Appl Intell (2014) 40:525–535
DOI 10.1007/s10489-013-0475-z

Combining pre-retrieval query quality predictors using genetic programming

Shariq Bashir

Published online: 21 September 2013

Abstract Predicting the effectiveness of queries plays an important role in information retrieval. In recent years, a number of methods are proposed for this task, however, there has been little work done on combining multiple predictors. Previous studies on combining multiple predictors rely on non-backtracking based machine learning methods. These studies show minor improvement over single predictors due to the limitation of non-backtracking. This paper discusses work on using machine learning to automatically generate an effective predictors' combination for query performance prediction. This task is referred to as—learning to predict for query performance prediction in the field. In this paper, a learning method, PredGP, is presented to address this task. PredGP employs genetic programming to learn a predictor by combining various pre-retrieval predictors. The proposed method is evaluated using the TREC Chemical Prior-Art Retrieval Task dataset and found to be significantly better than single predictors.

Keywords Intelligent information retrieval · Query performance prediction · Pre-retrieval predictors · Learning to rank · Genetic programming

1 Introduction

In a typical information retrieval process users issue a query and retrieval systems return the answer set in the form of ranked list. Users then traverse the ranked list and examine whether the returned answer set satisfy their search task. If the returned answer set does not fulfill their search task, users then rephrase or modify the query in order to improve the effectiveness of search. This is a manual way of predicting the effectiveness of a query and requires a lot of time in order to traverse a long returned answer set. Additionally, this increases the burden on users. Query performance prediction (QPP) is a technique in information retrieval (IR) for automatically predicting the effectiveness of queries [7, 37, 43]. QPP is useful for both users and system point of view. For users, it can provide feedback about the effectiveness of query, and users then can rephrase the query in order to improve its effectiveness. Whereas, retrieval systems can utilize QPP mechanism for invoking alternative retrieval strategies such as query expansion, diversity retrieval, or reduction of terms in long queries for increasing the effectiveness of low quality queries [16].

In recent years, several techniques are proposed for predicting the quality of queries [7, 17, 19, 32, 37, 41, 43]. Despite the numerous methods proposed, little research has been performed on combining QPP predictors. Previous approaches on combining multiple predictors rely on classification (decision trees) techniques [39] or regression models [23]. The prediction accuracy of these approaches is not significant. This is because, these approaches are non-backtracking and easily trapped into local maximum as finding the optimal combination of predictors for a specific training-set is NP-complete. In order to find the optimal function of predictors' combination, a learning method, *PredGP*, based on genetic programming is developed by using a set of pre-retrieval predictors [7, 17, 41]. Genetic programming is efficient for searching large spaces because it probes, in parallel, many points in the search space. The advantage of this evolutionary approach is that it can help in solving problems that are extremely complex where the traditional algorithm is computationally infeasible. *PredGP*

S. Bashir (✉)
Center for Science and Engineering, New York University Abu Dhabi, Musaffah, Abu Dhabi, United Arab Emirates
e-mail: shariq.bashir@nyu.edu

represents a potential solution (i.e. query performance predictor) as an individual in a population of genetic programming. 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 on the basis of correlation between query performance predictors and effectiveness measure is exploited to evaluate the performance (accuracy) of each individual in the population. The evolution is supposed to eventually generate an individual with the best fitness as the optimal solution. We check the effectiveness of *PredGP* on TREC Chemical Prior-Art Retrieval Task dataset [25] and compared it with existing well-known pre-retrieval predictors. We found a significant increase in the accuracy of QPP with *PredGP*. This indicates the usefulness of this approach.

The remainder of this paper is structured as follows. Section 2 reviews related work on QPP. Section 3 provides an introduction about TREC Chemical Prior-Art Retrieval Task dataset, query sets and retrieval models that we use for analyzing the performance of query performance predictors. Section 4 lists several pre-retrieval predictors that we use for combining predictors. Section 5 presents performance analysis of pre-retrieval on TREC dataset. In Sect. 6 we combine query performance predictor using genetic programming for automatically evolving effective combination of predictors. Finally, Sect. 7 briefly summarizes key lessons learned from this study.

2 Related work

In recent years a considerable number of methods have been proposed that attempt to provide an indication of a query quality. These can be divided into three broad categories: pre-retrieval predictors, post-retrieval predictors, and learning predictors.

Pre-retrieval predictors are search-independent predictors in this sense that they rely on information that is available at indexing time and are, thus, independent of the ranked list of results [2, 7, 17, 19, 27, 28, 30, 41]. This causes less overhead to the search system. The prediction methodology in pre-retrieval predictors is solely based on the collection statistics of query terms. The collection statistics can be computed either directly from the document collection or external source such as WordNet. Pre-retrieval predictors can be divided into four different groups according to the heuristic they exploit for making their prediction: (a) specificity based predictors [7, 17, 41], (b) ambiguity based predictors [7, 17, 41], (c) term relatedness based predictors [19, 27], and (d) ranking sensitivity based predictors [2, 28, 30].

Post-retrieval predictors calculate queries quality scores from ranked lists of queries results. Post-retrieval predictors are computationally more expensive than pre-retrieval predictors, in this sense, they need to process queries at least once before a prediction can be made [43]. However using this additional computation is fruitful as it provides additional information for a query and this is useful in order to increase the accuracy of prediction. Post-retrieval predictors can be categorized into the following four classes. The algorithms in the first class change the query and consider the differences in the respective ranked lists of results [37, 39, 43]. The algorithms in the second class perturb the documents of the initially retrieved result list and consider the stability of the ranking [9, 34, 42]. The algorithms in the third class perturb the retrieval approach and consider the diversity of the ranked list of results [1, 3, 35]. The algorithms in the fourth class utilize web resources in order to drive the estimate of prediction [5, 21, 24].

Learning predictors incorporate a variety of statistical regression [23] or classification methods [39], such as decision tree or random forest, to train on labeled examples of easy and difficult queries. The learned estimator is then used to predict the difficulty of previously unseen queries. These learning predictors have shown somewhat better correlation with system effectiveness than single predictors, however, these approaches can easily trap into local maximum as the machine learning techniques used in these approaches are non-backtracking.

3 Dataset

The document collection used to test the efficiency of the proposed method is the TREC Chemical Prior-Art (PA) Retrieval Task collection [25]. For all the documents and queries, the stop-words are removed using a standard list and the Porter's stemming algorithm is applied. The PA task consisted of 1,000 topic queries that are the full-text patent documents (i.e., consisting of at least claims and abstract or description) taken from both the European Patent Office (EPO) and the US Patent Office (USPTO). The goal of searching a patent database for the prior-art search task is to find all previously published related patents on a given topic [14, 25, 26]. It is a common task for *patent examiners* and *attorneys* to decide whether a new patent application is novel or contains technical conflicts with some already patented invention. They collect all related patents and report them in a search report. We use these reports as relevance judgments. Next, we apply a standard approach for query generation in the patent retrieval domain. From each topic, we select only the claim section because it is regarded as being the most representative piece of text, characterizing the scope of invention well due to the rules of the patent system worldwide as done also in [14, 20, 26, 33]. In order to build the queries from the claim sections, we first sort all the terms in the

claim sections on the basis of their increasing term frequencies. Next, we generate queries for performance analysis of predictors by considering the following three cases.

- **Short Query Set**: Only the three most frequent terms of claim sections are used.
- **Medium Query Set**: Only the 8 most frequent terms of claim sections are used.
- **Long Query Set**: Only the 20 most frequent terms of claim sections are used.

We run experiments for the three types of queries to check the impact of query length on the effectiveness of the predictors. For each query set, 33 % of total queries are used for training while the rest are used for testing the effectiveness of the prediction function. We use the following state-of-the-art retrieval models in order to analyze the performance of predictors.

- **tfidf**: The *tfidf* (term frequency inverse document frequency) is a retrieval model often used in information retrieval. It is a statistical measure used to evaluate how important a query terms is to a document. The importance increases proportionally to the number of times a term appears in the document but is offset by the frequency of the term in the collection. The standard *tfidf* retrieval model is described as follow:

$$tfidf(d,q) = \sum_{t \in q} \frac{tf_{t,d}}{|d|} \log \frac{|D|}{df_t} \tag{1}$$

  $tf_{t,d}$ is the term frequency of query term t in d, and $|D|$ is the total number of documents in the collection. df_t represents the total number of documents containing t.
- **BM25**: Okapi BM25 arguably is one of the most important and widely used information retrieval model. It is a probabilistic function and nonlinear combination of three key attributes of a document: term frequency $t_{t,d}$, document frequency df_t, and the document length $|d|$. The effectiveness of BM25 is controlled by two parameters k and b. These parameters control the contributions of term frequency and document length. If $k = 0$, the function reduces to 1 and the relevance scores of documents are calculated solely based on the occurrences of query terms across the collection only. The large value of k makes the function nearly linear in $tf_{t,d}$. Typically k is used with $k = 2.0$. This demonstrates the nonlinear contribution of $tf_{t,d}$ to the final document relevance scores. The parameter b controls the length normalization. It is set between 0 and 1. Large values of b (close to 1) simply make high normalization, thus short documents are more favored over long documents. The values are small or b approaches to zero, then the effect of normalization becomes small, and long documents are more favored over short documents due to the their large absolute term frequencies. We used the following standard function of BM25 proposed by [31]:

$$BM25(d,q) = \sum_{t \in q} \log \frac{|D| - df_t + 0.5}{df_t + 0.5} \times \frac{tf_{t,d}(k+1)}{tf_{t,d} + k(1 - b + b\frac{|d|}{|\overline{d}|})} \tag{2}$$

  $|\overline{d}|$ is the average document length in the collection from which the documents are drawn. k and b are two parameters, and they are used with $k = 2.0$ and $b = 0.75$.
- **Language Model with Term Smoothing**

  Language model tries to estimate the relevance of the document by estimating the probabilities of terms in the document. The terms are assumed to occur independently, and the probability is the product of the individual query's terms given the document model M_d of document d:

$$P(q|M_d) = \prod_{t \in q} P(t|M_d) \tag{3}$$

$$P(t|M_d) = \frac{tf_{t,d}}{|d|} \tag{4}$$

  $P(t|M_d)$ is the probability of term t occurring in the collection ($\sum_{d \in D} tf_{t,d} / \sum_{d \in D} |d|$).

  The overall similarity score for the query and the document could be zero if some of the query terms do not occur in the document. However, it is not sensible to rule out a document because it is missing a single or a few terms. For dealing with this, language models make use of smoothing to balance the probability mass between the occurrences of terms present in documents, and the terms not found in the documents. Although there exists many variations of term smoothing, however, Dirichlet (Bayesian) Smoothing (DirS) is widely used for comparison [40].

  DirS makes smoothing dependent on the document length. Since long documents allow us to estimate the language model more accurately, this technique smoothes them less, which is done with the help of a parameter μ. Since the value of μ is added in the document length, small values of μ retrieve less long documents. If the μ is used with large values, then the distinction for difference between document lengths becomes less extreme, and long documents are more favored over short documents. Again, this favoritism mostly occurs in case of long boolean OR queries.

$$P(t|M_d) = \frac{tf_{t,d} + \mu P(t|D)}{|d| + \mu} \tag{5}$$

  According to [40] suggestion, we use the μ with 2,000.

4 Pre-retrieval predictors

This section outlines several individual pre-retrieval predictors which model the notion of predicting the quality of queries.

1. **AvIDF**: *AvIDF* determines the query quality on the basis of specificity of a query, relying on the average of the inverse document frequency (idf) of the query terms. A term that occurs in many documents can be expected to have a high term frequency in the collection; thus, decreasing the specificity of a query [18].

$$AvIDF = \frac{1}{|q|} \sum_{t \in q} \left[\log \frac{|D|}{df_t} \right] \tag{6}$$

2. **AvICTF**: Instead of using idf, *AvICTF* relies on collection frequencies of query terms for calculating the specificity of a query [18].

$$AvICTF = \frac{1}{|q|} \sum_{t \in q} \left[\log \frac{|V|}{cf_t} \right] \tag{7}$$

$|V|$ represents the set of distinct terms of the collection, and cf_t is the total term count of t in collection D.

3. **Simplified Query Clarity (SCS)**: Query clarity refers to the specialty/ambiguity of a query. According to [7], the clarity (or on the contrary, the ambiguity) of a query is an intrinsic feature of a query, which has an important impact on the effectiveness of retrieval models. Their proposed clarity score is based on the sum of the Kullback Leibler divergence of the query model from the collection model, and this involves computation of relevance scores for the query model, which is time-consuming. To avoid this expensive computation, [18] proposed a simplified clarity score as a comparable pre-retrieval performance predictor. It is calculated as:

$$SumSCS = \sum_{t \in q} \frac{1}{|q|} \log_2 \frac{\frac{1}{|q|}}{\frac{cf_t}{|V|}} \tag{8}$$

4. **Sum of Collection Query Similarity (SumSCQ)**: This query quality predictor is based on the similarity between the collection and a query [41]. The authors argue that a query that is similar to the collection as a whole is more likely to have higher effectiveness, since the similarity is an indicator of whether documents answering the information need are contained in the collection. [41] used term frequency and inverse document frequency as evidence for checking the similarity between the query and the collection. As the query score increases with increased collection term frequency and increased inverse document frequency, terms that appear in few documents many times are favored. Those terms can be seen as highly specific, as they occur in relatively few documents, while at the same time, they occur often enough to be important to the query:

$$SumSCQ = \sum_{t \in q} \left(1 \cdot \log(cf_t) \cdot \log\left(1 + \frac{|D|}{df_t} \right) \right) \tag{9}$$

5. **Average of Collection Query Similarity (AvgSCQ)**: *AvSCQ* is the average *SCQ* similarity over all query terms:

$$AvgSCQ = \frac{1}{|q|} \sum_{t \in q} \left(1 \cdot \log(cf_t) \cdot \log\left(1 + \frac{|D|}{df_t} \right) \right) \tag{10}$$

6. **Maximum of Collection Query Similarity (MaxSCQ)**: *MaxSCQ* relies on the maximum *SCQ* collection query similarity score over all query terms.
7. **Sum of Term Weight Variability (SumVAR)**: *SumVAR* exploits the distribution of term weights across the collection [41]. If the term weights across all documents containing query's term t are similar, there is little evidence for a retrieval model on how to rank those documents given t, and thus different retrieval algorithms are likely to produce widely different rankings. Conversely, if the term weights differ widely across the collection, ranking becomes easier and different retrieval models are expected to produce similar rankings. This predictor is calculated as follows:

$$SumVAR = \sum_{t \in q} \sqrt{\frac{1}{df_t} \sum_{d \in D_t} (w_{t,d} - \hat{w_{t,d}})^2} \tag{11}$$

D_t represents the set of all documents having term t. $w_{t,d}$ is the term weight of t within document d and it is based on *tfidf* weighting, $\hat{w_{t,d}}$ is the average weight of $w_{t,d}$ over all documents containing t.

8. **Average of Term Weight Variability (AvgVAR)**: *AvgVAR* is the average *VAR* similarity over all query terms:

$$AvgVAR = \frac{1}{|q|} \sum_{t \in q} \sqrt{\frac{1}{df_t} \sum_{d \in D_t} (w_{t,d} - \hat{w_{t,d}})^2} \tag{12}$$

9. **Maximum of Term Weight Variability (MaxVAR)**: *MaxVAR* relies on the maximum *VAR* similarity over all query terms.

5 Experimental analysis of pre-retrieval predictors

Query performance prediction aims to identify whether a set of search results is likely to contain useful answers. The established information retrieval methodology for this type of

investigation involves testing the performance of a predictor across a set of queries that are run on a collection of documents. The performance of the predictor is measured by calculating the correlation between the predicted performance levels with retrieval model effectiveness.

In the query performance prediction literature, Spearman's rank order correlation is widely used for analyzing correlation (relationship) between predictors and retrieval models effectiveness. The correlation score close to $+1$ or close to -1 indicates that there exists a high relationship between the predictor and retrieval model's effectiveness. This indicates that on the basis of predictor it is possible to accurately predict the quality of query. When the correlation score is close to 0 then this means that there exists no correlation between the predictor and the retrieval model's effectiveness. Thus on the basis of given predictor, it is difficult to predict the quality of queries.

Information retrieval experimentation has a strong underlying experimental methodology as used for example in the ongoing series of Text REtrieval Conferences (TREC): a set of queries is run on a static collection of documents, with each query returning a list of answer resources. Humans assess the relevance of each document-query combination, and from this a variety of system effectiveness metrics can be calculated.

- **Recall**: Recall cares about all relevant (judged) documents. It is the ratio of the number of retrieved relevant documents relative to the total number of documents in the collection that are desired to retrieve.

$$Recall = \frac{tp}{tp + fn} \tag{13}$$

  tp represents the total number of relevant documents retrieved. fn represents the false negative, the documents that are relevant but could not retrieve.
- **Precision**: Precision is the ratio of the number of retrieved relevant documents relative to the total number of retrieved documents. Precision measures the quality of the rank lists. However, since it does not consider the total number of relevant documents, therefore a result list consisting of just few retrieved and relevant documents might provide high precision than a large result list with many relevant documents.

$$Precision = \frac{tp}{tp + fp} \tag{14}$$

  fp represents the false positive, the documents that are retrieve but are not relevant.

  Recall and precision are always used with rank cutoff levels. In our experiments we measured the Recall with $Recall@30$, and Precision with $Precision@30$ rank cutoff levels.
- **Mean Average Precision (MAP)**: Precision and Recall are not sensitive to the ranking order of documents (i.e., they do not consider how efficiently different retrieval models retrieve the relevant documents at the top ranked positions). Average precision cares this factor by averaging the precision values obtained after each relevant document found. Thus a retrieval model that ranks a large number of relevant documents at the top ranked positions would provide good average precision. It is calculated using the following equation.

$$AveP(q) = \frac{\sum_{d \in D_q} (Precision@k_{dg}(q)) \cdot rel(d)}{tp + fn} \tag{15}$$

  D_q represents the set of retrieved documents of a query q, and k_{dq} is the rank of a document d in D_q. $rel(d)$ returns 1, if d is a relevant judged document of q, otherwise 0. The mean average precision (MAP) is used for the average precision figures over a number of different queries.

$$MAP = \frac{\sum_{q \in Q} AveP(q)}{|Q|} \tag{16}$$

Tables 1, 2, and 3 show the performance of pre-retrieval predictors with *tfidf*, *BM25* and *DirS* on three query sets. Overall, it is clear that the length of query creates a strong effect on the performance of all predictors. On long queries the correlations of all predictors are low. This is because, long queries have many verbose terms and these drift the accuracy of predictors. The results show that the similarity between a query and the collection (SCQ) can provide useful information for the prediction of how well a query will perform. The most effective among the three collection based predictors is *MaxSCQ*, which considers the alignment between the most similar query term and the collection overall. If we only compare *AvICTF* and *AvIDF*, then across all query sets, *AvIDF* has slightly better performance than *AvICTF*. The only difference between *AvIDF* and *AvICTF* is *doccount* and *termcount*. *AvICTF* relies on *termcount* and *AvIDF* relies on *doccount*. On the basis of results, we can conclude that *docount* is somewhat more reliable than *termcount*. The performance of *SCS* is comparable to *AvICTF*, but always slightly worse than *AvIDF*. Predictors based on term weight variability (*SumVAR*, *AvgVAR* and *MaxVAR*) have overall good performance on different query sets.

6 Genetic programming based query performance prediction (PredGP)

Since the information represented by different search features is complementary, it is natural to combine features.

Table 1 Performance of pre-retrieval predictors with *tfidf*, *BM25*, and *DirS* on three query sets. The effectiveness scores of queries are calculated with *Recall@30*

Pre-retrieval predictors	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
SCS	0.345	0.271	0.279	0.235	0.210	0.189	0.171	0.214	0.163
SumSCQ	⋆**0.380**	0.315	0.309	0.285	0.210	0.174	⋆**0.222**	0.180	0.130
AvgSCQ	0.372	0.306	0.030	0.276	0.203	0.165	0.221	0.175	0.122
MaxSCQ	⋆**0.380**	0.315	0.309	0.285	0.210	0.174	⋆**0.222**	0.180	0.130
SumVAR	0.359	⋆**0.335**	0.325	0.274	0.253	0.216	0.216	0.195	0.166
AvgVAR	0.361	⋆**0.336**	0.326	0.279	0.259	0.216	0.216	0.195	0.166
MaxVAR	0.370	0.344	⋆**0.335**	⋆**0.292**	⋆**0.269**	⋆**0.230**	0.217	0.210	⋆**0.175**
AvICTF	0.355	0.288	0.294	0.228	0.212	0.187	0.177	⋆**0.226**	0.172
AvIDF	0.371	0.305	0.300	0.253	0.205	0.161	0.198	0.202	0.142

Table 2 Performance of pre-retrieval predictors with *tfidf*, *BM25*, and *DirS* on three query sets. The effectiveness scores of queries are calculated with *Precision@30*

Pre-retrieval predictors	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
SCS	0.336	0.278	0.162	0.274	0.206	0.159	0.287	0.184	0.130
SumSCQ	0.398	⋆**0.353**	⋆**0.249**	⋆**0.309**	0.239	⋆**0.204**	⋆**0.311**	0.211	⋆**0.179**
AvgSCQ	0.341	0.339	0.245	0.301	0.231	0.200	0.303	0.201	0.172
MaxSCQ	0.398	⋆**0.353**	⋆**0.249**	⋆**0.309**	0.239	⋆**0.204**	⋆**0.311**	0.211	⋆**0.179**
SumVAR	0.284	0.288	0.208	0.297	0.233	0.164	0.296	0.205	0.153
AvgVAR	0.287	0.293	0.208	0.296	0.236	0.164	0.295	0.209	0.153
MaxVAR	0.346	0.310	0.212	0.303	⋆**0.247**	0.168	0.303	⋆**0.222**	0.159
AvICTF	0.393	0.285	0.166	0.282	0.209	0.168	0.295	0.185	0.132
AvIDF	⋆**0.404**	0.327	0.213	0.303	0.233	0.202	0.309	0.200	0.160

Table 3 Performance of pre-retrieval predictors with *tfidf*, *BM25*, and *DirS* on three query sets. The effectiveness scores of queries are calculated with *MAP*

Pre-retrieval predictors	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
SCS	0.315	0.285	0.288	0.268	0.216	0.194	0.173	0.209	0.156
SumSCQ	0.389	0.356	0.347	⋆**0.358**	0.264	0.231	⋆**0.264**	⋆**0.233**	0.176
AvgSCQ	0.387	0.352	0.344	0.352	0.260	0.225	0.263	0.222	0.176
MaxSCQ	0.389	0.356	0.347	⋆**0.358**	0.264	0.231	⋆**0.264**	⋆**0.223**	0.176
SumVAR	0.391	0.368	0.360	0.333	0.288	0.259	0.248	0.211	⋆**0.187**
AvgVAR	0.393	0.368	⋆**0.361**	0.339	0.293	0.263	0.248	0.211	⋆**0.187**
MaxVAR	⋆**0.396**	⋆**0.371**	⋆**0.362**	0.350	⋆**0.302**	⋆**0.275**	0.249	0.212	⋆**0.187**
AvICTF	0.317	0.291	0.291	0.275	0.226	0.196	0.180	0.215	0.162
AvIDF	0.369	0.336	0.329	0.327	0.250	0.211	0.228	0.226	0.169

Among the possible feature combination techniques, genetic programming (GP) based features combination is widely used in IR for automatically evolving effective feature combination. The use of GP in IR is not a new research idea. In the past few years, there have been made several attempts on evolving effective retrieval models using genetic programming [4, 6, 8, 10–13, 29, 36, 38].

The learning process for evolving efficient combination of predictors is formalized as follows. Given a query collection Q, predictors scores and effectiveness scores of queries,

a feature extractor produces a vector of features that describe the match between the effectiveness scores of queries and the predictors scores of queries in Q. The range of features that are used in this study includes a set of pre-retrieval predictors as described in Sect. 4. During the evaluation process, GP tries to optimize a predictor combination $\widehat{f}$ such that the correlation between the combination of predictors and effectiveness scores could be maximized. Once all the generations finish their processing the individual having the highest correlation is returned as an output.

6.1 Features normalization

Formally, a feature f_k is a pre-retrieval predictor. Before starting the processing of GP a query-based normalization on all features is performed in order to normalize all feature values into a range of [0, 1]. For a query q, the normalized value of $f_k(q)$ is calculated by Eq. (17), where $\max(f_k(q))$ and $\min(f_k(q))$ are the maximum and minimum values of $f_k(q)$ respectively for feature f_k in Q.

$$f_k(q) = \frac{f_k(q) - \min(f_k(q))}{\max(f_k(q)) - \min(f_k(q))} \tag{17}$$

6.2 Genetic programming framework

Genetic programming is a branch of evolutionary computing. It helps to solve exhaustive search space problems without requiring the user to specify the structure of the solution in advance. There are two main steps in genetic programming; (a) initial population creation, and (b) recombination with the existing population to evolve better solutions. Generally, the initial population (generation) is created randomly and it is modeled in the form of trees. Each tree represents a solution, structured by several nodes. Nodes can be either operators (functions) or operands (terminals). From the initial population recombination occurs to evolve better solutions (next generation's population). This is performed either by a crossover or a mutation. In order to produce better populations it is important to select better solutions from the current population in a larger percentage. This selection is done by a fitness function that measures how well an individual performs in its environment. The process of recombination iterates until a predefined number of generations has been created or no further improvements can be observed. Some important parameters in GP are; (a) the population size, (b) the number of generations, (c) the depth of tree, (d) the function set, and (e) the terminal set.

The proposed GP based learning framework is summarized as follows. An individual I of the current population represents a combination of predictors. It is expressed as a functional expression of three components: Sv (query quality predictors), Sc (constants), and Sop (operators). Sv is a set of pre-retrieval predictors. Sc is a set of predefined real numbers ranging from 0 to 1. Sop is a set of arithmetic operators $(+, /, *)$. In the implementation, I is represented as a binary tree structure, in which an internal node is an arithmetic operator and a leaf node is a predictor. In experiments, we explore the maximum depth of individuals up to 6 levels. Furthermore, we perform experiments with up to 150 generations, and with 50 individuals per generation.

The evolution process works as follows. Individuals of the initial population are produced randomly by a ramped half-and-half method [22]. In this method, the individuals are produced randomly. However, it ensures that half of the individuals must not have all the branches of the maximum tree depth. For the reproduction of new individuals for the next generation, the top 10 % individuals of the current generation that exhibit minimum retrieval bias are moved to the next generation without any modification. This is done for the survival of the fittest individuals. Next, the remaining population is produced 70 % by crossover and 30 % by mutation. Parents for crossover are selected with the 5-tournament selection approach. This removes any kind of bias in the parents' selection process. The 5-tournament selection approach randomly selects a few individuals from the previous generation and returns one individual (for the mutation) or two individuals (for the crossover). Crossover is applied on the parents by simply switching their sub-trees to each other. The sub-trees for the crossover are also selected randomly. In this process, two new individuals are produced. In mutation, a mutant is created by randomly choosing an internal node of the selected individual, and then its whole sub-tree is replaced with a randomly generated tree. After the evolution ends it's processing, the output set O representing the best individuals are returned as a candidate solutions.

It is important to define a suitable fitness function for GP to work. The fitness function should reward good individuals and punish bad ones. We use *Recall@30* as a fitness function for analyzing the correlation between predictors' combination and retrieval models effectiveness scores. We consider the following facts in order to determine whether the GP approach presented above can be utilized effectively for evolving effective combination of predictors.

- The fittest (best) individual from each generation computed on the training data. This allows one to verify that the genetic programming method is functioning, as well as indicating that some improvement over base-line-retrieval models can be made.
- The average fitness for each generation can be examined to further ensure that the system is functioning as desired; a sharp initial rise is expected, after which average fitness should slowly (but not monotonically) rise.

We compare the *PredGP* performance with two pre-retrieval combination models. Both these models are non-

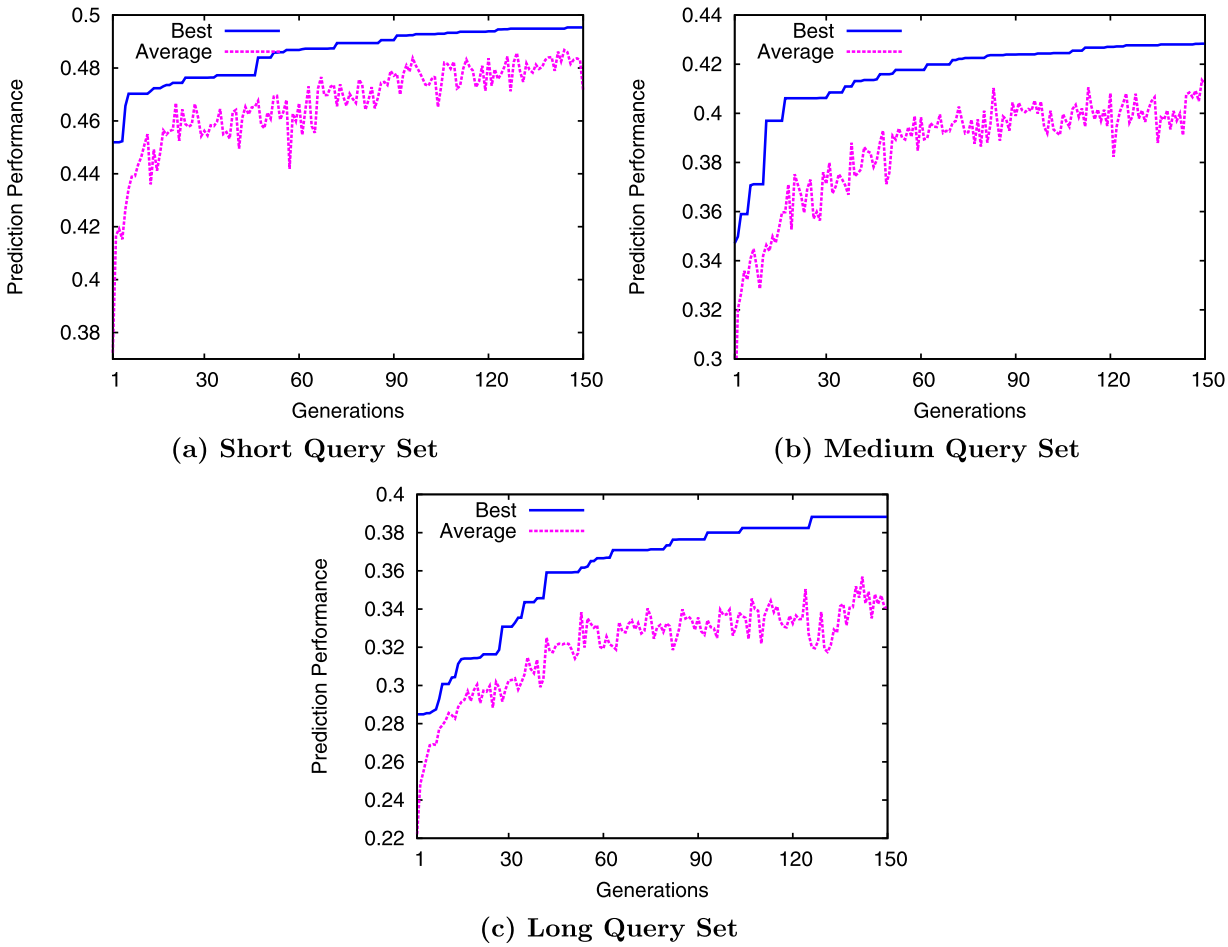

Fig. 1 Prediction accuracy gained by *PredGP* on the basis of average fitness and best-fitness as the generations evolve

backtracking, therefore, we expect that *PredGP* would perform better than these models. The first model combines pre-retrieval predictors using decision tree *(DesTree)* [23]. For this model we use minimum mean square error (MMSE) using the Moore-Penrose pseudo-inverse method in order to estimate the prediction error at each branch and learning decision tree [23]. The second model combines pre-retrieval prediction using multiple linear regression *(multiRegression)* [16]. We trained both models on training dataset with the help of open source Weka machine learning toolkit [15], and test dataset is used for evaluating the performance of both models. Weka does not have a mechanism for analyzing the decision tree prediction accuracy using Moore-Penrose pseudo-inverse. We implement this extension in order train the *DesTree* as recommended in [23].

Figure 1 shows the improvement in prediction performance that is gained by the fittest individual of each generation on three training datasets. As expected, performance increases in a non-strictly monotonic manner as the generations evolve. Figure 1 also shows how the average fitness improves as the generations evolve. Within the first few generations, a large number of individuals yield only nonsensical weights. This gives poor correlation between predictors combinations and effectiveness scores of queries. These individuals quickly die out, resulting in the dramatic improvement on average fitness for the first few generations. Once the system stabilizes, average fitness rises very slowly over the course of a large number of generations. After all generations finished their execution, the fittest individual (GP learning framework), having highest Spearman's rank correlation coefficient score, is used for performance analysis on test datasets. Table 7 shows the fittest functions that are evolved with GP on three training datasets. Tables 4, 5, and 6 show the performance of *PredGP* on test datasets with *Recall@30*, *Precision@30*, and *MAP*. If we compare *PredGP* with single best pre-retrieval predictor. Then *PredGP* achieves significant improvement with *Recall@30* = 58 %, *Precision@30* = 49 %, and *MAP* = 43 % respectively. Two other predictor combination methods (*multiRegression* and *DesTree*) also achieve high improvement as compared to single pre-retrieval predictors. However, due to better search space exploration, *PredGP*

Table 4 Prediction performance of *PredGP*, *multiRegression* and *DesTree* on different query sets. The effectiveness scores of queries are calculated with *Recall@30*

Prediction model	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
PredGP	**⋆0.493**	**⋆0.465**	**⋆0.463**	**⋆0.401**	**⋆0.414**	**⋆0.387**	**⋆0.373**	**⋆0.381**	**⋆0.384**
multiRegression	0.421	0.406	0.426	0.321	0.367	0.349	0.295	0.335	0.346
DesTree	0.402	0.402	0.402	0.313	0.343	0.334	0.291	0.311	0.332

Table 5 Prediction performance of *PredGP*, *multiRegression* and *DesTree* on different query sets. The effectiveness scores of queries are calculated with *Precision@30*

Prediction model	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
PredGP	**⋆0.506**	**⋆0.422**	**⋆0.437**	**⋆0.458**	**⋆0.370**	**⋆0.376**	**⋆0.373**	**⋆0.344**	**⋆0.293**
multiRegression	0.430	0.375	0.399	0.394	0.303	0.326	0.301	0.285	0.246
DesTree	0.412	0.364	0.374	0.389	0.295	0.311	0.294	0.261	0.230

Table 6 Prediction performance of *PredGP*, *multiRegression* and *DesTree* on different query sets. The effectiveness scores of queries are calculated with *MAP*

Prediction model	Short queries			Medium queries			Long queries		
	tfidf	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*	*tfidf*	*BM25*	*DirS*
PredGP	**⋆0.488**	**⋆0.468**	**⋆0.461**	**⋆0.458**	**⋆0.419**	**⋆0.390**	**⋆0.385**	**⋆0.376**	**⋆0.367**
multiRegression	0.416	0.394	0.434	0.383	0.367	0.369	0.303	0.321	0.321
DesTree	0.403	0.388	0.419	0.370	0.344	0.362	0.305	0.302	0.305

Table 7 Fittest individuals evolved with genetic programming for three training datasets

Query set	Retrieval model
Short	$\widehat{f1}(q) = \left((SCS + MaxSCQ) + \left(\frac{(SCS+0.1)}{(SCS + \frac{\frac{SCS}{AvIDF}}{0.2})} + \left((0.3 * SumVAR) * \frac{(SCS+0.1)}{(SCS + \frac{SCS}{AvIDF})}\right)\right)\right)$
Medium	$\widehat{f2}(q) = \left(\left(\left(\frac{AvgSCQ}{0.8} * 0.6\right) + (0.8 * SCS)\right) + \left(\left(\frac{AvICTF}{SCS} + \frac{MaxSCQ}{AvIDF}\right) + \left(\frac{AvgSCQ}{SCS} * (AvIDF + SCS)\right)\right)\right)$
Long	$\widehat{f3}(q) = \left(\left(\left(\frac{AvICTF}{SumVAR} + SCS\right) + \left(\frac{\frac{SumVAR}{lBF}}{\frac{AvICTF}{SumVAR}} * 0.6\right)\right) + \left(\frac{\frac{MaxSCQ}{AvIDF}}{\frac{AvgSCQ}{SCS}} + \frac{\frac{AvICTF}{SumVAR}}{\frac{MaxSCQ}{SCS}}\right)\right)$

performs significantly better than *multiRegression* and *DesTree*. This shows the merit of evolving predictors combination using genetic programming.

7 Conclusion

In this paper, a learning method, *PredGP*, is proposed to address the task of learning to predict the quality of queries. *PredGP* employs genetic programming to learn an effective prediction function by combining various pre-retrieval predictors. Experiments are conducted to evaluate the performance of *PredGP* using TREC chemical prior-art retrieval task dataset. Several single pre-retrieval predictors are compared with *PredGP*. The results show that *PredGP* performs significantly better than single pre-retrieval predictors.

There are still many open issues related to this research. First is how to improve the performance of *PredGP* in case of long queries, which are the type of queries that a search system is expected to serve in case of professional search (e.g. medical, patent or legal retrieval). One of the directions we would like to consider is looking for additional features that indicate the query difficulty but on the other hand do not depend on the query length.

Second main issue is the amount of training data for evolving prediction function. There is much evidence in our experiments that the quality of prediction is increased with the number of training examples. There is need to investigate how the training data for learning a prediction function can be accumulated in automatic, or at least semi-automatic manner.

References

1. Aslam JA, Pavlu V (2007) Query hardness estimation using Jensen-Shannon divergence among multiple scoring functions. In: Proceedings of the 29th European conference on IR research, ECIR'07, pp 198–209
2. Banerjee S, Pedersen T (2003) Extended gloss overlaps as a measure of semantic relatedness. In: Proceedings of the 18th international joint conference on artificial intelligence, IJCAI'03, pp 805–810
3. Buckley C (2004) Topic prediction based on comparative retrieval rankings. In: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '04, pp 506–507
4. Chen H (1995) Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. J Am Soc Inf Sci Technol 46(3):194–216
5. Collins-Thompson K, Bennett PN (2009) Estimating query performance using class predictions. In: Proceedings of the 32nd international ACM SIGIR conference on research and development in information retrieval, SIGIR '09, pp 672–673
6. Cordón O, Herrera-Viedma E, López-Pujalte C, Luque M, Zarco C (2003) A review on the application of evolutionary computation to information retrieval. Int J Approx Reason 34:241–264
7. Cronen-Townsend S, Zhou Y, Croft WB (2002) Predicting query performance. In: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '02, pp 299–306
8. Cummins R, O'Riordan C (2005) Evolving general term-weighting schemes for information retrieval: tests on larger collections. Artif Intell Rev 24(3–4):277–299
9. Diaz F (2007) Performance prediction using spatial autocorrelation. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '07, pp 583–590
10. Diaz-Aviles E, Nejdl W, Lars S-T (2009) Swarming to rank for information retrieval. In: GECCO '09, proceedings of the 11th annual conference on genetic and evolutionary computation, New York, NY, USA. ACM, New York, pp 9–16
11. Fan W, Fox EA, Pathak P, Wu H (2004) The effects of fitness functions on genetic programming-based ranking discovery for web search. J Am Soc Inf Sci Technol 55(7):628–636
12. Fan W, Gordon MD, Pathak P (2004) A generic ranking function discovery framework by genetic programming for information retrieval. Inf Process Manag J 40(4):587–602
13. Fan W, Gordon MD, Pathak P (2005) Genetic programming-based discovery of ranking functions for effective web search. J Manag Inf Syst 21(4):37–56
14. Fujii A, Iwayama M, Kando N (2007) Introduction to the special issue on patent processing. Inf Process Manag J 43(5):1149–1153
15. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11:10–18
16. Hauff C (2010) Predicting the effectiveness of queries and retrieval systems. Ph.D. Dissertation, University of Twente
17. He B, Ounis I (2004) Inferring query performance using pre-retrieval predictors. In: SPIRE. Lecture notes in computer science. Springer, Berlin, pp 43–54
18. He B, Ounis I (2006) Query performance prediction. Inf Syst J 31(7):585–594
19. He J, Larson M, De Rijke M (2008) Using coherence-based measures to predict query difficulty. In: Proceedings of the IR research, 30th European conference on advances in information retrieval, ECIR'08, pp 689–694
20. Itoh H (2004) Patent retrieval experiments at ricoh. In: Proc. of NTCIR '04: NTCIR-4 workshop meeting
21. Jensen EC, Beitzel SM, Grossman D, Frieder O, Chowdhury A (2005) Predicting query difficulty on the web by learning visual clues. In: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '05, pp 615–616
22. Koza JR (1992) A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In: Proceedings of IJCNN international joint conference on neural networks, vol IV. IEEE Press, New York, pp 310–318
23. Kwok KL (2005) An attempt to identify weakest and strongest queries. In: Predicting query difficulty, SIGIR 2005 workshop (2005)
24. Leskovec J, Dumais S, Horvitz E (2007) Web projections: learning from contextual subgraphs of the web. In: Proceedings of the 16th international conference on world wide web, WWW '07, pp 471–480
25. Lupu M, Huang J, Zhu J, Tait J (2009) TREC-CHEM: large scale chemical information retrieval evaluation at trec. SIGIR Forum 43(2):63–70
26. Mase H, Matsubayashi T, Ogawa Y, Iwayama M, Oshio T (2005) Proposal of two-stage patent retrieval method considering the claim structure. ACM Trans Asian Lang Inf Process 4(2):190–206
27. Mothe J, Tanguy L (2005) Linguistic features to predict query difficulty—a case study on previous trec campaigns. In: Predicting query difficulty, SIGIR 2005 workshop
28. Patwardhan S, Pedersen T (2006) Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In: Proceedings of the EACL 2006 workshop making sense of sense—bringing computational linguistics and psycholinguistics together, pp 1–8
29. Pham MQN, Nguyen ML, Bach NX, Shimazu A (2012) A learning-to-rank method for information updating task. Appl Intell 37:499–510
30. Rada R, Mili H, Bicknell E, Blettner M (1989) Development and application of a metric on semantic nets. IEEE Trans Syst Man Cybern 19(1):17–30
31. Robertson SE, Walker S (1994) Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In: SIGIR '94: proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval, Dublin, Ireland, pp 232–241
32. Scholer F, Williams HE, Turpin A (2004) Query association surrogates for web search: research articles. J Am Soc Inf Sci Technol 55:637–650
33. Shinmori A, Okumura M, Marukawa Y, Iwayama M (2003) Patent claim processing for readability: structure analysis and term explanation. In: Proceedings of the ACL-2003 workshop on patent corpus processing, vol 20, pp 56–65
34. Singhal A, Salton G, Buckley C (1995) Length normalization in degraded text collections. In: Proceedings of fifth annual symposium on document analysis and information retrieval, pp 15–17
35. Takaku M, Oyama K, Aizawa A (2006) An analysis on topic features and difficulties based on web navigational retrieval experiments. In: Proceedings of the third Asia conference on information retrieval technology, AIRS'06, pp 625–632

36. Verberne S, van Halteren H, Theijssen D, Raaijmakers S, Boves L (2011) Learning to rank for why-question answering. Inf Retr 14:107–132
37. Vinay V, Cox IJ, Milic-Frayling N, Wood K (2006) On ranking the effectiveness of searches. In: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '06, pp 398–404
38. Vrajitoru D (1998) Crossover improvement for the genetic algorithm in information retrieval. Inf Process Manag J 34(4):405–415
39. Yom-Tov E, Fine S, Carmel D, Darlow A (2005) Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '05, pp 512–519
40. Zhai C (2002) Risk minimization and language modeling in text retrieval. Ph.D. Thesis, Carnegie Mellon University
41. Zhao Y, Scholer F, Tsegay Y (2008) Effective pre-retrieval query performance prediction using similarity and variability evidence. In: Proceedings of the IR research, 30th European conference on advances in information retrieval, ECIR'08, pp 52–64
42. Zhou Y, Croft WB (2006) Ranking robustness: a novel framework to predict query performance. In: Proceedings of the 15th ACM international conference on information and knowledge management, CIKM '06, pp 567–574
43. Zhou Y, Croft WB (2007) Query performance prediction in web search environments. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '07, pp 543–550

Shariq Bashir is currently working as a Postdoctoral Associate at center of science and engineering, New York University Abu Dhabi. Before this, he was Assistant Professor of computer science at National University of Computer and Emerging Science, Islamabad. He is an approved Ph.D. supervisor for the Higher Education Commission (HEC) of Pakistan. He received his Ph.D. in Computer Science from Vienna University of Technology, Austria in 2011. His research interests include information retrieval, unsupervised IR systems evaluation, retrieval bias analysis, documents retrievability analysis, query expansion, query performance prediction, learning to rank, opinion-based entity ranking, decision support systems, expert systems and data mining. He has published more than 30 research papers in leading international conferences and journals of information retrieval and data mining domains.