rank for Information Retrieval Using Genetic Programming

Jen-Yuan Yeh[1], Jung-Yi Lin[1], Hao-Ren Ke[2], Wei-Pang Yang[3]

[1]Dept. of Computer Science, National Chiao Tung University, Hsinchu 300, TAIWAN

[2]Inst. of Information Management, National Chiao Tung University, Hsinchu 300, TAIWAN

[3]Dept. of Information Management, National Dong Hwa University, Hualien 974, TAIWAN

{jyyeh, jylin}@cis.nctu.edu.tw, claven@lib.nctu.edu.tw, wpyang@mail.ndhu.edu.tw

ABSTRACT

One central problem of information retrieval (IR) is to determine which documents are relevant and which are not to the user information need. This problem is practically handled by a ranking function which defines an ordering among documents according to their degree of relevance to the user query. This paper discusses work on using machine learning to automatically generate an effective ranking function for IR. This task is referred to as "learning to rank for IR" in the field. In this paper, a learning method, RankGP, is presented to address this task. RankGP employs genetic programming to learn a ranking function by combining various types of evidences in IR, including content features, structure features, and query-independent features. The proposed method is evaluated using the LETOR benchmark datasets and found to be competitive with Ranking SVM and RankBoost.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Retrieval Models.*

General Terms

Algorithms, Experimentation, Performance.

Keywords

Genetic programming, learning to rank for IR, ranking function

1. INTRODUCTION

Information retrieval (IR) deals with the representation, storage, organization of, and access to information items [1]. One central problem of IR is the issue of determining which documents are relevant and which are not to the user information need. In practice, this problem is usually regarded as a ranking problem, whose goal is to define, according to the degree of relevance (or similarity) between each document and the user query, an ordering among documents so as to rank relevant documents in higher positions of the retrieved list than irrelevant ones.

Over the past years, IR models, such as Boolean models, vector models, probabilistic models, and language models, have represented a document as a set of representative keywords (i.e., index terms) and defined a ranking function (or retrieval function) to associate a relevance degree with a document and a query [1]. In general, these models are designed in an unsupervised manner and thus the parameters of the underlying ranking functions, if exist, are usually tuned empirically.

In recent years, as more and more IR results coming along with relevance judgments become available, supervised learning-based methods (referred to as "learning to rank" methods in this paper) have been devoted to automatically learning an effective ranking function from training data. See [2], [3], [5], [9], [10].

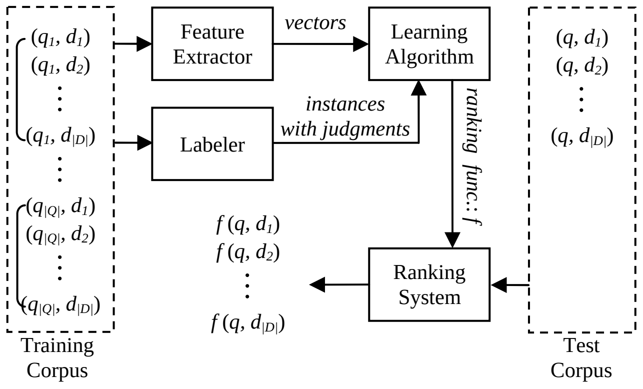

Figure 1. A general paradigm of learning to rank for IR.

Figure 1 shows a general paradigm that most learning-based methods follow to deal with the IR ranking problem. The learning process, formalized as follows, consists of two steps: training and test. Given a query collection, $Q = \{q_1, \ldots, q_{|Q|}\}$, and a document collection, $D = \{d_1, \ldots, d_{|D|}\}$, the training corpus is created as a set of query-document pairs, each $(q_i, d_j) \in Q \times D$, upon which a relevance judgment indicating the relationship between q_i and d_j is assigned by a labeler. The relevance judgment can be: (1) a class label, e.g., relevant or non-relevant, (2) a rating, e.g., definitely relevant, possibly relevant, or non-relevant, (3) an order, e.g., k, meaning that d_j is ranked in the k-th position of the ordering of all documents when q_i is considered, or (4) a score, e.g., $sim(q_i, d_j)$, specifying the degree of relevance between q_i and d_j. For each instance, (q_i, d_j), a feature extractor produces a vector of features that describe the match between q_i and d_j. Such features used in the field range from classical IR features (e.g., term frequency, inverse document frequency, and BM25 [24]) to recently-developed features (e.g., HostRank [27], Feature Propagation [23], [25], and Topical PageRank [21]). The inputs to the learning algorithm comprise training instances, their feature vectors, and the corresponding relevance judgments. The output is a ranking function, *f*, where $f(q_i, d_j)$ is supposed to give the "true" relevance judgment (as mentioned previously) for q_i and d_j. During the training process, the learning algorithm attempts to learn a ranking function such that a performance measure (e.g., classification accuracy, error rate, Mean Average Precision [1], etc.) with respect to the output relevance judgments can be optimized. In the test phase, the learned ranking function is applied to determine the relevance between each document d_i in D and a new query q. Clearly, factors, such as the form of the training instances, the form of the desired output, and the performance measure, will lead to different design of learning to rank for IR algorithms. Please refer to Section 2 for a general categorization of methods.

This paper discusses work on using machine learning to automatically produce an effective ranking function for IR. A learning method, RankGP, based on genetic programming (GP) is developed to learn a ranking function by combining different types of evidences in IR, including content features, structure features, and query-independent features. RankGP represents a potential solution (i.e., a ranking function) as an individual in a population of GP. The method evolves a population by applying genetic operations, such as crossover and mutation, over a series of generations. In each generation, a fitness function, modeled as an IR measure, MAP (Mean Average Precision) [1], is exploited to evaluate the performance of each individual in the population. The evolution is supposed to eventually generate an individual with the best fitness as the optimal solution.

The rest of this paper is organized as follows. It starts with a brief review of related works in Section 2. Section 3 then introduces the proposed GP-based learning method, RankGP, for the task of learning to rank for IR. The experimental results and discussions are provided in Section 4. Finally, Sections 5 and 6 conclude this paper and give directions for further research.

2. RELATED WORK

The task of learning to rank has recently drawn a lot of interest in machine learning. As distinguished by [3] and [4], previous works fall into three categories: (1) the point-wise approach, (2) the pair-wise approach, and (3) the list-wise approach.

In the point-wise approaches, each training instance is associated with a rating. The learning is to find a model that can map instances into ratings that are close to their true ones. A typical example is PRank [5], which trains a Perceptron model to directly maintain a totally-ordered set via projections. The pair-wise approaches take pairs of objects and their relative preferences as training instances and attempt learning to classify each object pair into correctly-ranked or incorrectly-ranked. Indeed, most existing methods are the pair-wise approaches, including Ranking SVM [10], RankBoost [9], and RankNet [2]. Ranking SVM employs support vector machine (SVM) to classify object pairs in consideration of large margin rank boundaries. RankBoost conducts Boosting to find a combined ranking which minimizes the number of misordered pairs of objects. RankNet defines Cross Entropy as a probabilistic cost function on object pairs and uses a neural network model to optimize the cost function. Finally, the list-wise approaches use a list of ranked objects as training instances and learn to predict the list of objects. For example, ListNet [3] introduces a probabilistic-based list-wise loss function for learning. Neural network and gradient descent are employed to train a list prediction model.

Researchers have investigated learning to rank algorithms in IR applications and obtained promising results. Examples are listed as follows: [20] treated IR as a binary classification of relevance and explored the applicability of discriminative classifiers (such as SVM) to solve the problem. [12] took pairs of documents and their relative preferences derived from click-through data (i.e., the log of links users clicked on in the presented ranking provided by a WWW search engine) as training instances and applied Ranking SVM for learning better retrieval functions. [4] modified the "Hinge Loss" function in Ranking SVM to take into account two essential factors for IR: (1) to have high accuracy on the top-ranked documents, and (2) to avoid training a biased model towards queries with many relevant documents. [26] used SVM and Ranking SVM to address definition search where the retrieved definitional excerpts of a term are ranked according to their likelihood of being good definitions. [28] extended the well-studied SVM selecting sampling technique in classification for learning to rank. [19] proposed a multiple nested ranker approach to re-rank the top scoring documents on the result list. RankNet was applied to learn a new ranking at each iteration.

Other studies using genetic programming (GP) are the closely related works to this paper. [8] presented a learning framework based on GP to help automate the design process of ranking functions. See also [7]. The effects of different fitness functions used in GP have been also examined in [6]. These works tried to combine evidences used in traditional term weighting strategies into a ranking function by GP. In contrast, the goal of this paper – to combine together different types of evidences such as classical IR content features, structure features, and even query-independent features – makes this work quite different.

3. THE PROPOSED GP-BASED LEARNING METHOD

Genetic programming (GP), an evolutionary methodology, is an automated problem-solving system for producing a computer program that is able to perform a user-defined task [15]. In essence, GP evolves a population of individuals (i.e., computer programs) by applying genetic operations, such as crossover and mutation, over a series of generations. The evolution is supposed to eventually generate an individual with the best fitness as the optimal solution to a given task, where the fitness is modeled by a user-defined measure to score the ability of an individual. In recent years, GP has been profitably employed in many application, including circuit layout design [16], robot control [15], and classification [13][17]. Inspired by the success of GP, in this paper, a GP-based method, RankGP, is proposed to handle the task of learning to rank for IR.

3.1 Data Normalization

As shown in Figure 1, a training set, *T*, is a group of query-document pairs, each with a relevance judgment and a vector of features describing the match between the corresponding query and document. For a query collection, $Q = \{q_1, \ldots, q_{|Q|}\}$, a document collection, $D = \{d_1, \ldots, d_{|D|}\}$, a feature set, $F = \{f_1, \ldots, f_{|F|}\}$, and a relevance judgment set, Y = {relevant, non-relevant}, T is formulated as a set of triples by Eq. (1):

$$T = \{((q_i, d_j), (f_1(q_i, d_j), \ldots, f_{|F|}(q_i, d_j)), y_{ij})\} \quad (1)$$

where $(q_i, d_j) \in Q \times D$, $y_{ij} \in Y$, $(f_1(q_i, d_j), \ldots, f_{|F|}(q_i, d_j))$ is a $|F|$-dimensional feature vector in which $f_k(q_i, d_j)$ stands for the feature value for q_i and d_j in terms of f_k.

Before applying RankGP to the training set, a query-based normalization on features is performed in order to normalize all feature values into a range of [0, 1]. For a query q_i, the normalized value of $f_k(q_i, d_j)$ is calculated by Eq. (2), where $\max\{f_k(q_i, d_l)\}$ and $\min\{f_k(q_i, d_l)\}$ are the maximum value and the minimum value of $f_k(q_i, d_l)$ respectively for all $d_l \in D$.

$$f_k(q_i, d_j) = \frac{f_k(q_i, d_j) - \min\{f_k(q_i, d_l)\}}{\max\{f_k(q_i, d_l)\} - \min\{f_k(q_i, d_l)\}} \quad (2)$$

3.2 The Learning Method: RankGP

The proposed GP-based learning method, RankGP, is summarized in Figure 2.

The proposed learning method: RankGP

Input 1: a set of query-document pairs with their feature vectors and relevance judgments (i.e., the training set, T)

Input 2: GP-related parameters: G (# of generations), *PSize* (size of a population), R_c (crossover rate), R_m (mutation rate)

Output: a ranking function, f, which is supposed to associate a real number with a query and a document as their degree of relevance

Learning Procedure:

(1) Randomly initialize a set of individuals as the initial population, P, and create an empty output set, O

(2) Score by a fitness function the performance of each individual in P on T

(3) Put the most fit individual (i.e., the individual with the best fitness) in P into O and meanwhile create a new population by reproducing the most fit individual into it

(4) Generate individuals to fulfill the new population to the size of *PSize* via crossover and mutation

(5) Set P to the new population and repeat Steps (2)–(5) until the number of iterations reaches G

(6) Evaluate the performance of each individual in O by a score function and output the best one

Figure 2. Overview of RankGP.

In RankGP, an individual is a potential ranking function to associate a real number with a query and a document as their degree of relevance. An individual, I, is defined as a functional expression using three components: S_v (variables), S_c (constants), and S_{op} (operators). S_v is a set of symbolic notations, referring to features of the training set T. S_c is a set of predefined real numbers, ranging from 0 to 1. S_{op} is a set of arithmetic operators. Eq. (3) shows the formulation of an individual, I.

$$I = (S_v, S_c, S_{op}) \quad (3)$$

where S_v, S_c, and S_{op} are given by:

$$S_v = \{f_i \mid f_i \in F\},$$
$$S_c = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\},$$
$$S_{op} = \{+, -, \times, /\}.$$

The reasons to adopt only simple arithmetic operators in S_{op} are: (1) it has been shown that these operators are sufficient to achieve good results in classification problems[1] [13], and (2) using these operators helps reduce the computational cost.

In the implementation, I is represented as a binary tree structure, in which an internal node is a binary operator and a leaf node is either a variable or a constant. The maximum number of available nodes of an individual is determined by the depth of the tree, which is defined before the learning process. A tree structure example is provided in Figure 3 for the function, f: $((f_1 + f_2) + (0.3 \times f_3))$.

With regard to the type of ranking functions that RankGP targets at, one can choose a linear or a non-linear function. In this work, only the linear version is used and thus the operator / in S_{op} is discarded.

The fitness function is a function used to evaluate the fitness of an individual (i.e., how good an individual is for the given task). Since the problem on which this paper concentrates is the ranking problem of IR, the fitness function is defined as a widely-used IR measure, MAP[2] (Mean Average Precision) [1] (as described in Section 4.2). For an individual, MAP is computed over all queries, based on the rankings of documents provided by the individual.

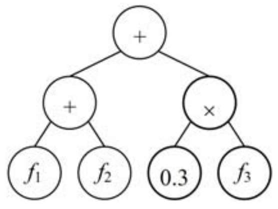

Figure 3. A tree structure example.

The evolution process of RankGP is described as follows. A population is composed of a set of individuals. The initial population of individuals is produced by the ramped half-and-half method [15]. With this method, individuals are generated by a random process. However, it ensures that half of individuals *must* have and the others *might not* have all branches of the maximum tree depth. In the evolution process, three genetic operations, *crossover*, *mutation*, and *reproduction*, are applied to generate a new population. The choice of the operations depends on a predefined probability over these operations. For reproduction, a number of the best individuals in the current generation get through to the next generation unmodified. This is to imitate the

[1] It needs to be verified for learning to rank problems.

[2] Based on the relevance between a query and a document, which is given by a ranking function, an ordering among documents can be established and hence MAP can be calculated.

nature selection principle of survival of the most fit (i.e., the elite). The remaining individuals are produced by either crossover or mutation. During the production, tournament selection [15] is used to bias the fitness. This method first randomly selects a few individuals from the previous generation and returns one individual (for mutation) or two individuals (for crossover) which have the best fitness values. For crossover, two new individuals are produced by exchanging sub-trees of the selected individuals. For mutation, a mutant is created by randomly choosing an internal node of the selected individual and then replacing its whole sub-tree with a randomly generated tree.

When the evolution process ends, the output set, *O*, contains the best individuals obtained from all generations as candidate solutions. To select a good solution as the final output, this paper adopts the following principle: given a validation set, a good individual should have not only a good fitness on the training set but also a good MAP value on the validation set. Hence, a score function, as defined in Eq. (4), is exploited to select the final solution.

$$score(I_j) = F(I_j) + MAP(I_j, V) \qquad (4)$$

In the equation, I_j denotes an individual, $F(I_j)$ is the fitness of I_j, and $MAP(I_j, V)$ stands for the MAP of I_j on the validation set, V.

3.3 Implementation

RankGP has been implemented within an open source GP toolkit, LAGEP[3] [17]. LAGEP implements a recently-developed layer architecture of multi-population genetic programming and an adaptive mutation rate tuning method, called AMRT. The layer architecture of LAGEP works as follows: in one layer, a set of functions are learned based on the given training set. A new training set is constructed using the learned functions. The successive layer then takes the new training set to discover new functions. As for AMRT, it dynamically changes the mutation rate during the learning process. In each generation, AMRT is triggered to increase the mutation rate for the next generation if all individuals in the current generation have similar fitness values. Otherwise, the mutation rate is set unchanged as the initial one. Moreover, AMRT ensures that the mutation rate achieves a maximum of 0.5 in the last generation.

In the current work, RankGP does not adopt either the layer architecture or the design of multi-population GP of LAGEP. Instead, only the single-population GP is used. The AMRT strategy is also employed in RankGP since it has been shown effective in classification problems (see [17]).

4. EXPERIMENT

This section first describes the test datasets and evaluation measures, and then reports the preliminary experimental results.

4.1 The LETOR Benchmark Datasets

The LETOR benchmark datasets[4] [18], released by Microsoft Research Asia for research on learning to rank for IR, are used to conduct experiments in this work. LETOR contains two data sets: OHSUMED and TREC (TD2003 and TD2004), evaluation tools, and two baseline evaluation results. The data sets were created as query-document pairs, each consisting of a vector of features and the corresponding relevance judgment. There are in total 16,140 instances in OHSUMED, 49,171 instances in TD2003 and 74,170 instances in TD2004. The extracted features cover most of the standard IR features, such as classical features (e.g., term frequency, inverse document frequency, and BM25 [24]), and the features proposed in recent SIGIR papers (e.g., HostRank [27], Feature Propagation [23][25], and Topical PageRank [21]). While 25 features were extracted in OHSUMED, 44 features were extracted in both TD2003 and TD2004. The relevance judgments are on three levels in OHSUMED: *definitely*, *possibly* and *not relevant*, and on two levels in TD2003 and TD2004: *relevant* and *not relevant*.

In this paper, the proposed RankGP was only tested on the TREC datasets (i.e., TD2003 and TD2004). Experiments on OHSUMED are left in the future due to the time limitation.

The TREC datasets contain a total of 1,053,110 web pages, together with 11,164,829 hyperlinks. There are 50 queries and 75 queries in TD2003 and TD2004 respectively. The query sets here are exactly the same with those used in topic distillation tasks of the Web track in TREC 2003 and TREC 2004. The TREC committee made judgments to identify whether a page is appropriate to a given query. Three types of features were extracted for learning, including content features, hyperlink features, and hybrid features. Table 1 lists all the features. For an overview of these features and the feature extraction principles, please refer to LETOR.

Table 1. Features extracted in TD2003 and TD2004

Feature type	Feature name	Number of features
Content	dl [1]	4
	tf [1]	4
	idf [1]	4
	tfidf [1]	4
	BM25 [24]	4
	LMIR [29]	9
Hyperlink	HostRank [27]	1
	PageRank [22]	1
	Topical PageRank [21]	1
	HITS [14]	2
	Topical HITS [21]	2
Hybrid	Sitemap-based relevance propagation [23]	2
	Hyperlink-based relevance propagation [25]	6
		Total: 44

4.2 Evaluation Measures

The evaluation tool, provided by LETOR [18], is utilized to evaluate the effectiveness of the proposed RankGP. The tool supports three common IR evaluation measures, all are briefed as follows.

- *P@n* (Precision at Position *n*) [1]

For a given query, its precision of the top *n* results of the ranking list is defined as Eq. (5):

[3] http://www.cis.nctu.edu.tw/~gis91815/lagep/lagep.html.
[4] http://research.microsoft.com/users/tyliu/LETOR/.

$$P@n = \frac{\#\text{ of relevant docs in top } n \text{ results}}{n} \quad (5)$$

- *MAP* (Mean Average Precision) [1]

Given a query, its average precision is computed by Eq. (6) where N is the number of retrieved documents and $rel(n)$ is either 1 or 0, indicating the n-th document is relevant or not to the query. MAP is obtained as the mean of average precisions over a set of queries.

$$AP = \frac{\sum_{n=1}^{N} P@n \times rel(n)}{\#\text{ of relevant docs for this query}} \quad (6)$$

- *NDCG* (Normalized Discount Cumulative Gain) [11]

For a query, the NDCG of its ranking list at position m is calculated as Eq. (7):

$$NDCG@m = Z_m \sum_{j=1}^{m} \frac{2^{r(j)} - 1}{\log(1+j)} \quad (7)$$

In the equation, $r(j)$ is the rating of the j-th document in the list, and the normalization constant Z_m is set so that the perfect list gets a NDCG of 1. $r(j)$ is set to 1 when the j-th document is relevant to the query and is set to 0 when the document is irrelevant.

For comparisons, this paper reports NDCG@1, …, NDCG@10, P@1, …, P@10, as well as MAP.

4.3 Experimental Settings and Baselines

In LETOR [18], both TD2003 and TD2004 are already partitioned for users to conduct 5-fold cross validation. For each fold, three subsets are used for training, one subset for validation, and the other one for testing. The score reported in this paper is the average of those in the five folds.

One non-learning baseline method, BM25 [24], and two learning-based baseline methods, Ranking SVM [10] (denoted as RankSVM in the next section) and RankBoost [9], were also evaluated on the datasets as comparisons to the proposed RankGP. The evaluation results of the baselines reported in this paper are excerpted from LETOR. For the learning-based baselines, the query-based normalization on features (see Section 3.1) was similarly conducted before applying these state-of-the-art learning algorithms. The linear version was used in Ranking SVM and each single feature was defined as a weak learner in RankBoost. Please refer to LETOR for the settings and the optimization methods of parameters for these baselines.

Table 2 lists the parameter settings of RankGP, all were set empirically in the experiments. The type of the ranking function that RankGP targets at is assumed linear. The number of generations, crossover rate, mutation rate, and reproduction are set according to [17]. The tree depth is set in order to at least cover the case that leaf nodes contain 44 features and 44 constants. Note that a full binary tree with a depth of 8 has 128 leaf nodes, which is enough to include 44+44=88 nodes. We acknowledge that non-linear functions, different settings of RankGP, and the effect of the use of AMRT [17] for mutation rate tuning should be also explored but leave them to future work.

In the experiments, RankGP was performed 10 times. In each run, 5-fold cross validation was conducted and an average score was obtained. The reported score of RankGP is the average of those in the 10 runs. This is to reduce the effect of the phenomenon: the initialization of the first population by a random process might accordingly affect the final results [8].

Table 2. Parameter settings of RankGP

Parameter	Value
Function type	Linear function
# of generations (G)	100
Population size ($PSize$)	600
Tree depth	8 (a total of 255 nodes in a full tree)
Tournament size	5
Crossover rate (R_c)	0.95
Mutation rate (R_m)	Start with 0.05 and is dynamically changed by AMRT [17] during the evolution to a maximum of 0.5 in the last generation.
Reproduction	Only the most fit individual in the current generation gets through to the next generation unmodified

4.4 Results

In this section, two evaluation results of the proposed RankGP are given: RankGP (Avg.), as mentioned in Section 4.3, is the average score of those in 10 runs of 5-fold cross validation, while RankGP (Best) is the average score of the best score of each fold in 10 runs.

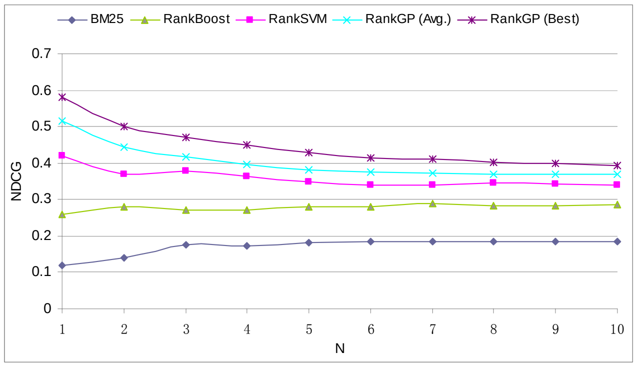

Figure 4. NDCG@N on TD2003.

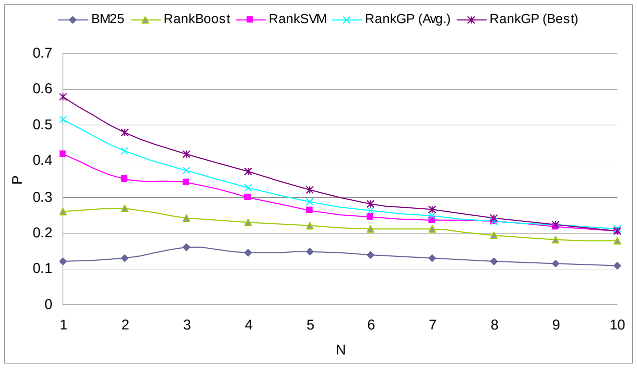

Figure 5. P@N on TD2003.

Figures 4-6 give the results on TD2003. It is obvious that RankGP (Avg.) outperforms BM25, RankBoost and RankSVM in terms of all measures. In Figure 4, RankGP (Avg.) obtains improvements over the best baseline, RankSVM, for NDCG@1-10, especially NDCG@1 and NDCG@2. The relative improvements are about 22.86% and 19.73% respectively. Similar phenomenon can be observed in Figure 5 with respect to P@1 and P@2. However, the P@N values when N is greater than 3 tend towards those of RankSVM and are almost the same after N=8. Figure 6 shows that RankGP (Avg.) has an MAP of 0.283778, which is better than that of BM25, that of RankSVM, and that of RankBoost in about 126.11%, 10.67%, and 33.55% respectively. Based on the results, a conclusion is made that the proposed RankGP is good at ranking relevant documents at the very top positions (such as the first and the second places) to get a high MAP. Finally, Table 3 lists the paired *t*-test results on TD2003 in terms of MAP. This table indicates that the improvements of RankGP (Avg.) over BM25 and RankBoost are statistically significant at α=0.05 (p=0.0002 and p=0.0475 respectively). Although the improvement of RankGP (Avg.) over RankSVM is not significant enough (p=0.1648), the improvement becomes significant at α=0.10 (p=0.0768) when RankGP (Best) is considered.

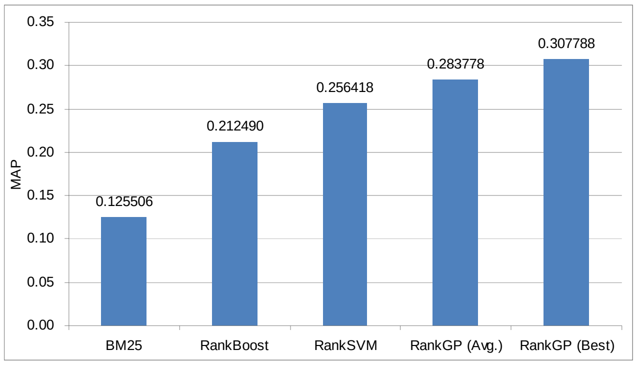

Figure 6. MAP on TD2003.

Table 3. Paired *t*-test results on TD2003 in terms of MAP

p-value	BM25	RankSVM	RankBoost
RankGP (Avg.)	0.0002	0.1648	0.0475
RankGP (Best)	0.0003	0.0768	0.0405

Figures 7-9 show the results on TD2004. It can be seen that RankGP (Avg.) is superior to BM25 and RankSVM, but is inferior to RankBoost. However, RankGP (Best) outperforms BM25, RankBoost and RankSVM in terms of all measures. In Figures 7-8, RankGP (Avg.) obtains close values to those of RankBoost in terms of NDCG@4 and NDCG@5, as well as P@4-10. RankGP (Best) obtains improvements over the best baseline, RankBoost, for NDCG@1-5 and P@1-5, but tends towards RankBoost for both NDCG and P scores after N=6. As for MAP, Figure 9 shows that RankGP (Avg.) outperforms BM25 and RankSVM in about 28.84% and 3.57% respectively, but is beaten by RankBoost in about 5.66%. RankGP (Best) performs better than BM25, RankSVM, and RankBoost in about 42.98%, 14.93%, and 5.03% respectively. Finally, Table 4 gives the paired *t*-test results on TD2004 in terms of MAP. While the improvement of RankGP (Avg.) over BM25 is statistically significant at α=0.05 (p=0.0086), the improvement over RankSVM is not significant (p=0.2274). In consideration of RankGP (Best), the improvements over BM25 and RankSVM are statistically significant at α=0.05 (p=0.0004 and p=0.0199 respectively). However, the improvement of RankGP (Best) over RankBoost is not significant (p=0.1711).

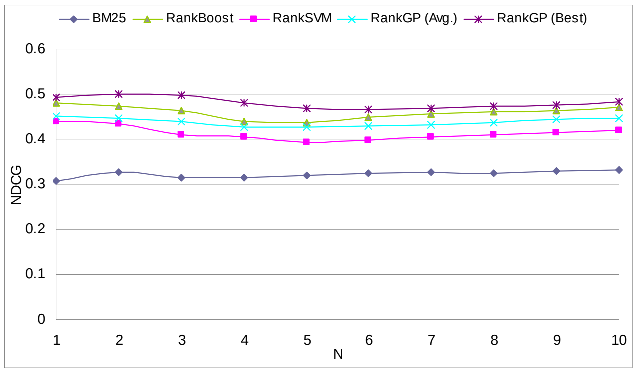

Figure 7. NDCG@N on TD2004.

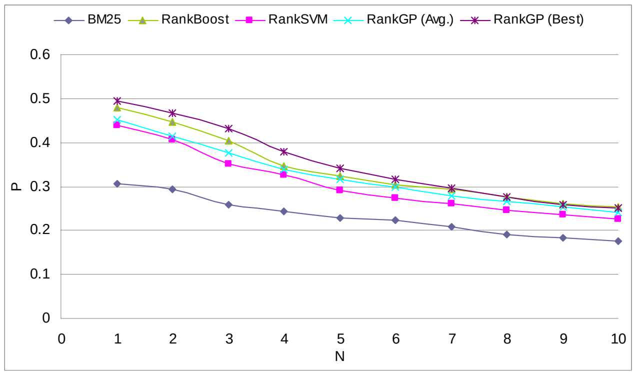

Figure 8. P@N on TD2004.

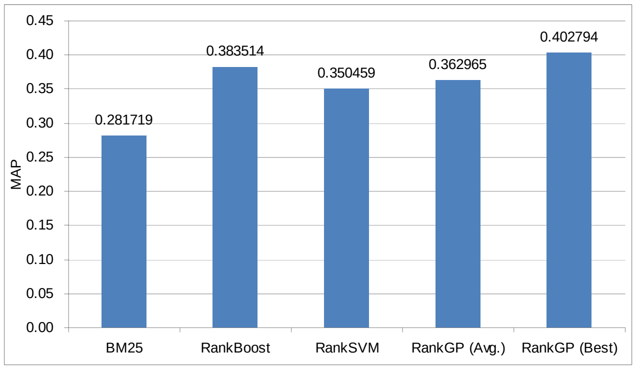

Figure 9. MAP on TD2004.

Table 4. Paired *t*-test results on TD2004 in terms of MAP

p-value	BM25	RankSVM	RankBoost
RankGP (Avg.)	0.0086	0.2274	0.0838*
RankGP (Best)	0.0004	0.0199	0.1711

*: RankBoost is better than RankGP (Avg.)

To sum up, RankGP behaves differently on different datasets. Similar phenomenon happens to RankBoost as well. However, the results show that the average performance of RankGP, i.e., RankGP (Avg.), outperforms RankSVM and is competitive to RankBoost. Regarding TD2003, RankGP (Avg.), RankSVM, and RankBoost are ranked the best, the second, and the worst respectively. When considering TD2004, the ranking becomes the second, the worst, and the best, respectively. Although RankGP (Avg.) is roughly comparable to RankBoost on TD2004, RankGP (Best) is superior to RankBoost. We conjecture that RankGP (Avg.) on TD2004 might be improved if more learning generations and a bigger size of the population are given. This issue is left to future work.

4.5 Discussion

The reasons why GP is employed in this work include: (1) the use of functional expressions to represent an individual is successful in GP (see [13] and [17]). No doubt, it is natural to represent a ranking function for IR in a functional expression. (2) To find a near or the exact global optimal solution by GP is possible [15]. In comparison with other methods (e.g., Ranking SVM [10] and RankBoost [9]) which decouple the problem into individual pair-wise evaluations, the main advantage of RankGP is that any goodness criteria (e.g., MAP in this paper) can be directly optimized, since the evolution process is discrete. However, the computational cost of RankGP is much huge and is not relative to other approaches. Running the current implementation of RankGP with the parameter settings listed in Table 2, it costs us approximately 35 hours to conduct one run of 5-fold cross validation on TD2003 with a PC, which is equipped with an 1.8GHz Intel Core 2 CPU and 2GB memory. Thus, it is the user's choice to determine whether RankGP or other approaches will be adopted. In general, it is suggested to apply RankGP when the main concern is to find a potential optimal solution, provided that computing resources and time are not limited.

In this work, RankGP aims to find the optimal linear ranking function, which could be also directly optimized using greedy search algorithms (e.g., conjugate gradient descent). However, it is easy to extend RankGP with more complex operators in S_{op} so as to generate a wider class of non-linear ranking functions. This makes the benefits of RankGP more clear since a non-linear function can not be directly optimized using greedy search.

5. CONCLUSION

In this paper, a learning method, RankGP, is proposed to address the task of learning to rank for IR. RankGP employs genetic programming to learn an effective ranking function by combining various types of evidences in IR, including content features, structure features, and query-independent features. Experiments were conducted to evaluate the performance of RankGP using TD2003 and TD2004 of the LETOR benchmark datasets. One non-learning method, BM25, and two state-of-the-art learning methods, Ranking SVM and RankBoost, were compared with RankGP. The results show that RankGP outperforms BM25 and Ranking SVM, and is competitive to RankBoost.

The main contributions of this work are threefold. First, this paper offers a GP-based learning method for learning ranking functions. The second contribution is the direct use of IR measure, Mean Average Precision, as the internal performance measure of the learning method, in comparison with the major approaches proposed in the field, which transform the ranking problem into a classification problem of determining the preference of two objects and use loss functions to train a ranking model. Finally, the proposed method is verified in a case study with the LETOR benchmark datasets.

6. FUTURE WORK

In this work, the selection of features is not conducted before applying RankGP. It should be important to study the relationships between distinct features in order to develop a feature selection method for the ranking problem.

Future works will also continue to compare the use of linear and non-linear functions in RankGP, as well as to investigate the performance of different settings of the algorithm. The effect of the AMRT strategy for mutation rate tuning needs to be further studied on the ranking problem as well. Another interesting issue is to know whether more complex operators, such as log, in S_{op} will help discover a better ranking function.

ACKNOWLEDGEMENTS

This work was supported by the National Science Council (grant number: NSC-95-2221-E-259-044). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors only and do not necessarily reflect the viewpoints of the National Science Council.

REFERENCES

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

[2] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning* (Bonn, Germany. 2005).

[3] Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., and Li, H. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning* (Corvallis, OR. 2007).

[4] Cao, Y. B., Xu, J., Liu, T. Y., Li, H., Huang, Y. L., and Hon, H. W. Adaptive ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA. 2006).

[5] Crammer, K., and Singer, Y. PRanking with ranking. *Advances in Neural Information Processing Systems*, 14: 641-647. 2002.

[6] Fan, W., Fox, E. A., Pathak, P., and Wu, H. The effects of fitness functions on genetic programming-based ranking discovery for web search. *Journal of Management Information Systems*, 21(4): 37-56. 2005.

[7] Fan, W., Gordon, M. D., and Pathak, P. A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing and Management*, 40(4): 587-602. 2004.

[8] Fan, W., Gordon, M. D., and Pathak, P. 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] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4: 933-969. 2003.

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

[11] Jarvelin, K., and Kekalainen, J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4): 422-446. 2002.

[12] Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM Conference on Knowledge Discovery and Data Mining* (Edmonton, Alberta, Canada. 2002).

[13] Kishore, J. K., Patnaik, L. M., Mani, V., and Agrawal, V. K. Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3): 242-258. 2000.

[14] Kleinberg, J. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5): 604-622. 1999.

[15] Koza, J. R. *Genetic Programming: On the programming of computers by means of nature selection*. MIT Press, Cambridge, MA, 1992.

[16] Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J. and Lanza, *G. Genetic programming IV: Routine Human-competitive Machine Intelligence*. Kluwer Academic Publishers. 2003.

[17] Lin, J. Y., Ke, H. R., Chien, B. C., and Yang, W. P. Designing a classifier by a layered multi-population genetic programming approach. *Pattern Recognition*, 40(8): 2211-2225. 2007.

[18] Liu, T. Y., Xu, J., Qin, T., Xiong, W., and Li, H. LETOR: Benchmarking learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval* (Amsterdam, Netherlands, 2007).

[19] Matveeva, I., Burges, C., Burkard, T., Laucius, A., and Wong, L. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA. 2006).

[20] Nallapati, R. Discriminative models for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Sheffield, South Yorkshire, UK. 2004).

[21] Nie, L., Davison, B. D., and Qi, X. Topical link analysis for web search. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA. 2006).

[22] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank citation ranking: Bring order to the Web. Technical Report, Stanford University, 1998.

[23] Qin, T., Liu, T. Y., Zhang, X. D., Chen, Z., and Ma, W. Y. A study of relevance propagation for web search. In P*roceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Salvador, Brazil. 2005).

[24] Robertson, S. E. Overview of the Okapi projects. *Journal of Documentation*, 53(1): 3-7. 1997.

[25] Shakery, A., Zhai, C. X. Relevance propagation for topic distillation UIUC TREC 2003 Web Track experiments. In *Proceedings of TREC 2003*.

[26] Xu, J., Cao, Y. B., Li, H., and Zhao, M. Ranking definitions with supervised learning methods. In *Proceedings of the 14th International World Wide Web Conference* (Chiba, Japan. 2005).

[27] Xue, G. R., Yang, Q., Zeng, H. J., Yu, Y., and Chen, Z. Exploring the hierarchical structure for link analysis. In P*roceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Salvador, Brazil. 2005).

[28] Yu, H. SVM selective sampling for ranking with application to data retrieval. In *Proceedings of the 11st ACM Conference on Knowledge Discovery and Data Mining* (Chicago, IL. 2005).

[29] Zhai, C., and Lafferty, J. A study of smoothing methods for language models applied to Ad Hoc in formation retrieval. In P*roceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New Orleans, LA. 2001).