LEARNING RANKING FUNCTIONS FOR INFORMATION RETRIEVAL USING LAYERED MULTI-POPULATION GENETIC PROGRAMMING

***Jen-Yuan Yeh*[1], *Jung-Yi Lin*[*2]**

[1]Dept. of Operation, Visitor Service, Collection and Information Management,
National Museum of Natural Science, Taichung 40453, Taiwan

Innovation Digital System Business Group,
[2]Hon Hai Precision Industry Company Ltd., Taipei City 11492, Taiwan

Email: jenyuan@mail.nmns.edu.tw[1], jungyilin@gmail.com[2]

[*2] Corresponding author. Tel.: +886 2 2799 6111×65220.

ABSTRACT

Ranking plays a key role in many applications, such as document retrieval, recommendation, question answering, and machine translation. In practice, a ranking function (or model) is exploited to determine the rank-order relations between objects, with respect to a particular criterion. In this paper, a layered multi-population genetic programming based method, known as RankMGP, is proposed to learn ranking functions for document retrieval by incorporating various types of retrieval models into a singular one with high effectiveness. RankMGP represents a potential solution (i.e., a ranking function) as an individual in a population of genetic programming and aims to directly optimize information retrieval evaluation measures in the evolution process. Overall, RankMGP consists of a set of layers and a sequential workflow running through the layers. In one layer, multiple populations evolve independently to generate a set of the best individuals. When the evolution process is completed, a new training dataset is created using the best individuals and the input training set of the layer. Then, the populations in the next layer evolve with the new training dataset. In the final layer, the best individual is obtained as the output ranking function. The proposed method is evaluated using the LETOR datasets and is found to be superior to classical information retrieval models, such as Okapi BM25. It is also statistically competitive with the state-of-the-art methods, including Ranking SVM, ListNet, AdaRank and RankBoost.

Keywords: learning to rank for information retrieval, ranking function, supervised learning, layered multi-population genetic programming, LAGEP, LETOR

1.0 INTRODUCTION

One central problem of information retrieval (IR) is the issue of determining which documents are relevant to the user's information needs, and which are not (i.e., finding the potentially relevant documents) [3]. In practice, it is usually addressed as a ranking problem, whose goal is, according to the degree of relevance (or similarity) between each document and the user's query, to define a total order of documents that ranks relevant documents in higher positions on the retrieved list than irrelevant ones.

Traditional IR models, including the Boolean model, the vector space model, and the probabilistic model, are developed based on the bag-of-words model. In short, a document is decomposed into keywords (i.e., index terms) and a ranking function (or retrieval function) is defined to associate a relevance degree with the document and query [3]. The aforementioned models are typically realized in an *unsupervised* manner and thus, the parameters of underlying ranking functions are usually tuned empirically. However, manual tuning suffers high costs and sometimes leads to over-fitting, especially when the functions are carefully tuned to fit particular needs [30].

Nowadays, as increasingly many IR results are accompanied by relevance judgments (e.g., query and click-through logs), *supervised* learning-based methods, referred to as "learning to rank (LTR)" methods, e.g., Pranking [13], Ranking SVM [23][26], ListNet [8], AdaRank [56], RankBoost [21] and RankNet [7], have been devoted to automatically learning an effective ranking function from training data, for tuning parameters or for incorporating distinct retrieval models into a singular one with high effectiveness.

Since the performance of IR systems is generally evaluated in terms of measures, such as Mean Average Precision (MAP) [3] and Normalized Discounted Cumulative Gain (NDCG) [25], LTR methods are practically designed to optimize loss functions loosely related to IR evaluation measures [57]. A straightforward way of efficiently finding a solution by directly optimizing evaluation measures is to use genetic programming (GP). This paper proposes a GP-based LTR method, known as RankMGP, to learn ranking functions for document retrieval by incorporating various types of IR evidences, such as classical content features, structure features and query-independent features. RankMGP represents a potential solution (i.e., a ranking function) as an individual in a population of GP. Instead of using traditional GP that works with only a single population, RankMGP utilizes multi-population GP and a layered architecture that has proven effective in [29] to arrange multiple populations. Overall, RankMGP consists of a set of layers and a sequential workflow running through the layers. In one layer, multiple populations evolve independently to generate a set of the best individuals. In each generation of evolution, a novel fitness function, which is modelled as the weighted average of NDCG scores, is exploited to measure the performance of each individual in each population. When the evolution process is completed, a new training dataset is created using the best individuals and the input training set of the layer. Then, populations in the next layer evolve with the new training dataset. In the final layer, the best individual is obtained as the output ranking function.

The main contributions of this study are summarized as below:

1. The use of layered multi-population GP in the context of LTR is investigated, and a novel learning method, known as RankMGP, is proposed. In addition, a new fitness function, which is modelled as the weighted average of NDCG scores, is introduced;
2. RankMGP is evaluated in a case study using the LETOR datasets. The results show that RankMGP is superior to classical IR models, such as Okapi BM25 [40] and LMIR [62]. It is also suggested that RankMGP obtains statistically competitive results as compared to the state-of-the-art methods, including Ranking SVM [23][26], ListNet [8], AdaRank [56] and RankBoost [21];
3. In-depth discussions are given from various perspectives behind the design and the effectiveness of RankMGP, e.g., the pros and cons, and the learning behaviors over layers.

The rest of this paper is organized as follows. Section 1.1 elaborates the general paradigm of LTR for IR. Section 2.0 provides a brief review of related works. Section 3.0 introduces the proposed learning method, RankMGP. The experimental results and discussions are provided in Sections 4.0 and 5.0, respectively. Finally, Section 6.0 concludes this paper and points out possible directions for further research.

1.1. The general paradigm of LTR for IR

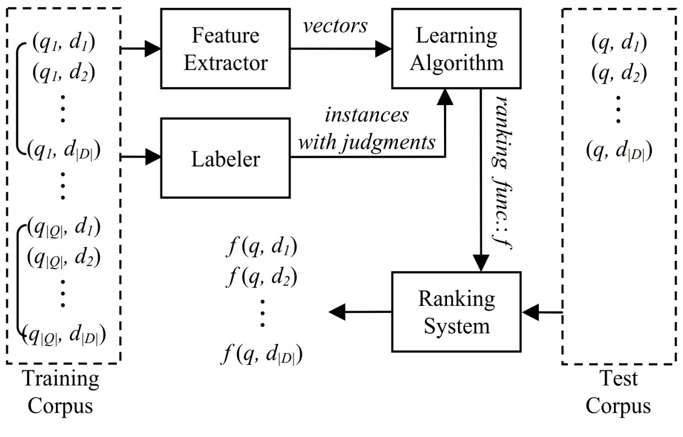


Fig. 1. The general paradigm of LTR for IR [58].

Fig. 1 presents the general paradigm of LTR for IR. The learning process consists of two phases, namely, *training* and *test*. First, the training phase is introduced. Given a query collection, $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$, and a document set, $D = \{d_1, d_2, \ldots, d_{|D|}\}$, a training instance is a query-document pair, i.e., $(q_i, d_j) \in Q \times D$, upon which a relevance judgment indicating the relationship between q_i and d_j is assigned by a labeller. The relevance judgment can be (1) a class label, e.g., relevant or non-relevant; (2) a rating, e.g., 3-star rating scaling from 0 to 2 for non-relevant, possibly relevant, and definitely relevant; (3) an order, e.g., k, meaning that d_j is ranked at the k-th position of the ordering of documents when q_i is considered; and (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 describes the match between q_i and d_j. Such features can be classical IR models (e.g., term frequency, inverse document frequency and Okapi BM25 [40]) or newly developed models (e.g., Feature Propagation [37] and Topical PageRank [33]). The inputs to the learning algorithm comprise training instances, their feature vectors and the corresponding relevance judgments. The output is a ranking function (or model) f, where $f(q_i, d_j)$ is supposed to give the "true" relevance judgment for q_i and d_j. The learning algorithm attempts to learn a ranking function, such that a performance measure (e.g., classification accuracy, error rate, and MAP) with respect to the output relevance judgments can be optimized. In the test phase, the learned ranking function is applied to judge the relevance between each document d_i in D and a new query q.

2.0 RELATED WORKS

The previous works are grouped into three categories [30]: (1) the *point-wise* approach; (2) the *pair-wise* approach; and (3) the *list-wise* approach.[1] In the *point-wise* approach, each training instance is associated with a class or a rating. The learning process finds a model that maps instances into rates or classes close to their true values. The point-wise approach can be further divided into three sub-categories, namely, regression-based (e.g., [12]), classification-based (e.g., [32] and McRank [28]), and ordinal regression-based (e.g., [43] and Pranking [13]). A typical example is Pranking [13], which trains a perceptron model to directly maintain a totally-ordered set via projects. Another one is McRank [28], which defines a 5-star rating scaling from 0 to 4, where 0 corresponds to "poor" relevance and 4 corresponds to "perfect" relevance. It casts the ranking problem as multiple classifications in accordance with an observation that perfect classifications lead to perfect DCG (Discounted Cumulative Gain) [25] scores. The classification model is learned via gradient boosting.

The *pair-wise* approach takes pairs of objects and their relative preferences as training instances and learns to classify each object pair as correctly-ranked or incorrectly-ranked. Indeed, most existing methods are pair-wise approaches, e.g., Ranking SVM [23][26], RankBoost [21] and RankNet [7]. Ranking SVM [23][26] employs Support Vector Machines (SVM) to classify object pairs in consideration of large margin rank boundaries. Both RankBoost [21] and QBrank [63] conduct boosting to find a combined ranking, which minimizes the number of mis-ordered pairs of objects. RankNet [7] defines cross entropy as a probabilistic cost function on object pairs and uses a neural network model to optimize the cost function. LambdaRank [6] also employs neural network but uses gradient based on NDCG smoothed by the RankNet loss (see LambdaMART [5], the boosted tree version of LambdaRank). FRank [48] adopts Fidelity to measure loss of ranking and uses a generalized additive model to minimize the Fidelity loss. Semi-RankSVM [35] extends Ranking SVM by a graph-based regularized algorithm to learn the ranking function that minimizes the least squares ranking loss.

Finally, the *list-wise* approach uses a list of ranked objects as training instances and learns to predict the list of objects. There are two sub-categories which are, respectively, based on the direct optimization of IR evaluation measures (e.g., SoftRank [46], AdaRank [56], and $\text{SVM}^{\Delta}_{map}$ [61]) and the minimization of list-wise ranking losses (e.g., ListNet [8], and ListMLE [54]). Examples are briefed as follows. ListNet [8] introduces a probabilistic-based list-wise loss function for learning. Neural network and gradient descent are adopted to train a list prediction model. In ListMLE [54], the likelihood loss is used as the surrogate for IR evaluation measures. It is worth noting that RankMGP belongs to the family of list-wise approaches since it directly predicts the ranking permutation.

[1] Please refer to [30] which gives a formal definition of the problem and provides a comprehensive survey of the literature.

More examples are described as follows: [32] treats IR as a binary classification of relevance and explores the applicability of discriminative classifiers to solve the problem. [26] takes pairs of documents and their relative preferences derived from click-through data as training instances and applies Ranking SVM for learning better retrieval functions. [9] modifies the “Hinge Loss” function in Ranking SVM to consider 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. [55] uses 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. [59] extends the well-studied SVM selecting sampling technique in classification for LTR. [31] proposes a multiple nested ranker approach to re-rank the top scoring documents of the result list, in which RankNet is applied to learn a new ranking at each iteration. $\text{SVM}_{map}^{\Delta}$ [61] is a SVM-based learning algorithm that efficiently finds a globally optimal solution to a straightforward relaxation of MAP. RankCosine [38] uses cosine similarity between the ranking list and the ground truth as a query-level loss function. AdaRank [56], a novel learning algorithm within the framework of boosting, repeatedly constructs “weak rankers” and finally linearly combines the weak rankers for making ranking predictions. RV-SVM [60] develops a 1-norm Ranking SVM, which is based on 1-norm objective function, for faster training using much less support vectors than the standard Ranking SVM.

On the other hand, other research directions, which are receiving increasing attention in recent years, include [10]: *online learning to rank* for quickly learning the best re-ranking of the top position of the original ranked list based on real-time user click feedback [11][42]; *large-scale learning to rank* which leverages both the learning theory and computational theory for ranking when facing large-scale training data [39][45][50]; *learning to rank for diversity* aims to optimize not only for relevancy, but also for diversity (i.e., for minimum redundancy) by taking into account document similarity and ranking context [41][44]; and *robust learning to rank* optimizes the tradeoffs between model effectiveness and robustness for real-world retrieval scenarios [49].

2.1. Evolutionary learning techniques based methods

Owing to the freedom that evolutionary learning techniques offer in the definition of the problem and representation of possible solutions [27], numerous attempts have been made at applying such techniques, in particular GP, for discovering IR ranking functions by the direct optimization of non-convex objective functions of IR evaluation measures. Prior studies concentrate on automatically generating term-weighting schemes. For instance, [34] employs GP to perform a search for new tf-idf like schemes. [18] presents a GP-based learning framework to automate the design of ranking functions (see also [17][19]). [47] exploits GP to develop new general purpose ranking functions based on primitive atomic features. Four baseline ranking functions, namely, inner product, cosine measure, probability measure and Okapi BM25 [40] are added in the initial population to guarantee the worst performance of an individual is as good as the baselines. [14] uses GP to evolve term-weighting schemes in an adhoc IR model. The evolution of the entire term-weighting scheme is divided into three phases to learn a global scheme, a term-frequency scheme and a normalization scheme. [2] introduces a combined component approach to train collection-adapted ranking functions in which atomic term weighting components are combined using GP.

Instead of taking into account primitive atomic IR features, later studies learn ranking functions by incorporating distinct retrieval models into a singular one with high effectiveness. RankGP [58] is one of the first GP-based methods that evolves ranking functions with various types of IR models, including content features, structure features and query-independent features. It adopts single-population GP to directly optimize MAP for a linear ranking function. AdaGP-Rank [53] uses RankGP to develop linear ranking functions as weak rankers, while the boosting procedure in AdaRank is applied to assist the evolved ranking functions concentrate on the most ‘hard’ queries. RankGPES [24] enhances RankGP with evolution strategies to optimize parameters by different selection schemes. Besides GP, the uses of other evolutionary learning techniques have been explored. [51] presents RankIP, a ranking function discovery method using immune programming for its high diversity. RankDE [4] is the first LTR method that uses differential evolution. It directly optimizes MAP without requiring any convex approximations. [15] proposes Swarm-Rank, the first LTR method that applies particle swarm optimization. The method learns ranking functions by optimizing the combination of various ranking models where MAP is directly maximized. RankPSO [1] also uses particle swarm optimization to construct ranking functions by directly optimizing IR evaluation measures. CCRank [52], the first parallel framework for LTR, aims to significantly improve the learning efficiency while maintaining accuracy. CCRank is developed with cooperative coevolution, a divide-and-conquer framework that has demonstrated high promise in function optimization with large search space and complex structures.

2.2. Comparison between evolutionary learning techniques based related works and this work

Pioneer evolutionary-based studies devote to automate the generation of term-weighting schemes by combining primitive atomic evidences of terms, documents and queries. In contrast, RankMGP learns ranking functions by incorporating the state-of-the-art IR retrieval models.

Indeed, RankMGP is essentially an extension of RankGP [58]. While RankGP outputs linear functions using traditional single-population GP with MAP being the fitness, RankMGP targets non-linear functions with complex operators using layered multi-population GP and introduces a new fitness function modelled as the weighted average of NDCG scores. Similar to RankGP [58], RankIP [51], RankDE [4], Swarm-Rank [15] and CCRank [52], RankMGP also belongs to the family of evolutionary-based LTR methods. However, this work is quite distinct from the others due to the underlying learning algorithm, the design of fitness function and the target function type. Table 1 presents the differences between RankMGP and other evolutionary-based LTR methods.

Table 1. Comparisons between RankMGP and other evolutionary-based LTR methods.

Method	Learning techniques	Fitness	Function Type
RankMGP (this work)	layered multi-population GP	the weighted average of NDCG scores	non-linear
RankGP [58]	single-population GP	MAP	linear
AdaGP-Rank [53]	single-population GP within a boosting procedure	MAP	linear
RankGPES [24]	single-population GP with evolution strategies	MAP; NDCG	linear
RankIP [51]	immune programming	MAP; NDCG	non-linear
RankDE [4]	differential evolution	MAP	linear
Swarm-Rank [15]	particle swarm optimization	MAP	linear
RankPSO [1]	particle swarm optimization	the summation of differences of IR evaluation measures between the ground truth ranking and the output ranking	linear
CCRank [52]	a parallel evolution framework based on cooperative coevolution	MAP; NDCG	non-linear

3.0 THE PROPOSED GP-BASED LEARNING METHOD

RankMGP is developed with LAGEP (Layered Architecture GEnetic Programming) [29]. LAGEP discovers discriminant functions for binary classification using layered multi-population GP. It has proven effective in real-world medical problems and is capable of generating highly accurate discriminant functions efficiently. In contrast, the ranking problem that we address in this work is not a classification problem. A classification problem considers the correct class that an object belongs to. However, a ranking problem is concerned about the ordering relations between objects given a particular criterion. In the context of IR, this is document classification versus document ranking regarding a given query. To utilize LAGEP for RankMGP, the following steps are proposed: (1) viewing the output function as a ranking function that assesses the relationship between a document and a query, where a high value implies the document is at a top position in the document list; and (2) introducing IR evaluation measures as fitness functions to be directly optimized.

3.1. The training set and query-level feature normalization

Given a query collection, $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$; a document set, $D = \{d_1, d_2, \ldots, d_{|D|}\}$; a feature set, $F = \{f_1, f_2,$

$\ldots, f_{|F|}\}$; and a relevance judgment set, $Y = \{$*definitely relevant*, *possibly relevant*, *not relevant*$\}$,[2] the training set, T, is formulated as a set of triples:

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

where $(q_i, d_j) \in Q \times D$, $y_{ij} \in Y$ and $(f_1(q_i, d_j), \ldots, f_{|F|}(q_i, d_j))$ 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.

Consider that, for different queries, the absolute values of a feature might not be comparable owing to the different ranges of values. For fair comparisons, a query-level feature normalization needs to be conducted to map feature values into a common range of [0, 1]. For a query q_i, the normalized value of $f_k(q_i, d_j)$ is calculated via *min-max* normalization, as shown in Eq. (2):[3]

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

where $\max\{f_k(q_i, d_l)\}$ and $\min\{f_k(q_i, d_l)\}$ are the maximum and the minimum values of $f_k(q_i, d_l)$, respectively, for all $d_l \in D$.

3.2. The proposed learning method: RankMGP

Fig. 2 presents the layered multi-population GP architecture of RankMGP. RankMGP consists of a set of layers and a sequential workflow running through the layers. In one layer, multiple populations evolve independently to generate a set of the best individuals. When the evolution process is completed, a new training dataset is created using the best individuals and the input training set of the layer. Then, populations in the next layer evolve with the new training dataset. In the final layer, the best individual is obtained as the output ranking function.

Eq. (3) provides the general encapsulation of RankMGP:

$$\text{RankMGP} = \{ L_i \mid i = 1, 2, \ldots, \Gamma \}, \tag{3}$$

where Γ is the number of layers. A layer in RankMGP is composed of a training set, a variable set and a set of populations, as formulated in Eq. (4):

$$L_i = (T_i, S_v^i, (P_1, P_2, \ldots, P_{l(i)})), \tag{4}$$

where P_i implies a population, $l(i)$ is the total number of populations in L_i, S_v^i denotes the variables used to construct a functional expression and T_i represents the training set inputted to populations. Note that L_i is a multi-population GP model, in which several populations evolve concurrently, each to discover an optimal solution. Two multi-population GP models have proven successful, namely, parallel and distributed GP (PADGP) [36] and isolated multi-population GP (IMGP) [20]. For simplicity, the IMGP scheme is employed. In such a way, each population evolves independently and no individual migrates from one population to another.

The individual with the best fitness subsists in each population as the evolution finishes. Denoting $\{\Lambda_{i1}, \Lambda_{i2}, \ldots, \Lambda_{il(i)}\}$ as the set of the best individuals collected from the populations, a new training set, T_{i+1}, and a new variable set, S_v^{i+1}, are defined as follows:

$$T_{i+1} = \{ t_{(i+1)j} \mid t_{(i+1)j} = (\Lambda_{i1}(t_{ij}), \Lambda_{i2}(t_{ij}), \ldots, \Lambda_{il(i)}(t_{ij})), t_{ij} \in T_i \} \text{ and} \tag{5}$$

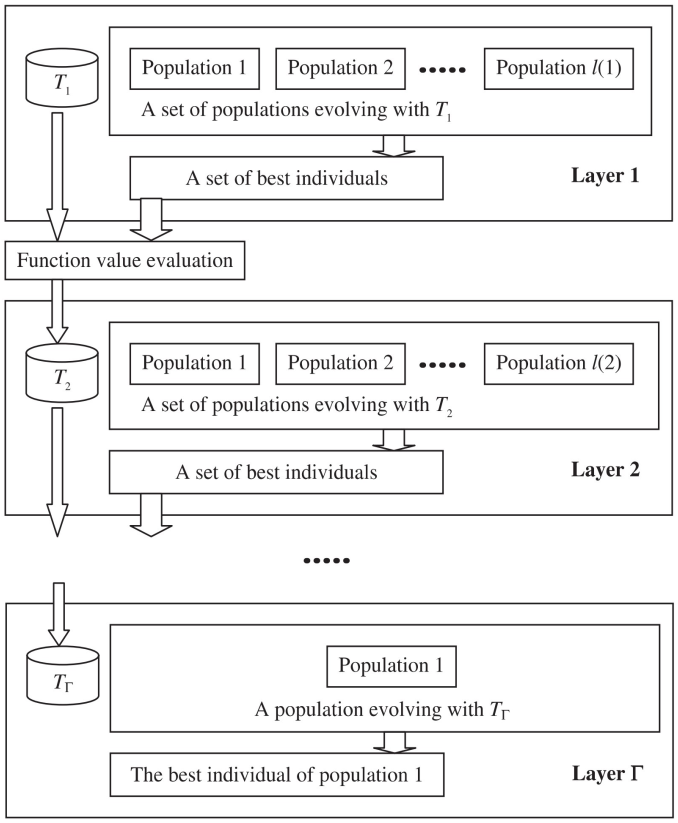
$$S_v^{i+1} = \{ v_{(i+1)1}, v_{(i+1)2}, \ldots, v_{(i+1)l(i)} \}, \tag{6}$$

where t_{ij} indicates an instance in T_i and $v_{(i+1)j}$ is a new variable referenced to Λ_{il}. Obviously, an attribute of

[2] The labelling scheme in this paper is a 3-star rating, scaling from 0 to 2 for non-relevant, possibly relevant and definitely relevant.

[3] The same normalization process is also conducted on the test dataset.

instance $t_{(i+1)j}$ of T_{i+1} is created by instance t_{ij} of T_i with a corresponding individual belonging to $\{\Lambda_{i1}, \Lambda_{i2}, \ldots, \Lambda_{il(i)}\}$ obtained in the previous layer L_i. It is said that an *era* ends while T_{i+1} is produced. The layer L_{i+1} then begins the evolution process with T_{i+1}.



T_1
Population 1
Population 2
Population $l(1)$
A set of populations evolving with T_1
A set of best individuals
Layer 1
Function value evaluation
T_2
Population 1
Population 2
Population $l(2)$
A set of populations evolving with T_2
A set of best individuals
Layer 2
T_Γ
Population 1
A population evolving with T_Γ
The best individual of population 1
Layer Γ

Fig. 2. The layered multi-population GP architecture of RankMGP.

In the implementation, T_1 and S_v^1 are initialized as T and the feature set, F, in Eq. (1), respectively. The final layer contains a population only. The reason for such a setting is obvious: RankMGP aims to produce a single value for each document. Having multiple outputs will require a voting mechanism to decide which one is the final judgment, hence making things more complicated. Another thing worth pointing out is that RankMGP passes query-document pairs and their relevance judgments from the initial layer L_1 to the next layer L_2, from the layer L_2 to L_3, and so forth. In this way, the effectiveness of one individual can be evaluated by comparing the predicted relevance judgments and the given judgments.

The proposed RankMGP method is summarized in Fig. 3.

```
Learning Procedure
Input 1: the feature set, $F$; the training dataset, $T$; the validation dataset, $V$;
Input 2: RankMGP-related parameters
Output: a ranking function, $f$, which is supposed to bind a query and a document with a real number,
        based on which a ranking of documents can be constructed

Step 1:
  Set $T_1 = T$ and $S_v^1 = F$.
Step 2:
  for $i = 1$ to $\Gamma$
```

```
        Set $\Lambda_i = \emptyset$.
        **for** $j$ = 1 to $l(i)$
            (1) Randomly initialize a set of individuals as the initial population for $P_j$.
            (2) $P_j$ evolves with $T_i$ for a number of generations and collects the best individuals in all
                generations as candidate solutions.
            (3) Use $T_i$ and $V$ to select the winning individual $\Lambda_{ij}$ from the candidates obtained in Step (2)
                and put it into $\Lambda_i$.
        **end for**
        Use $\Lambda_i$ to generate a new training set, $T_{i+1}$, and a new variable set, $S_v^{i+1}$, for the next layer, as
        defined in Eq. (5) and Eq. (6).
    **end for**
**Step 3:**
    Output the best and only individual $\Lambda_{\Gamma 1}$ in $\Lambda_\Gamma$ as $f$.


**Test Procedure**

**Input:** the test dataset, $T'$; the ranking function, $f$;
**Output:** evaluation reports

**Step 1:** Input $T'$ into $f$ and obtain a set of document rankings for queries.
**Step 2:** Generate evaluation reports in accordance with performance measures, such as MAP and
          MeanNDCG (as defined in Section 4.2).
```

Fig. 3. The learning and test procedures of RankMGP.

3.2.1. The design of a single population

In GP, a population is composed of a set of individuals; each is a possible solution for the given problem [27]. In this paper, an individual is viewed as a potential ranking function to bind a query and a document with a real number,[4] according to which a ranking of documents can be reasoned with respect to a given query.

An individual, I, is defined as a functional expression with three components: S_v (variables), S_c (constants), and S_{op} (operators); see Eq. (7):

$$I = (S_v, S_c, S_{op}),\ where \tag{7}$$

$$S_v = \{v_1, v_2, \ldots, v_{|v|} \mid |v| \text{ is the number of dimensions of the feature vector that represents a training instance}\},$$

$$S_{op} = \{+, -, \times, /, sin, cos, log\},$$

$$S_c = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, \pi, e\},$$

where S_v is a set of symbolic notations, referring to the dimensions of the vector representation of a training instance; S_c is a set of predefined real numbers, ranging from 0 to 1; and S_{op} is a set of arithmetic operators. Notably, the operator / in S_{op} is a protected division, meaning that when the denominator equals zero, the division gets 1. Though the computational costs increase, a logarithm operator and two trigonometric functions are included in S_{op} for generating complex non-linear functions. Two constants π and e are also added in S_c, particularly for operators $\{sin, cos, log\}$.

Consider that operators $\{+, -, \times, /\}$ and $\{sin, cos, log\}$ in S_{op} perform, binary and unary operations, respectively. A broadly adopted way to formulate I is to represent it as a binary tree structure, in which an internal node

[4] Ideally, the numerical value can be seen as the degree of relevance (or similarity) between a document and a query.

indicates a unary or binary operator while a leaf is a variable or a constant. The maximum number of available nodes of an individual is determined by the depth of the tree. For example, Fig. 4 the tree structure for the expression $cos(-f_1) / (f_2 + 0.5 \times f_3)$.

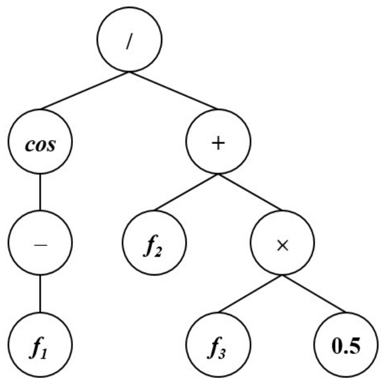


Fig. 4. The tree structure for the expression $cos(-f_1) / (f_2 + 0.5 \times f_3)$.

The fitness function is utilized to evaluate the fitness of an individual, i.e., how good an individual is for the given task. Since this study focuses on the ranking problem of IR, measures such as Precision at Position *n* (P@*n*) [3], MAP [3] and NDCG at Position *n* (NDCG@*n*) [25] are the candidates. This is due to the fact that an individual implies a ranking function that measures the relationship between a query and a document, based on which an ordering of documents can be established. Hence, IR evaluation measures can be calculated. For a training set T_i and an individual *I*, the fitness function *FFunc* is provided by:

$$FFunc(I, T_i) = \sum_{n=1}^{\lambda} \frac{(\lambda - n + 1)}{\lambda} \times \text{NDCG}@n\,, \tag{8}$$

where λ is a constant and NDCG@*n* is the average NDCG@*n* over all queries. NDCG takes into account the fact that the greater the ranked position of a relevant document, the less valuable it is since it is less likely to be examined by the user [25]. From this perspective, the practical design of *FFunc* is in essence the weighted average of NDCG@*n*, where NDCG@1 gets the highest weight, NDCG@2 gets the second highest weight, and so forth. Please refer to Section 4.2 for details about NDCG.

For a population, $P = \{I_1, I_2, \ldots, I_{|P|}\}$, the individual with the best fitness can be obtained by an evolution process of *P* that iterates through generations. The population in the initial generation is produced by the ramped half-and-half method [27]. Individuals are generated by a random process, but it is ensured that half of the individuals *must* have and the others *might not* have all branches of the maximum tree depth. In each generation, the evolution process simulates a natural selection mechanism by performing genetic operations, including *crossover*, *mutation* and *reproduction*, to produce a new population for the next generation. Theoretically, the choice of the operations depends on a predefined probability over these operations. In this paper, however, an *elitism* strategy is further taken into account. For the reproduction, a number of the best individuals in the current generation pass through to the next generation. This is to imitate the principal of natural selection, that is, survival of the fittest (i.e., the elite). In the implementation, only the fittest individual is passed to the next generation in order to retain the most diversity of the new population. The remaining individuals are produced via crossover and mutation, and the deterministic tournament selection [27] is exploited to bias the fitness. The method first randomly selects a few individuals from the previous generation and returns one individual (for mutation) or two individuals (for crossover) that have the best fitness values. For crossover, two new individuals are produced by swapping randomly chosen sub-trees of the selected individuals. For mutation, a mutant is created by randomly picking up an internal node of the selected individual (say, the mutation point) and replacing its whole sub-tree with a randomly generated tree. The mutation benefits the evolution process because a new structure, implying a new solution, is a great help to escape from the so-called "*local optimum*". Note that, in cases of crossover and mutation, only those offspring or mutants with better fitness values than the parents are included in the new population.

Traditional GP outputs the best individual in the final generation as the solution [27]. This study, instead, collects the best individuals in all generations as the candidate solutions. To select the best solution as the final output, Eq. (9) defines a score which leverages the fitness values on the training and validation sets:

$$score(I, T_i) = \alpha \cdot FFunc(I, T_i) + \beta \cdot FFunc(I, V_i), \quad (9)$$

where α and β are constants and $\alpha + \beta = 1.0$ holds; $FFunc(I, T_i)$ and $FFunc(I, V_i)$[5] represent the fitness values on the training set T_i and the validation set V_i, respectively. Typically, the validation set helps to avoid *over-fitting*, where the evolution process excessively learns to adapt to the training set. This study adopts a simple principle that a good individual should have good fitness values both on the training and validation sets [22], so as to examine the generalization of an individual.

3.2.2. The implementation

RankMGP has been implemented using the LAGEP toolkit.[6] The toolkit implements the layered multi-population GP, proposed in [29], for classification problems. The output can be a linear or a non-linear function, depending on the operators used to produce the functional expression of an individual. The function works as a binary classifier, which recognizes positive instances if and only if their function values are greater than or equal to a predefined threshold and otherwise repels negative instances. The default fitness function reports the accuracy of the classification.

The changes that this study makes to LAGEP are (1) letting LAGEP produce a real number as the relevance judgment for each query-document pair; (2) plugging in the new fitness function, as defined in Eq. (8); and (3) implementing the selection rule, as indicated in Eq. (9), for choosing the best individual from the candidate solutions. The layered structure and multi-population GP in LAGEP are employed in RankMGP. In addition, an adaptive mutation rate tuning method (AMRT), coming with LAGEP, is adopted since it has been shown effective in classification problems. AMRT dynamically changes the mutation rate during the learning process (see [29]). In each generation, AMRT increases the mutation rate for the next generation if all individuals in the current generation have similar fitness values; otherwise, the mutation rate is set as the initial one. AMRT ensures that the mutation rate achieves a maximum of 0.5 in the last generation.

4.0 EXPERIMENTS

4.1. The LETOR benchmark datasets

The LETOR 4.0 benchmark datasets[7] are used to conduct the experiments. The datasets are created as query-document pairs, each containing a 46-dimensional feature vector and its corresponding relevance judgment. The 5-fold partitions are provided for cross-validation. In each fold, three subsets are for learning, one subset for validation, and the other one for testing. RankMGP is tested on the MQ2008 dataset, following the settings for supervised ranking defined in LETOR. There are in total 15,211 instances. In each fold, an average of 9,127 instances in the training set, an average of 3,042 instances in the validation set and an average of 3,042 instances in the testing set. The features cover most of the standard IR features, such as low-level content features (e.g., term frequency, inverse document frequency and document length), high-level content features (e.g., Okapi BM25 [40] and LMIR [62] with different smoothing methods), and others, such as the in-link number of a webpage and the length of the URL. The relevance judgments are quantified on three levels, namely, 2 for *definitely relevant*, 1 for *possibly relevant* and 0 for *not relevant*. Table 2 provides an illustration of the sample data, in which each row indicates a query-document pair.

Table 2. Sample data excerpted from the MQ2008 dataset.

label	query_id	f_1	f_2	…	f_{46}	doc_id
2	10032	0.056537	0.000000	…	0.076923	GX029-35-5894638

[5] Since an individual is a function formed with the original variable set, no additional effort is required for the validation before calculating the fitness of it.

[6] http://people.cs.nctu.edu.tw/~jylin/lagep/lagep.html.

[7] http://research.microsoft.com/en-us/um/beijing/projects/letor/letor4dataset.aspx.

0	10032	0.279152	0.000000	…	1.000000	GX030-77-6315042
0	10032	0.130742	0.000000	…	1.000000	GX140-98-13566007
1	10032	0.593640	1.000000	…	0.000000	GX256-43-0740276

4.2. Evaluation measures

The standard P@*n*, NDCG@*n*, and MAP measures are exploited in the evaluation.

(1) Precision at Position *n* (P@*n*) [3]

For a given query, the precision of the top *n* results of the ranking list is defined as:

$$\text{P@}n = \frac{\text{\# of relevant documents in top } n \text{ results}}{n}. \tag{10}$$

Note that, when computing P@*n*, a document with the relevance judgment of either *definitely* or *possibly relevant* is regarded as a document relevant to the given query. The mean P@*n* is reported by averaging the P@*n* values of all the queries.

(2) Mean Average Precision (MAP) [3]

For a query, the average precision, *AP*, is computed as shown in Eq. (11), where *N* is the number of retrieved documents and *rel*(*n*) is either 1 or 0, indicating whether the *n*-th document is relevant to the query or not. The MAP is obtained as the mean average precision over a set of queries.

$$AP = \frac{\sum_{n=1}^{N} P@n \times rel(n)}{\text{\# of relevant documents for this query}}. \tag{11}$$

(3) Normalized Discounted Cumulative Gain (NDCG) [25]

For a query, the NDCG of its ranking list at position *n* is calculated by:

$$\text{NDCG@}n = Z_n \sum_{j=1}^{n} \begin{cases} 2^{r(j)} - 1, j = 1 \\ \dfrac{2^{r(j)} - 1}{\log(j)}, j > 1 \end{cases}, \tag{12}$$

where *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 receives an NDCG of 1. The *r*(*j*) is set to the relevance judgment, i.e., configured to 2 when the *j*-th document is definitely relevant to the query, configured to 1 when the *j*-th document is possibly relevant to the query, and finally configured to 0 when the *j*-th document is irrelevant to the query. The NDCG@*n* is computed as the average of the NDCG@*n* values of all the queries.

For comparison, this paper reports results for {NDCG@1, NDCG@2, …, NDCG@10}, {P@1, P@2, …, P@10}, MAP and MeanNDCG.[8]

4.3. Baseline ranking algorithms

Several baseline ranking algorithms, namely, RankSVM-Struct (an efficient version of Ranking SVM [23][26]), ListNet [8], AdaRank [56] (i.e., in two versions, AdaRank-NDCG and AdaRank-MAP) and RankBoost [21], are tested for comparison studies. All baselines choose linear ranking functions except RankBoost which adopts non-linear ranking functions. Query-level feature normalization (see Section 3.1) is also conducted. The same settings mentioned in the original papers are used, and all the algorithms utilize MAP on the validation set for

[8] The MeanNDCG is reported as the mean average NDCG over a set of queries, where for a query the average NDCG is computed by averaging NDCG@*n* for all *n*.

model selection.

In addition, the evaluation results of single features, including Okapi BM25 of the whole document (ID: 25), LMIR.ABS of the whole document (ID: 30), LMIR.DIR of the whole document (ID: 35) and LMIR.JM of the whole document (ID: 40), are provided.

4.4. Experimental settings

Table 3. Parameter settings of RankMGP.

Parameter	**Setting**
Function type	Non-linear function
Γ	3
$l(1)$, i.e., # of populations in L_1	10
$l(2)$, i.e., # of populations in L_2	10
$l(3)$, i.e., # of populations in L_3	1
Population size	600 for each population in L_1 and L_2; 1000 for the population in L_3
Generations	200 for all populations
Tree depth	10 (a total of 1023 nodes in a full tree)
Constants	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, π, e
Operators	+, –, ×, /, sin, cos, log
Tournament size	5 for L_1 and L_2; 7 for L_3
Crossover rate	0.9
Mutation rate	0.1 (Starts with 0.1 and is dynamically changed by ARMT [29] during the evolution to a maximum of 0.5 in the last generation)
Reproduction	Only the fittest individual in the generation gets through to the next generation unmodified
λ	10 (i.e., only NDCG@1, …, NDCG@10 are considered)
α	0.5
β	0.5

Table 3 lists all the parameter settings of RankMGP. Most of the parameters are set empirically based on preliminary experiments on a small randomly built held-out dataset. RankMGP targets the non-linear ranking function since operators {/, *sin*, *cos*, *log*} are employed (see Section 3.2.1). The number of generations, tournament size, crossover rate, mutation rate and reproduction are configured according to [29] and [58]. The tree depth is set to at least cover the case that leaf nodes contain 46 distinct variables and 46 distinct constants. In other words, a full binary tree with a depth of 10 has 512 leaf nodes, which is enough to include 92 nodes (i.e., 46 + 46 = 92).

In the experiments, RankMGP is performed five times. In each run, 5-fold cross-validation is conducted and an average score is obtained. The reported score of RankMGP is the average of those in the five runs. This is to reduce the effects in which the initialization of the first population by a random process might accordingly affect the final results [19].

4.5. Results

Three types of evaluation results of RankMGP are presented: (1) RankMGP (Avg), as mentioned in Section 4.4, is the average score of five runs of 5-fold cross-validation; (2) RankMGP (Max-MAP) is the average score of the best score, regarding MAP, of each fold in five runs; and (3) RankMGP (Max-NDCG) is the average score of the best score, regarding MeanNDCG, of each fold in five runs. Note that RankMGP (Max-MAP) and RankMGP (Max-NDCG) are both considered to be the upper bounds of performance.

Table 4 provides the performance of RankMGP in terms of MAP and MeanNDCG, where the 95% confidence intervals for RankMGP are given in brackets. Taking into consideration MAP, RankMGP (Avg) outperforms RankSVM-Struct with an improvement of 1.04% only. When MeanNDCG is considered, it outdoes RankSVM-Struct and RankBoost with improvements of 0.55% and 0.19%, respectively. Overall, AdaRank-NDCG has the best performance in both MAP and MeanNDCG. RankMGP (Max-MAP) has the second best performance in MAP, with a slight difference of -0.19% as compared to AdaRank-NDCG. Meanwhile, RankMGP (Max-

NDCG) has the second best performance in MeanNDCG, with a difference of -0.63% as compared to AdaRank-NDCG.

Table 4. Ranking performance of MAP and MeanNDCG.

Algorithm	MAP	MeanNDCG
BM25	0.3588	0.3595
LMIR.ABS	0.3497	0.3484
LMIR.DIR	0.3137	0.3082
LMIR.JM	0.4469	0.4529
RankSVM-Struct	0.4696	0.4832
ListNet	0.4775	0.4914
AdaRank-NDCG	0.4824	0.4950
AdaRank-MAP	0.4764	0.4915
RankBoost	0.4775	0.4850
RankMGP (Avg)	0.4745 [± 0.0028]	0.4859 [± 0.0024]
RankMGP (Max-MAP)	**0.4815** [± 0.0444]	0.4893 [± 0.0460]
RankMGP (Max-MeanNDCG)	0.4771 [± 0.0444]	**0.4919** [± 0.0455]

Table 5 and Table 6 present the relative performance of MAP and MeanNDCG, respectively, of RankMGP and other ranking methods. A widely used test in machine learning, namely the *k*-fold cross-validated paired *t*-test [16], is exploited for comparing different algorithms. Given a ranking method in a column, a (+) sign is included when RankMGP is statistically better than the method. On the other hand, a (−) sign is included when RankMGP is statistically inferior to the method and a (=) sign is included when RankMGP is statistically equal to the method. The values *x*/*y*/*z* stand for the number of wins, ties and losses. For instance, RankMGP (Avg) is statistically equal to RankSVM-Struct in Table 5, although the relative improvement of RankMGP (Avg) over RankSVM-Struct is 1.03%. 3/1/1 implies that RankMGP (Avg) outperforms RankSVM-Struct three times, RankMGP (Avg) and RankSVM-Struct tie one time while RankMGP (Avg) loses to RankSVM-Struct one time.

In Table 5, RankMGP is found to be significantly superior to all single features. It is also observed that the performance of MAP of RankMGP is statistically equal to the performance of other baselines, except that RankMGP (Avg) is statistically inferior to AdaRank-NDCG with a difference of -1.65% and RankMGP (Max-MAP) statistically outperforms RankSVM-Struct with an improvement of 2.54%. In Table 6, it is again shown that no single feature is statistically better than RankMGP. In most comparisons, a similar behavior can be seen, where the performance of MeanNDCG of RankMGP is statistically equal to the performance of other baselines. However, RankMGP (Avg) is statistically inferior to ListNet and AdaRank-NDCG with differences of -1.12% and -1.84%, respectively; the performance of RankMGP (Max-MAP) is statistically worse than that of AdaRank-NDCG with a difference of -1.16%. Overall, Table 5 and Table 6 illustrate that the proposed RankMGP performs well with statistically competitive results, as compared to other baselines.

Table 5. Relative performance of MAP among different ranking algorithms.

	RankMGP (Avg)	RankMGP (Max-MAP)	RankMGP (Max-MeanNDCG)
BM25	32.24% (+, 5/0/0)	34.21% (+, 5/0/0)	32.98% (+, 5/0/0)
LMIR.ABS	35.68% (+, 5/0/0)	37.70% (+, 5/0/0)	36.44% (+, 5/0/0)
LMIR.DIR	51.25% (+, 5/0/0)	53.5% (+, 5/0/0)	52.1% (+, 5/0/0)
LMIR.JM	6.17% (+, 5/0/0)	7.75% (+, 5/0/0)	6.77% (+, 5/0/0)
RankSVM-Struct	1.03% (=, 3/1/1)	2.54% (+, 4/0/1)	1.61% (=, 3/0/2)
ListNet	-0.64% (=, 0/0/5)	0.85% (=, 3/1/1)	-0.08% (=, 1/0/4)
AdaRank-NDCG	-1.65% (−, 0/0/5)	-0.18% (=, 1/0/4)	-1.09% (=, 0/0/5)
AdaRank-MAP	-0.41% (=, 2/0/3)	1.08% (=, 4/0/1)	0.16% (=, 2/0/3)
RankBoost	-0.64% (=, 2/0/3)	0.85% (=, 3/0/2)	-0.08% (=, 3/0/2)

Table 6. Relative Performance of NDCG among different ranking algorithms.

	RankMGP (Avg)	RankMGP (Max-MAP)	RankMGP (Max-MeanNDCG)
BM25	35.15% (+, 5/0/0)	36.10% (+, 5/0/0)	36.82% (+, 5/0/0)
LMIR.ABS	39.46% (+, 5/0/0)	40.44% (+, 5/0/0)	41.18% (+, 5/0/0)
LMIR.DIR	57.65% (+, 5/0/0)	58.75% (+, 5/0/0)	59.59% (+, 5/0/0)
LMIR.JM	7.28% (+, 5/0/0)	8.03% (+, 5/0/0)	8.6% (+, 5/0/0)
RankSVM-Struct	0.55% (=, 3/0/2)	1.26% (=, 4/0/1)	1.79% (=, 4/0/1)
ListNet	-1.12% (–, 0/0/5)	-0.43% (=, 2/0/3)	0.09% (=, 2/0/3)
AdaRank-NDCG	-1.84% (–, 0/0/5)	-1.16% (–, 0/0/5)	-0.63% (=, 1/0/4)
AdaRank-MAP	-1.14% (=, 2/0/3)	-0.45% (=, 2/0/3)	0.07% (=, 2/0/3)
RankBoost	0.18% (=, 2/0/3)	0.88% (=, 2/0/3)	1.41% (=, 4/0/1)

5.0 DISCUSSIONS

5.1. From classification to ranking

In Section 3.1, each query-document pair is labelled with a 3-star rating of relevance, namely, non-relevant, possibly relevant and definitely relevant. To address the ranking problem, the simplest way is to treat it as a classification problem. In other words, to learn a multi-class classification model such that each document is classified to one rating, and then to rank documents according to their class labels. This can be done as perfect classifications leads to perfect IR performance. However, there are two major shortcomings of classification-oriented methods [28]: (1) the classification result is not always perfect; and (2) documents with the same labels are arbitrarily ranked, which might lead to highly unstable ranking results.

Instead of following the classification paradigm, this paper assumes that there is a function that gives a score for each document while a particular query is considered. In accordance with the scores, all documents are sorted in order and IR evaluation measures can be calculated based on the known rating of relevance for each query-document pair. So far, the only issue is to find such a function. Fortunately, GP naturally provides an evolutionary computation scheme that is able to efficiently find a solution in a functional expression by directly optimizing IR evaluation measures, which can be transformed as the fitness function to score the ability of a possible solution fitting with the addressed problem.

5.2. Why GP is naturally beneficial for LTR for IR

The reasons GP is employed in this work include (1) the use of functional expressions to represent individuals[9] and (2) the ability to find out a near or exact global optimal solution [27]. In comparison with other methods (e.g., Ranking SVM [23][26] and RankBoost [21]), which decouple the problem into individual pairwise evaluations, the main advantage of RankMGP is that non-convex objective functions of IR evaluation measures (e.g., the weighted average of NDCG scores of this paper) can be directly optimized.

In the literature, many studies have aimed to find a linear ranking function that can be optimized using greedy search algorithms (e.g., conjugate gradient descent). In contrast, RankMGP is designed to generate a wider class of non-linear ranking functions by using more complex operators in S_{op}. This makes the benefits of RankMGP much clearer, since a non-linear function cannot be directly optimized with greedy algorithms. Note that RankMGP can also learn a linear ranking function as long as only simple operators are engaged.

However, the computational cost of RankMGP is higher and more disproportionate to other methods. Running the current implementation of RankMGP with the parameter settings listed in Table 3, it takes approximately 8 hours to conduct a run of 5-fold cross-validation on MQ2008 with a PC equipped with one 3 GHz processor and 8 GB memory. It is the user's choice to determine whether RankMGP or other methods are suitable for the cases he/she is facing. In general, it is suggested to apply RankMGP when the main goal is to find a potentially optimal solution, provided that computing resources and time are not limited.

[9] It is natural to represent a ranking function for IR in a function expression.

5.3. The layered multi-population GP in RankMGP

The advantages of RankMGP are discussed below. First, a layer has a great probability of having a higher fitness value than its previous layer (see Section 5.4). Secondly, the employment of multi-population GP in RankMGP provides greater diversity among populations, better search space coverage and faster convergence against traditional GP [20]. Thirdly, recall that RankMGP replaces features by evolutionary feature generation to construct new training instances (i.e., a training instance $t_{(i+1)j} \in T_{i+1}$ is derived by transforming $t_{ij} \in T_i$, with the set of the best individuals obtained in L_i, to a new space). Each feature value of an instance provides ranking information inherited from its corresponding instance in the previous layer. In other words, the layer architecture is capable of passing ranking information through layers. Furthermore, the number of new features decreases gradually since the number of populations over layers is usually set degressively. Therefore, a successive layer evolves with shorter individuals, which speeds up the entire evolution process. This is especially beneficial for handling high-dimensional training instances. Finally, the layered architecture allows each layer to have its own configuration. The scope of the configuration includes the number of populations, the number of generations and the depth limitation of individuals. One can benefit from the flexibility of configuration for an attempt to find better solutions with different specified configurations or for the accelerated learning speed.

5.4. The learning behaviors of RankMGP over layers

Fig. 5 graphs the average learning scores of distinct folds in different layers. The average score is computed by averaging scores of populations. An upward trend of scores is observed during the learning processes of each layer, indicating that RankMGP is able to discover a better ranking function (i.e., individual) through the layers for the training dataset. RankMGP also demonstrates different learning capacities for different folds; e.g., relatively high scores for Fold 2 and Fold 3, but a relatively low score for Fold 5. This phenomenon might occur when the data partition of folds varies greatly. In such a situation, it is difficult for RankMGP to find a suitable function. In Fig. 6, the average learning scores of five runs for 5-fold cross-validation are provided. It is shown that RankMGP gradually obtains better scores through layers, which testifies that the layered multi-population architecture of RankMGP has a better learning capacity than traditional single-population GP.

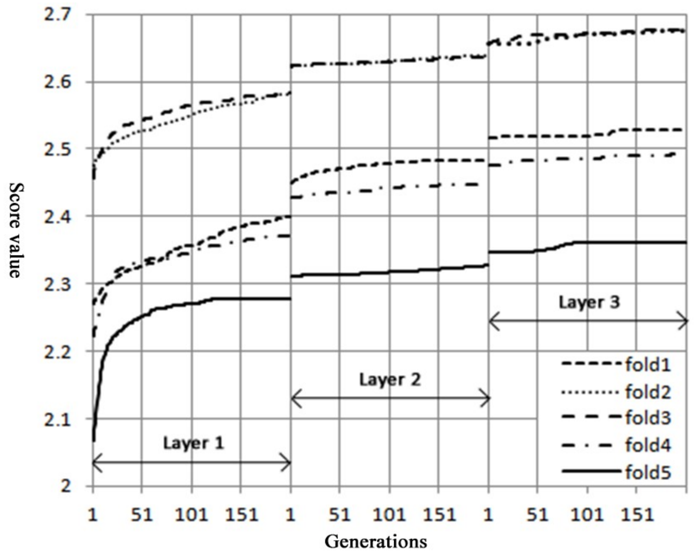


Fig. 5. The average learning values of *FFunc* of folds in different layers.

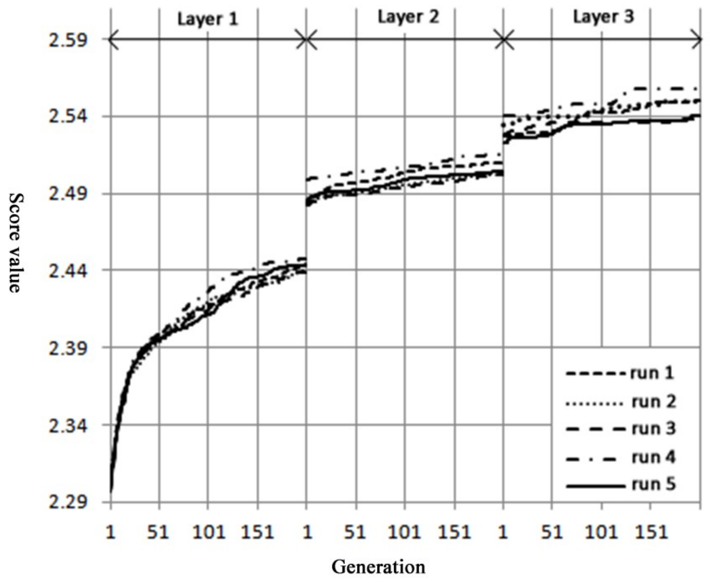


Fig. 6. The average learning values of *FFunc* of five runs for 5-fold cross-validation in different layers.

The average learning scores of distinct folds in different layers are shown in Fig. 7. Similar to Fig. 5, an upward trend of scores is observed during the learning processes of each layer, except for Fold 1. Fig. 8 provides further information about RankMGP on Fold 1. It can be seen that the fitness on the training data increases smoothly through each layer. However, the fitness on the validation data fluctuates. The fitness on the validation is approximately 2.3 in Layer 1, then drops off to about 2.1 in Layer 2, and finally increases to approximately 2.19 in Layer 3. It is conjectured that the properties of the validation data in Fold 1 might be quite different from those of the training data. Note that the validation set is used to prevent over-fitting since an individual with a good fitness on the training dataset but with a bad fitness on the validation set is not considered a good individual. In such a case, the selection of the best individual based on its score might be affected and hence fails to find a good individual for Fold 1.

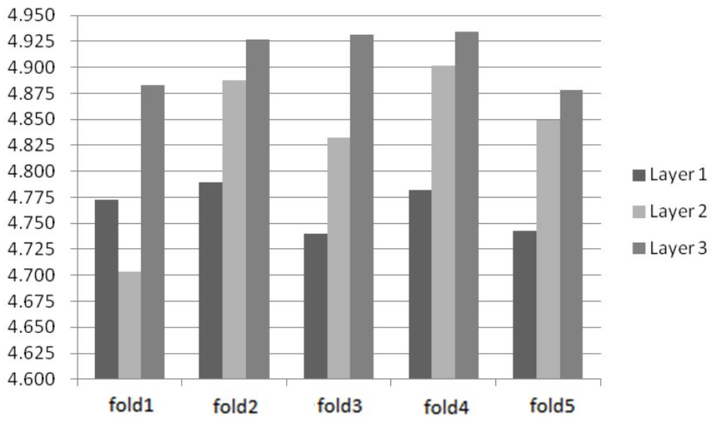


Fig. 7. The average learning scores of folds in different layers.

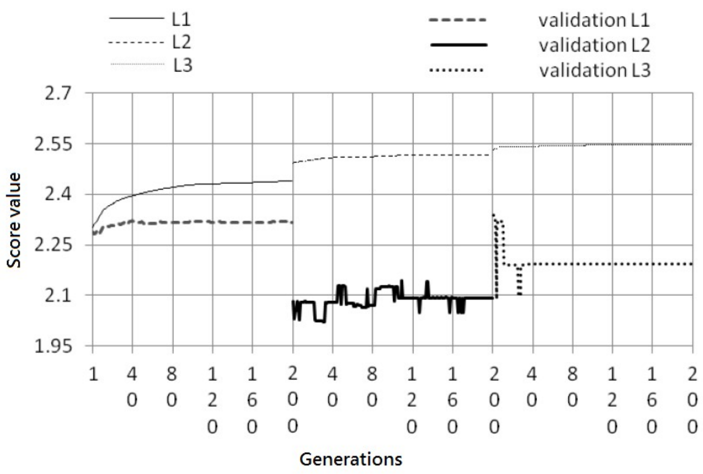


Fig. 8. The average fitness values in different layers, obtained from the training and validation datasets.

6.0 CONCLUSION AND FUTURE WORK

This paper proposed a novel layered multi-population GP based method, called RankMGP, for LTR for IR. The main idea behind RankMGP is to generate a function that gives a score to each document while a particular query is considered. By arranging documents in order according to the scores, IR evaluation measures can be utilized as fitness functions to guide RankMGP for finding an effective ranking function. RankMGP consists of a set of layers and a sequential workflow running through the layers. In one layer, multiple populations evolve independently to generate a set of the best individuals. When the evolution process is completed, a new training dataset is created using the best individuals and the input training set of the layer. Then, populations in the next layer evolve with the new training dataset. In the final layer, the best individual is obtained as the output ranking function. Experiments are conducted to evaluate the effectiveness of RankMGP using the MQ2008 dataset of the LETOR benchmark package. Several baseline ranking algorithms, namely, Ranking SVM, ListNet, AdaRank and RankBoost, are also tested to prove the effectiveness of RankMGP. The results show that RankMGP performs well with statistically competitive results.

As for future work, we will continue to investigate the relationships between features for developing feature selection methods to identify significant features for the ranking problem. It is also important to compare the use of linear and non-linear functions in RankMGP and study the effects of different parameter settings of the algorithm. Another interesting issue is to explore whether more complex operators will help in finding a better ranking function. Moreover, it is intended to study the effectiveness of RankMGP with different population configurations (e.g., the heterogeneous multi-population GP model). Lastly, the scalability issues need to be taken into account to speed up the training process of RankMGP, e.g., to evolve populations in a distributed manner by the parallel and distributed GP model [36].

REFERENCES

[1] O. Alejo, J. M. Fernandez-Luna, J. F. Huete, & R. Perez-Vazquez, “Direct Optimization of Evaluation Measures in Learning to Rank Using Particle Swarm”, in Proceedings of the 21st International Workshop on Database and Expert Systems Applications, Bilbao, Spain, 2010, pp. 42-46.

[2] H. M. Almeida, M. A. Goncalves, M. Cristo, & P. Calado, “A Combined Component Approach for Finding Collection-Adapted Ranking Functions Based on Genetic Programming”, in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007), Amsterdam, Netherlands, 2007, pp. 399-406.

[3] R. Baeza-Yates, & B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search. Addison-Wesley Professional, 2011.

[4] D. Bollegala, N. Noman, & H. Iba, “RankDE: Learning a Ranking Function for Information Retrieval

Using Different Evolution", in Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO 2011), Dublin, Ireland, 2011, pp. 1771-1778.

[5] C. J. C. Burges, "From RankNet to LambdaRank to LambdaMART: An Overview". Microsoft Research Technical Report (MSR-TR-2010-82), 2010.

[6] C. J. C. Burges, R. Ragno, & Q. V. Le, "Learning to Rank with Nonsmooth Cost Functions", in Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS 2006), Vancouver, BC, Canada, 2006, pp. 193-200.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, & G. Hullender, "Learning to Rank Using Gradient Descent" in Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), Bonn, Germany, 2005, pp. 89-96.

[8] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, & H. Li, "Learning to Rank: From Pairwise Approach to Listwise Approach", in Proceedings of the 24th International Conference on Machine Learning (ICML 2007), Corvallis, OR, 2007, pp. 129-136.

[9] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, & H.-W. Hon, "Adapting Ranking SVM to Document Retrieval", in Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006), Seattle, WA, 2006, pp. 186-193.

[10] O. Chapelle, Y. Chang, & T.-Y. Liu, "Future Directions in Learning to Rank", in Proceedings of the Yahoo! Learning to Rank Challenge, Haifa, Israel, 2011, pp. 91-100.

[11] Y. Chen, & K. Hofmann, "Online Learning to Rank: Absolute vs. Relative", in Proceedings of the 24th International Conference on World Wide Web (WWW 2015), Florence, Italy, 2015, pp. 19-20.

[12] D. Cossock, & T. Zhang, "Subset Ranking Using Regression", in Proceedings of the 19th Annual Conference on Learning Theory (COLT 2006), Pittsburgh, PA, 2006, pp. 605-619.

[13] K. Crammer, & Y. Singer, "Pranking with Ranking", in Proceedings of the 15th Annual Conference on Neural Information Processing Systems (NIPS 2001), Vancouver, BC, Canada, 2001, pp. 641-647.

[14] R. Cummins, & C. O'Riordan, "Term-Weighting in Information Retrieval Using Genetic Programming: A Three Stage Process", in Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), Riva del Garda, Italy, 2006, pp. 793-794.

[15] E. Diaz-Aviles, W. Nejdl, & L. Schmidt-Thieme, "Swarming to Rank for Information Retrieval", in Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009), Montreal, QC, Canada, 2009, pp. 9-16.

[16] T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms". Neural Computation, Vol. 10, No. 7, 1998, pp. 1895-1923.

[17] W. Fan, M. Gordon, & P. Pathak, "Automatic Generation of Matching Function by Genetic Programming for Effective Information Retrieval", in Proceedings of the 5th Americas Conference on Information Systems (AMCIS 1999), Milwaukee, WI, 1999, pp. 49-51.

[18] W. Fan, M. Gordon, & P. Pathak, "A Generic Ranking Function Discovery Framework by Genetic Programming for Information Retrieval". Information Processing and Management, Vol. 40, No. 4, 2004, pp. 587-602.

[19] W. Fan, M. Gordon, & P. Pathak, "Discovery of Context-Specific Ranking Functions for Effective Information Retrieval Using Genetic Programming". IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 4, 2004, pp. 523-527.

[20] F. Fernandez, M. Tomassini, & L. Vanneschi, "An Empirical Study of Multipopulation Genetic Programming". Genetic Programming and Evolvable Machines, Vol. 4, No. 1, 2003, pp. 21-51.

[21] Y. Freund, R. Iyer, R. E. Schapire, & Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences". Journal of Machine Learning Research, Vol. 4, No. 6, 2003, pp. 933-969.

[22] C. Gagne, M. Schoenauer, M. Parizeau, & M. Tomassini, "Genetic Programming, Validation Sets, and Parsimony Pressure", in Proceedings of the 9th European Conference on Genetic Programming (EuroGP 2006), Budapest, Hungary, 2006, pp. 109-120.

[23] R. Herbrich, T. Graepel, & K. Obermayer, "Large Margin Rank Boundaries for Ordinal Regression", in A. J. Smola, P. Bartlett, B. Scholkopf, & D. Schuurmans (Eds.), Advances in Large Margin Classifiers, pp. 115-132. The MIT Press, 2000.

[24] M. A. Islam, "RankGPES: Learning to Rank for Information Retrieval Using a Hybrid Genetic Programming with Evolutionary Strategies". Master Thesis in Computer Science, Ryerson University, Toronto, ON, Canada 2013.

[25] K. Jarvelin, & J. Kekalainen, "Cumulated Gain-based Evaluation of IR Techniques". ACM Transactions on Information Systems, Vol. 20, No. 4, 2002, pp. 422-446.

[26] T. Joachims, "Optimizing Search Engines Using Clickthrough Data", in Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002), Edmonton, AB, Canada, 2002, pp. 133-142.

[27] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Nature Selection. The MIT Press, 1992.

[28] P. Li, C. J. C. Burges, & Q. Wu, "McRank: Learning to Rank Using Multiple Classification and Gradient Boosting", in Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS 2007), Vancouver, BC, Canada, 2007, pp. 897-904.

[29] J.-Y. Lin, H.-R. Ke, B.-C. Chien, & W.-P. Yang, "Designing a Classifier by a Layered Multi-Population Genetic Programming Approach". Pattern Recognition, Vol. 40, No. 8, 2007, pp. 2211-2225.

[30] T.-Y. Liu, Learning to Rank for Information Retrieval. Springer, 2011.

[31] I. Matveeva, C. Burges, T. Burkard, A. Laucius, & L. Wong, "High Accuracy Retrieval with Multiple Nested Ranker", in Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006), Seattle, WA, 2006, pp. 437-444.

[32] R. Nallapati, "Discriminative Models for Information Retrieval", in Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2004), Sheffield, UK, 2004, pp. 64-71.

[33] L. Nie, B. D. Davison, & X. Qi, "Topical Link Analysis for Web Search", in Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006), Seattle, WA, 2006, pp. 91-98.

[34] N. Oren, "Reexamining *tf.idf* Based Information Retrieval with Genetic Programming", in Proceedings of the 2002 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT 2002), Stellenbosch, South Africa, 2002, pp. 224-234.

[35] Z. Pan, X. You, H. Chen, D. Tao, & B. Pang, "Generalization Performance of Magnitude-Preserving Semi-Supervised Ranking with Graph-Based Regularization". Information Sciences, Vol. 221, 2013, pp. 284-296.

[36] R. Poli, "Parallel Distributed Genetic Programming", in D. Corne, M. Dorigo, & F. Glover (Eds.), New Ideas in Optimisation, pp. 403-432. McGraw-Hill Inc., 1999.

[37] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, & W.-Y. Ma, "A Study of Relevance Propagation for Web Search", in Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005), Salvador, Brazil, 2005, pp. 408-415.

[38] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, & H. Li, "Query-Level Loss Functions for Information Retrieval". Information Processing & Management, Vol. 44, No. 2, 2008, pp. 838-855.

[39] V. C. Raykar, R. Duraiswami, & B. Krishnapuram, "A Fast Algorithm for Learning a Ranking Function from Large-Scale Data Sets". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 30, No. 7, 2008, pp. 1158-1170.

[40] S. E. Robertson, "Overview of the Okapi Projects". Journal of Documentation, Vol. 53, No. 1, 1997, pp. 3-7.

[41] R. L. T. Santos, C. Macdonald, & I. Ounis, "On the Suitability of Diversity Metrics for Learning-to-Rank for Diversity", in Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011), Beijing, China, 2011, pp. 1185-1186.

[42] A. Qazi, H. Fayaz, A. Wadi, R. G. Raj, N.A. Rahim, W. A. Khan, "The artificial neural network for solar radiation prediction and designing solar systems: a systematic literature review", Journal of Cleaner Production, vol. 104, pp. 1-12, 2015. ISSN 0959-6526, http://dx.doi.org/10.1016/j.jclepro.2015.04.041.(http://www.sciencedirect.com/science/article/pii/S0959652615004096).

[43] A. Shashua, & A. Levin, "Ranking with Large Margin Principles: Two Approaches", in Proceedings of the 16th Annual Conference on Neural Information Processing Systems (NIPS 2002), Vancouver, BC, Canada, 2002, pp. 961-968.

[44] A. Slivkins, F. Radlinski, & S. Gollapudi, "Ranked Bandits in Metric Spaces: Learning Diverse Rankings over Large Document Collections". Journal of Machine Learning Research, Vol. 14, 2013, pp. 399-436.

[45] N. Tax, "Scaling Learning to Rank to Big Data: Using MapReduce to Parallelise Learning to Rank". Master Thesis in Computer Science, University of Twente, Enschede, Overijssel, Netherlands, 2014.

[46] M. Taylor, J. Guiver, S. Robertson, & T. Minka, "SoftRank: Optimising Non-Smooth Rank Metrics", in Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM 2008), Stanford, CA, 2008, pp. 77-86.

[47] A. Qazi, K. B. S. Syed, R. G. Raj, E. Cambria, M. Tahir, D. Alghazzawi, "A concept-level approach to the analysis of online review helpfulness", *Computers in Human Behavior*, Vol. 58, May 2016, PP. 75-81, ISSN 0747-5632, http://dx.doi.org/10.1016/j.chb.2015.12.028.

[48] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, & W.-Y. Ma, "FRank: A Ranking Method with Fidelity Loss", in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007), Amsterdam, Netherlands, 2007, pp. 383-390.

[49] A. Qazi, R. G. Raj, M. Tahir, M. Waheed, S. U. R. Khan, and A. Abraham, "A Preliminary Investigation of User Perception and Behavioral Intention for Different Review Types: Customers and Designers Perspective," The Scientific World Journal, vol. 2014, Article ID 872929, 8 pages, 2014. doi:10.1155/2014/872929.

[50] L. Wang, J. Lin, D. Metzler, & J. Han, "Learning to Efficiently Rank on Big Data", in Proceedings of the 23rd International Conference on World Wide Web (WWW 2014), Seoul, Korea, 2014, pp. 209-210.

[51] S. Wang, J. Ma, & Q. He, “An Immune Programming-Based Ranking Function Discovery Approach for Effective Information Retrieval”. Expert Systems with Applications, Vol. 37, No. 8, 2010, 5863-5871.

[52] S. Wang, Y. Wu, B. J. Gao, K. Wang, H. W. Lauw, & J. Ma, “A Cooperative Coevolution Framework for Parallel Learning to Rank”. IEEE Transactions on Knowledge and Data Engineering, Vol. 27, No. 12, 2015, 3152-3165.

[53] F. Wang, & X. Xu, “AdaGP-Rank: Applying Boosting Technique to Genetic Programming for Learning to Rank”, in Proceedings of the 2010 IEEE Youth Conference on Information Computing and Telecommunications (YC-ICT 2010), Beijing, China, 2010, pp. 259-262.

[54] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, & H. Li, “Listwise Approach to Learning to Rank: Theory and Algorithm”, in Proceedings of the 25th International Conference on Machine Learning (ICML 2008), Helsinki, Finland, 2008, pp. 1192-1199.

[55] J. Xu, Y. Cao, H. Li, & M. Zhao, “Ranking Definitions with Supervised Learning Methods”, in Proceedings of the 14th International Conference World Wide Web (WWW 2005), Chiba, Japan, 2005, pp. 811-819.

[56] J. Xu, & H. Li, “AdaRank: A Boosting Algorithm for Information Retrieval”, in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007), Amsterdam, Netherlands, 2007, pp. 391-398.

[57] J. Xu, T.-Y. Liu, M. Lu, H. Li, & W.-Y. Ma, “Directly Optimizing Evaluation Measures in Learning to Rank”, in Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2008), Singapore, Singapore, 2008, pp. 107-114.

[58] J.-Y. Yeh, J.-Y. Lin, H.-R. Ke, & W.-P. Yang, “Learning to Rank for Information Retrieval Using Genetic Programming”, in Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007), Amsterdam, Netherlands, 2007, pp. 41-48.

[59] H. Yu, “SVM Selective Sampling for Ranking with Application to Data Retrieval”, in Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005), Chicago, IL, 2005, pp. 354-363.

[60] M. Moohebat, R.G. Raj, S.B.A.Kareem, D. Thorleuchter, “Identifying ISI-indexed articles by their lexical usage: A text analysis approach”, Journal of the Association for Information Science and Technology, Vol. 66, No. 3, pp. 501–511. doi: 10.1002/asi.23194.

[61] Y. Yue, T. Finley, F. Radlinski, & T. Joachims, “A Support Vector Method for Optimizing Average Precision”, in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007), Amsterdam, Netherlands, 2007, pp. 217-278.

[62] C. Zhai, & J. Lafferty, “A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval”, in Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001), New Orleans, LA, 2001, pp. 334-342.

[63] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, & G. Sun, “A General Boosting Method and Its Application to Learning Ranking Functions for Web Search”, in Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS 2007), Vancouver, BC, Canada, 2007, pp. 1697-1704.