International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Survey on Algorithms Predicting Performance of Keyword Queries

Snehal Borole

Computer Engineering Department, PES Modern College of Engineering, Pune, Maharashtra, India

Abstract: ***Querying using keywords is easily the most widely used form of querying today. Data in database can be easily access with the help of Keyword queries but it may be suffer with low ranking i.e., low precision and/or recall. Responses to keyword searches are often imprecise. It would be beneficial to identify queries having low ranking quality to improve result. Generally in existing systems result is retrieved based on user keyword query, but it may happen that due to low ranking the expected result is discarded. User remains unsatisfied on retrieved result. Hence it is necessary to study different algorithms which help to predict the difficulty level of the keyword query so that alternate query can be used to improve results. This paper provides different algorithms and their comparative study for prediction of difficult keyword queries.***

Keywords: Query performance, query effectiveness, keyword query

1. Introduction

Predicting difficulty of keyword query plays an important role in information retrieval. Mostly User issue keyword query, in response keyword query interface returns a ranked list of answers. If returned answer is not satisfied then rephrase or modify the keyword query in order to improve effectiveness of the search. In recent years, several techniques are proposed for predicting the quality of the keyword queries like clarity score [2], probabilistic retrieval models for semi-structured data [9], IR style algorithm [7] etc.

1.1 Keyword Queries

Querying using keywords is easily the most widely used form of querying today. Data in database can be easily access with the help of Keyword queries. Query performance prediction is useful in a variety of information retrieval (IR) areas such as improving retrieval consistency, query refinement, and distributed IR. The keyword query is work with unstructured dataset, semi structure dataset as well as structured dataset.

Keyword query does not consider structure of a database. In SQL, we specify the schema name for query term but in keyword queries user does not provide desired schema for query term. For instance, query Q1: Godfather on the IMDB database (http://www.imdb.com) does not specify if the user is interested in movies whose title is Godfather or movies distributed by the Godfather Company. Thus, a Keyword Query Interface must find the desired attributes associated with each term in the query. Second, the schema of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities. For example, Q1 may return movies or actors or producers.

The main problem with keyword queries are they can suffer from the low ranking quality. Low ranking quality refers to the low recall or /and low precision.

1.2 Hard Keyword Queries

The queries which are difficult to answer correctly are called hard keyword queries. The researchers propose 3 sources of difficulty for answering a query over a database [1] as follows:

1) More entities matches the term in query:
If more entities match the terms in a query, the query is less specific and it is harder to answer properly. For example: there are more one person named JOEL in the IMDB database and user submits query Q1: JOEL, the keyword query interface must resolve desire JOEL that satisfy user's information need. If the more number of people in IMDB is JOEL then it will be hard to predict the correct answer. As oppose to Q1, Q2: KIM matches the smaller number of people in IMDB, so it is easier for keyword query interface to return relevant answer.

2) Attribute level ambiguity
Each attribute explains a different feature of an entity and defines the context of terms in attribute values of it. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity. For example: Q3: GODFATHER, contains in *title* and the *distributor* of IMDB dataset. Keyword query interface must find out the desired attribute for GODFATHER to find correct answer. Answer for the query Q4: SPEED does not match with any instance of attribute *distributor*, so keyword query interface easily predict the relevant answer for this query.

3) Entity level ambiguity
Each entity set contains the information about a different type of entities and defines another level of context (in addition to the context defined by attributes) for terms. Hence, if a query matches entities from more entity sets, it will have higher entity set level ambiguity.

For example: IMDB has two entity sets mainly one is '*Movie*', contains the information about movie and other is

'Person', contains the information about people who making movies. Consider query Q5: 'MARRIED', the both *Movie* and *Person* contains MARRIED word, so the query become hard to find relevant as keyword query interface does not know whether user is interested in people who are MARRIED or the movies containing MARRIED word. In case of query Q6: COMEDY SUSPENSE is only match with the entities of *Movie* entity set. So it will be less difficult to keyword query interface to find relevant answer as compared to query Q5.

1.3 Ranking Robustness

Ranking robustness in noisy data retrieval refers to a property of a ranked list of documents that indicates how stable the ranking is in the presence of noise brought by the recognition process. Ranking robustness is calculated by comparing the ranking list from the corrupted dataset to the corresponding ranking list of the original dataset using same query and the ranking function. There many methods using ranking robustness principle to calculate the difficulty of query is clarity score [2], SR algorithm [1].

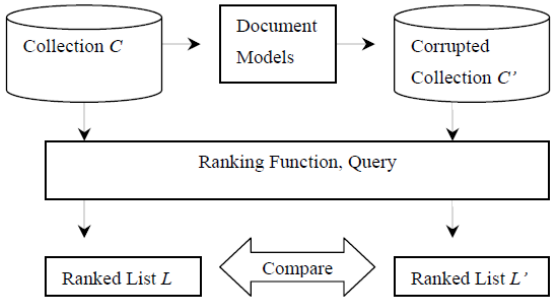

Figure 1: Robustness Score Calculation

2. Related Work

Prediction of query performance has long been of interest in information retrieval. It is invested under a different names query difficulty, query ambiguity and sometimes hard query.

Clarity score method [2] is an attempt to specify the query ambiguity. It uses unigram distribution over terms to estimate the query language model and calculates the threshold value by performing kernel density estimation with automatically setting of the degree of smoothing. Kernel density estimation is a smoothing technique that allows us to estimate the underlying probability density by summing Gaussians centered at each observed data point. The value of threshold is selected heuristically so that the 80% of probability density is below threshold. This method has used TREC dataset for evaluating and predicting query ambiguity. Several successors propose methods to improve the efficiency and effectiveness of clarity score [6], [8].

The robustness algorithm [3] works in following ways, first, perform retrieval with query Q and retrieval function G. Then generate J simulated documents using the document models of the top J documents retrieved and rank the simulated documents with the same query and retrieval function. The similarity between the two ranked lists is computed using the Spearman rank correlation coefficient. Repeat this K times and the average of the Spearman correlation coefficient is the robustness score. This method is also known as Unstructured Robustness Method. The analysis reveals that the robustness score predicts the value of average precision better than the clarity score.

The robustness score formula is given by:

$$\text{Robustness score}(Q,G,C,X) \approx \frac{1}{K}\sum_{i=1}^{K} SimRank\left(L(Q,G,C), L(Q,G,T(i))\right)$$

Where, Q is the query and a document collection of M documents C = (D_1, D_2, …, D_M), G is the retrieval function, and X is the noisy version of database. The $T(i)$ is a sample independently drawn from fx(T) where

T ∈ M(|A| x V).

For the web search engine, researcher has invented Weighted Information Gain (WIG) [5] which gives accuracy in predicting query difficulty. It measures the divergence between the mean retrieval score of top-ranked documents and that of the entire corpus. Researcher focus on two types of queries in web engine: the content based and named-page finding queries.

Researchers also proposed Normalized query commitment (NQC) [6] that predicts the query performance based on estimating the potential amount of query drift in the list of top-retrieved documents using the standard deviation of their retrieval scores. The NQC, WIG and URM are the post-retrieval query prediction techniques over the unstructured database.

IR style algorithm [7] is proposed for keyword search over relational databases. Keyword query is a simply list is keywords and does not need to provide any schema, relation or attribute name. The answer to such a query consists of rank of "tuple trees", which includes tuples from multiple relations that are combined via joins. The IR engine stores an IR index that is inverted index of keyword that appears in database and the list of occurrences of that keyword. Each occurrences of keyword is recorded as tuple attribute pair. The ranking function is introduced to rank the tuple tree that leverages and extends ability to provide keyword search on individual text attributes and rank tuples accordingly. IR style algorithm is hybrid algorithm which combines global pipelined and sparse algorithms.

The sparse algorithm executes one CN (Candidate Network) at a time and updates the current top-k results; The CN is the non-empty tuple sets from the IR Engine. The global pipeline algorithm adopts a more aggressive optimization: it does not execute a CN to its full; instead, at each iteration, it (a) first selects the most promising CN, i.e., the CN with the highest upper bound score; (b) admits the next unseen tuple from one of the CN's non-free tuple sets and join the new tuple with all the already seen tuples in all the other non-free tuple sets.

The probabilistic retrieval models for semi-structured data (PRMS) [9] focused on infers structural information automatically from keyword queries and incorporates this into a retrieval model. It employs a hierarchical language model approach to search over structured data. It computes the language model of each attribute value smoothed by the language model of its attribute. It assigns each attributes a query keyword specific weight, which specifies its contribution in the ranking score.

SR algorithm [1] is Structured Robustness algorithm which uses robustness score for measuring difficulty of a query. It uses the INEX dataset and the SEMSEARCH dataset for the evaluation of a query. In SR algorithm first calculate the top k ranked list and the updated global statistics that are stored in Metadata from the original database. It also uses the inverted index to calculate top k ranked list. Then it generates the noise in the database on the fly during the query processing. It corrupts only the top k entities which are already returned by the ranking module. SR algorithm finds the corrupted result and updated global statistics over the corrupted database and passes those to the ranking module to compute the corrupted ranking list. The each attribute value is corrupted by a combination of three corruption level: on the value itself, its attribute and its entity set.

Let $Y_{t,j}$ be the random variable that represents the frequency of term w_j in attributeT_t. Probability mass function $fY_{t,j}(x_{t,j})$computes the probability of w_j to appear $x_{t,j}$ times inT_t. Similarly, $z_{s,j}$is the random variable that denotes the frequency of term w_j in entity set S_s and probability mass function $fZ_{s,j}(x_{s,j})$ computes the probability of w_j to appear $x_{s,j}$ times in S_s.

The above terms are calculated using Poisson distribution as follows,

$$fX_{a,j}(x_{a,j}) = \frac{e^{-\lambda_{a,j}}\lambda_{a,j}^{x_{a,j}}}{x_{a,j}}$$

$$fY_{t,j}(x_{t,j}) = \frac{e^{-\lambda_{t,j}}\lambda_{t,j}^{x_{t,j}}}{x_{t,j}}$$

$$fZ_{s,j}(x_{s,j}) = \frac{e^{-\lambda_{s,j}}\lambda_{s,j}^{x_{s,j}}}{x_{s,j}}$$

The noise generation module for SR algorithm is given by,

$$\hat{f}X_{a,j}(x_{a,j}) = \begin{cases} \gamma_A fx_{a,j}(x_{a,j}), & if\ w_j \in A_a \\ \gamma_T fy_{t,j}(x_{t,j}), & if\ w_j \notin A_a, w_j \in T_t \\ \gamma_S fz_{s,j}(x_{s,j}), & if\ w_j \notin A_a, T_t, w_j \in S_s \end{cases}$$

Where $0 <= \gamma_A + \gamma_T + \gamma_S <= 1$

If term *w* appears in attribute value *A*, we use only the first term in above Equation to model the frequency of *w* in the noisy version of database. Otherwise, we use the second or third terms if *w* belongs to *T* and *S*, respectively.

The SR algorithm uses the Spearman rank Correlation and Pearson's Correlation to compute the value of similarity function. It ranges from -1 to 1. A value close to 1 means a perfect positive correlation between two ranking and a value close to -1, a perfect negative correlation. If the two ranking have almost no correlation, the correlation coefficient will be close to zero.

Spearman rank correlation formula

$$SR(Q, g, DB, X_{DB}) = \sum_{\vec{x}} Sim(L(Q, g, DB), L(Q, g, \vec{x}))fXDB(\vec{x})$$

Where, Q = query
g is the retrieval function,
X_{DB} is the noisy version of database.
$\vec{x}$ is sample independently drawn from fX_{DB} where
$\vec{x} \in$ M(|A| x V)

The result shows that SR algorithm is more efficient than the other algorithms for predicting difficult queries and resulting ranked top k list [1].

Table 1: Spearman's Correlation of Average Precision against Each Metric when K=10

K = 10				
Method	SR	WIG	CR	URM
INEX	**0.303**	0.242	0.199	0.196
SemSearch	**0.519**	0.27	0.182	-0.012

Table 2: Spearman's Correlation of Average Precision against Each Metric when K=20

K = 20				
Method	SR	WIG	CR	URM
INEX	**0.475**	0.218	0.202	0.27
SemSearch	**0.576**	0.253	0.179	0.074

Table 1 and Table 2 shows that the Spearman's Correlation scores for SR algorithm is more efficient than weighted Information Gain[5], Clarity Score[2], and Unstructured Robustness Method [3].

The drawback of SR algorithm is to spend large portion of the robustness calculation time on the loop that re-ranks the corrupted result. So the researcher invented better approach by modifying this algorithm named as Approximation algorithm. The approximation algorithm improves efficiency of SR algorithm. It is a combination of QAO-Approx (Query specific Attribute values Only Approximation and SGS-Approx (Static Global Stat Approximation.

QAO-Approx: In SR algorithm, the attribute values that do not contain any query term still get corrupted because their attributes or entity sets may contain some query keywords. This drawback of SR algorithm is solved in QAO-Approx. QAO-Approx is based on the observation that the noise in the attribute values that contain query terms dominates the corruption effect. So time spend on corruption can be decreased if only the attribute values are corrupted that contain the query terms.

SGS-Approx: Extracting top k entities from database constitute a very small portion of the database. Thus the

global statistics largely remain unchanged or change very little. Therefore in SGS-Approx, the global statistics remains as it is. Just combining the corruption and ranking module together, re-ranking is done on the fly during corruption. The result shows the combined algorithm delivers the best balance of improvement in efficiency and reduction in effectiveness for both INEX and SEMSEARCH datasets.

3. Conclusion

This paper is focuses on main problem of retrieving appropriate top k results for a keyword query and predicting the difficulty level that is the keyword query is easy or hard. The described algorithms are the various approaches and the Approximation of Structured Robustness Algorithm is the best approach as its performance and accuracy is better than other approaches.

References

[1] Shiwen Cheng, Arash Termehchy, and Vagelis Hristidis, Efficient Prediction of Difficult Keyword Queries over Databases, IEEE transactions on knowledge and data engineering, Vol. 26, no. 6, June 2014.
[2] S. C. Townsend, Y. Zhou, and B. Croft, "Predicting query performance," in Proc. SIGIR, Tampere, Finland, 2002.
[3] Y. Zhou and B. Croft, "Ranking robustness: A novel framework to predict query performance," in Proc. 15th ACM Int. CIKM, Geneva, Switzerland, 2006.
[4] A. Trotman and Q. Wang, "Overview of the INEX 2010 data centric track," in 9th Int. Workshop INEX, Vugh, The Netherlands, 2010.
[5] Y. Zhou and W. B. Croft, "Query performance prediction in web search environments," in Proc. 30th Annu. Int. ACM SIGIR, New York, NY, USA, 2007
[6] A. Shtok, O. Kurland, and D. Carmel, "Predicting query performance by query-drift estimation," ACM Transaction on Information System, Vol. 30, Issue 2,May 2012.
[7] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IRstyle keyword search over relational databases," in Proc. 29th VLDB Conf., Berlin, Germany, 2003.
[8] K. Collins-Thompson and P. N. Bennett, "Predicting query performance via classification," Springer, Volume 5993, 2010.
[9] Jinyoung Kim, Xiaobing Xue, W. Bruce Croft, A Probabilistic Retrieval Model for Semi-structured Data, Springer Transaction on Advances in Information Retrieval, Volume 5478, 2009.

Author Profile

Snehal Borole received a Bachelor of Engineering degree in Computer. Now pursuing Master's of Engineering in Computer from University of Pune. Her interests are in database and data mining field.