The International Journal Of Engineering And Science (IJES)
||Volume|| 2 ||Issue|| 10 ||Pages|| 07-11 ||2013||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

A Practical Approach for Webpage Ranking Using User and Query Dependent Framework

[1] Rani Gangishetti, [2] R. Swapna
[1, 2] Department of Computer Science, KITS, Warangal

ABSTRACT

With the emergence on the deep Internet, searching Internet databases throughout domains including vehicles, real estate property, etc. has changed into a routine job. One on the problems in this context can be ranking the results of a new user dilemma. Earlier techniques for addressing this challenge have used frequencies regarding database valuations, query records, and end user profiles. A standard twine throughout a large number of techniques can be that ranking is finished inside a user- and/or query-independent manner. This particular document offers a new fresh query- and user-dependent strategy with regard to ranking dilemma leads to Internet data source. Most of existing and new ranking model, determined supporting thoughts regarding end user and dilemma similarity, for you to gain a new ranking perform for the offered end user dilemma. This particular perform can be bought from the sparse workload composed of regarding a number of this kind of ranking features made with regard to various user-query frames. The particular model will depend on the particular intuition that equivalent end users screen equivalent ranking inclinations over the outcomes of equivalent requests

Keywords — Ranking, Web Database, Query and User Similarity

Date of Submission: 10, October, 2013 ⟷ Date of Acceptance: 30, October 2013

I. INTRODUCTION

The emergence of the deep Web has led to the proliferation of a large number of Web databases for a variety of applications e.g., airline reservations, vehicle search, real estate scouting. These databases are typically searched by formulating query conditions on their schema attributes. When the number of results returned is large, it is time-consuming to browse and choose the most useful answer(s) for further investigation. Currently, Web databases simplify this task by displaying query results sorted on the values of a single attribute e.g., Price, Mileage, etc. However, most Web users would prefer an ordering derived using multiple attribute values, which would be closer to their expectation. The current sorting-based mechanisms used by Web databases do not perform such ranking. While some extensions to SQL allow manual specification of attribute weights, this approach is cumbersome for **most** Web users. Automated ranking of database results has been studied in the context of relational databases, and although a number of techniques perform query-dependent ranking, they do not differentiate between users and hence, provide a single ranking order for a given query across all users. In contrast, techniques for building extensive user profiles as well as requiring users to order data tuples, proposed for user-dependent ranking, do not distinguish between queries and provide a single ranking order for any query given by the same user. Recommendation i.e., collaborative and content filtering as well as information retrieval systems use the notions of user- and object/item-similarity for recommending objects to users. Although our work is inspired by this idea, there are differences that prevent its direct applicability to database ranking.

In this paper, we propose a user- and query-dependent approach for ranking the results of Web database queries. For a query Qj given by a user Ui, a relevant ranking function (Fxy) is identified from a workload of ranking functions (inferred for a number of user-query pairs), to rank Qj 's results. The choice of an appropriate function is based on a novel similarity-based ranking model proposed in the paper. The intuition behind our approach is for the results of a given query, similar users display comparable ranking preferences, and a user displays analogous ranking preferences over results of similar queries. We decompose the notion of similarity into: **query similarity**, and **user similarity**. While the former is estimated using either of the proposed metrics query-condition or query-result, the latter is calculated by comparing individual ranking functions over a set of common queries between users. Although each model can be applied independently, we also propose a unified model to determine an improved ranking order. The ranking function used in our framework is a linear weighted-sum function comprising of: attribute-weights denoting the significance of individual attributes and value weights representing the importance of attribute values.

II. PREVIOUS WORK

Data integration is the problem of combining information from multiple heterogeneous databases. One step of data integration is relating the primitive objects that appear in the different databases specifically, determining which sets of identifiers refer to the same real-world entities. A number of recent research papers have addressed this problem by exploiting similarities in the textual names used for objects in different databases. (For example one might suspect that two objects from different databases named "USAMA FAYYAD" and "Usama M. Fayyad" " respectively might refer to the same person.) Integration techniques based on textual similarity are especially useful for databases found on the Web or obtained by extracting information from text, where descriptive names generally exist but global object identifiers are rare. Previous publications in using textual similarity for data integration have considered a number of related tasks. Although the terminology is not completely standardized, in this paper we define *entity-name matching* as the task of taking two lists of entity names from two different sources and determining which pairs of names are co-referent (*i.e.*, refer to the same real-world entity). We define *entity-name clustering* as the task of taking a single list of entity names and assigning entity names to clusters such that all names in a cluster are co-referent. Matching is important in attempting to join information across of pair of relations from different databases, and clustering is important in removing duplicates from a relation that has been drawn from the union of many different information sources. Previous work in this area includes work in distance functions for matching and scalable matching and clustering algorithms. Work in *record linkage* is similar but does not rely as heavily on textual similarities. [1]

Important business decisions; therefore, accuracy of such analysis is crucial. However, data received at the data warehouse from external sources usually contains errors: spelling mistakes, inconsistent conventions, etc. Hence, significant amount of time and money are spent on *data cleaning*, the task of detecting and correcting errors in data. The problem of detecting and eliminating duplicated data is one of the major problems in the broad area of data cleaning and data quality. [2]

Many times, the same logical real world entity may have multiple representations in the data warehouse. For example, when Lisa purchases products from SuperMart twice, she might be entered as two different customers due to data entry errors. Such duplicated information can significantly increase direct mailing costs because several customers like Lisa may be sent multiple catalogs. Moreover, such duplicates can cause incorrect results in analysis queries (say, the number of SuperMart customers in Seattle), and erroneous data mining models to be built. We refer to this problem of detecting and eliminating multiple distinct records representing the same real world entity as the *fuzzy duplicate elimination problem*, which is sometimes also called merge/purge, dedup, record linkage problems. This problem is different from the standard duplicate elimination problem, say for answering "select distinct" queries, in relational database systems which considers two tuples to be duplicates if they match exactly on all attributes. However, data cleaning deals with *fuzzy duplicate elimination,* which is our focus in this paper. Henceforth, we use *duplicate elimination* to mean fuzzy duplicate elimination. [2]

Duplicate elimination is hard because it is caused by several types of errors like typographical errors, and *equivalence errors* different (non-unique and nonstandard) representations of the same logical value. For instance, a user may enter "WA, United States" or "Wash., USA" for "WA, United States of America." Equivalence errors in product tables ("winxp pro" for "windows XP Professional") are different from those encountered in bibliographic tables ("VLDB" for "very large databases"), etc. Also, it is important to detect and clean equivalence errors because an equivalence error may result in several duplicate tuples. The class of equivalence errors can be addressed by building sets of rules. For instance, most commercial address cleaning software packages (e.g., Trillium) use rules to detect errors in names and addresses. In this paper, we focus on domain independent duplicate elimination techniques. Domain-specific information when available complements these techniques. Previous domain in dependent methods for duplicate elimination rely on textual similarity functions (e.g., edit distance or cosine metric) predicting that two tuples whose textual similarity is greater than a pre-specified similarity threshold are duplicates. However, using these functions to detect duplicates due to equivalence errors (say, "US" and "United States") requires that the threshold be dropped low enough, resulting in a large number of *false positives* pairs of tuples incorrectly detected to be duplicates. For instance, tuple pairs with values "USSR" and "United States" in the country attribute are also likely to be declared duplicates if we were to detect "US" and "United States" as duplicates using textual similarity. In this paper, we exploit dimensional hierarchies typically associated with dimensional tables in data warehouses to develop an efficient, scalable, duplicate elimination algorithm called Delphi, which significantly reduces the number of false positives without missing out on detecting duplicates. We rely on hierarchies to detect an important class of equivalence errors in each relation, and to significantly reduce the number of false positives. [2]

The critical ingredient of a fuzzy match operation is the similarity function used for comparing tuples. In typical application domains, the similarity function must definitely handle string valued attributes and possibly even numeric attributes. In this paper, we focus only on string-valued attributes, where defining similarity and performing fuzzy matching is more challenging. Given the similarity function and an input tuple, the goal of the fuzzy match operation is to return the *reference tuple* a tuple in the reference relation which is closest to the input tuple. An extension is to return the closest *K* reference tuples enabling users, if necessary, to choose one among them as the target, rather than *the* closest. A further extension is to only output K or fewer tuples whose similarity to the input tuple exceeds a user-specified *minimum similarity threshold*. This formulation is essentially that of the nearest neighbor problem, but there the domain is typically a Euclidean (or other normed) space with well-behaved similarity functions. In our case, the data are not represented in "geometric" spaces, and it is hard to map them into one because the similarity function is relatively complex. [4]

Previous approaches addressing the fuzzy match operation either adopt proprietary domain-specific functions (e.g., Trillium's reference matching operation for the address domain) or use the *string edit distance* function for measuring similarity between tuples. A limitation of the edit distance is illustrated by the following example. The edit distance function would consider the input tuple I3 to be closest to R2, even though we know that the intended target is R1. Edit distance fails because it considers transforming 'corporation' to 'company' more expensive than transforming 'boeing' to 'bon.' However, we know that 'boeing' and '98004' are more informative tokens than 'corporation' and so replacing 'corporation' with 'company' should be considered cheaper than replacing 'boeing' with 'bon' and '98004' with '98014.' In yet another example, note that the edit distance considers I4 closer to R3 than to its target R1. This is because it fails to capture the notion of a token or take into account the common error of token transposition. [4]

One of the major impediments to integrating data from multiple sources, whether by warehousing, virtual integration or web services, is resolving references at the instance level. Data sources have different ways of referring to the same real-world entity. Variations in representation arise for multiple reasons: misspellings, use of abbreviations, different naming conventions, naming variations over time, and the presence of several values for particular attributes. To join data from multiple sources, and therefore, to do any kind of analysis, we must detect when different references refer to the same real-world entity. This problem is known as reference reconciliation. Reference reconciliation has received significant attention in the literature, and its variations have been referred to as record linkage, merge/purge, de-duplication, hardening soft databases, reference matching, object identification and identity uncertainty. Most of the previous work considered techniques for reconciling references to a single class. Furthermore, typically the data contained many attributes with each instance. However, in practice, many data integration tasks need to tackle complex information spaces where instances of multiple classes and rich relationships between the instances exist, classes may have only few attributes, and references typically have unknown attribute values. The main motivation for our work comes from the application of Personal Information Management (PIM), and specifically, supporting higher-level browsing of information on one's desktop. The need for better search tools for the desktop was highlighted by systems like SIS and the Google Desktop search tool. However, these systems provide only keyword search to the desktop's contents. The vision of the Personal Memex and recent systems such as Haystack and Semex emphasize the ability to browse personal information by associations, representing important semantic relationships between objects. To support such browsing, a PIM system examines data from a variety of sources on the desktop (e.g., mails, contacts, ¯les, spreadsheets) to extract instances of multiple classes: Per- son, Message, Article, Conference, etc. In addition, the system extracts associations between the instances, such as senderOf, earlyVersionOf, authorOf, and publishedIn, which then provide the basis for browsing. However, since the data sources are heterogeneous and span several years, a real-world object is typically referred to in several different ways. Reconciliation of the above classes of references guarantees that they mesh together seamlessly, and so the PIM system can provide palatable browsing and searching experiences. [5]

III. PROPOSED SYSTEM

A. Query Condition Similarity

In this model, the similarity between two queries is determined by comparing the attribute values in the query conditions. Consider an example queries. Intuitively, "Honda" and "Toyota" are vehicles with similar characteristics i.e., they have similar prices, mileage ranges, and so on. In contrast, "Honda" is a very different from "Lexus". Similarly, "Dallas" and "Atlanta", both being large metropolitan cities, are more similar to each other than "Basin", a small town. From the above analysis, Q1 appears more similar to Q2 than Q5. In order to validate this intuitive similarity, we examine the relationship between the different values for each attribute in the query conditions. For this, we assume independence of schema attributes, since, availability of appropriate knowledge of functional dependencies and/or attribute correlations is not assumed.

B. Query Result Similarity

In this model, similarity between a pair of queries is evaluated as the similarity between the tuples in the respective query results. The intuition behind this model is that if two queries are similar, the results are likely to exhibit greater similarity. We observe that there exists certain similarity between the results of Q1 and Q2 for attributes such as 'price' and 'mileage' and even 'color' to a certain extent. In contrast, the results of Q5 are substantially different; thus, allowing us to infer that Q1 is more similar to Q2 than Q5. Formally, we define query-result similarity below. Given two queries Q and Q', let N and N' be their query results. The query-result similarity between Q and Q' is then computed as the similarity between the result sets N and N'.

C. Query Independent User Similarity

This model follows the simplest paradigm and estimates the similarity between a pair of users based on all the queries common to them. This model determines the similarity between U1 and U2 using the ranking functions of Q2 and Q7. From the queries, let the query-similarity model indicate that Q2 is most similar to Q1 whereas Q7 is least similar to Q1, and let us consider the user-similarity results. This model will pick U3 as the most similar user to U1. However, if only Q2 which is most similar to Q1 is used, U2 is more similar to U1. Based on our premise that similar users display similar ranking preferences over the results of similar queries, it is reasonable to assume that employing F21 to rank Q1's results would lead to a better ranking order from U1's viewpoint than the one obtained using F31.

D. Cluster Based User Similarity

In order to meaningfully restrict the number of queries that are similar to each other, one alternative is to cluster queries in the workload based on query similarity. This can be done using a simple K-means clustering method. Given an existing workload of m queries (Q1, Q2, ... Qm), each query (Qj) is represented as a m-dimensional vector of the form <sj1, ..., sjm> where sjp represents the query-condition similarity score between the queries Qj and Qp. Using K-means, we cluster m queries into K clusters based on a pre-defined K and number of iterations. Assuming the similarities specified in (Q2 and Q7 are most and least similar to Q1 respectively), for a value of K = 2, the simple K-means algorithm will generate two clusters C1 containing Q1 and Q2 along with other similar queries, and C2 containing Q7 in addition to other queries not similar to Q1. We then estimate the similarity between U1 and every other user only for the cluster C1 since it contains queries most similar to the input query. Using the scenario, U2 would be chosen as the most similar user and F21 would be used to rank the corresponding query results.

E. Top-K User Similarity

Instead of finding a reasonable K for clustering, we propose a refinement, termed top-K user similarity. We propose three measures to determine top-K queries that are most similar to an input query, and estimates the similarity between the user (U1) and every other user. Given an input query Q1 by U1, only the top-K most similar queries to Q1 are selected. However, the model does not check the presence of ranking functions in the workload for these K queries. For instance, based on assumption and using K = 3, Q2, Q3 and Q4 are the three queries most similar to Q1, and hence, would be selected by this model. In the case of workload, similarity between U1 and U2 as well as between U1 and U3 will be estimated using Q2. However, in the case of another workload, similar to the problem in the clustering alternative, there is no query common between U1 and U2 as well as U3. Consequently, similarity cannot be established and hence, no ranking is possible..

IV. RESULTS

The concept of this paper is implemented and different results are shown below, The proposed paper is implemented in Java technology on a Pentium-IV PC with minimum 20 GB hard-disk and 1GB RAM. The propose paper's concepts shows efficient results and has been efficiently tested on different Datasets.

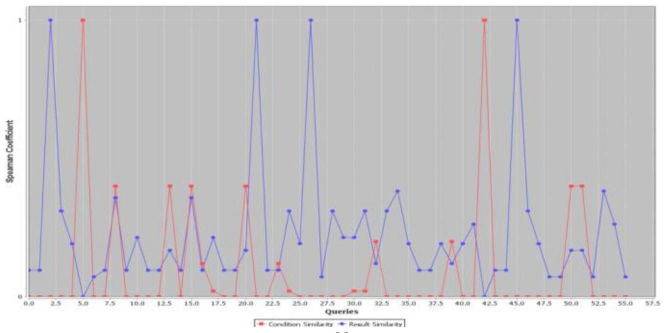

Fig. 1 Speaman Coefficient Comparison

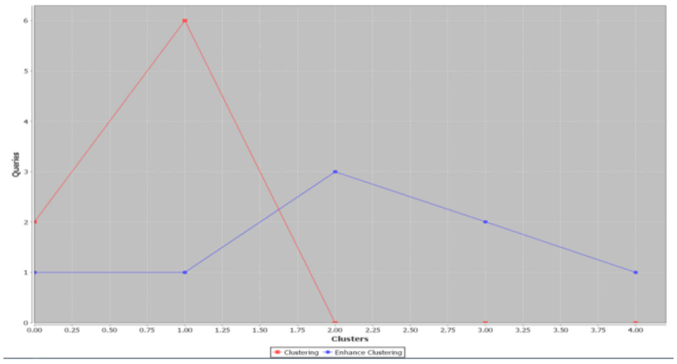

Fig. 1 User Clustering and Enhanced Clustering Comparision

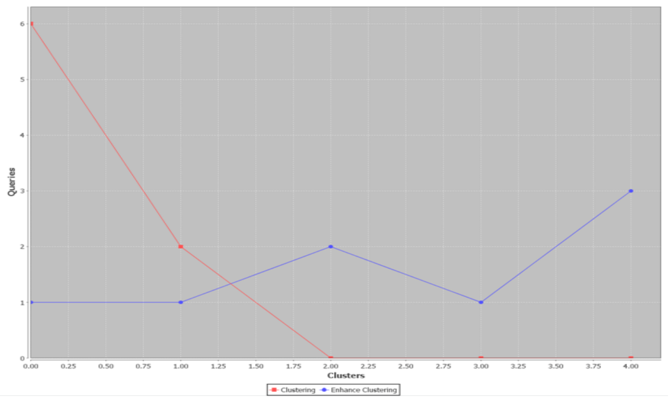

Fig. 3 User Clustering and Enhanced Clustering Comparision

V. CONCLUSIONS

In this paper, we proposed a user and query dependent solution for ranking query results for Web databases. We formally defined the similarity models user, query and combined and presented experimental results over two Web databases to corroborate our analysis. We demonstrated the practicality of our implementation for real-life databases. Further, we discussed the problem of establishing a workload, and presented a learning method for inferring individual ranking functions. Our work brings forth several additional challenges. In the context of Web databases, an important challenge is the design and maintenance of an appropriate workload that satisfies properties of similarity-based ranking. Determining techniques for inferring ranking functions over Web databases is an interesting challenge as well. Another interesting problem would be to combine the notion of user similarity proposed in our work with existing user profiles to analyze if ranking quality can be improved further. Accommodating range queries, usage of functional dependencies and attribute correlations needs to be examined. Applicability of this model for other domains and applications also needs to be explored.

REFERENCES

[1] S. Amer-Yahia, A. Galland, J. Stoyanovich, and C. Yu. From del.icio.us to x.qui.site: recommendations in social tagging sites. In *SIGMOD Conference*, pages 1323–1326, 2008.

[2] J. Basilico and T. Hofmann. A joint framework for collaborative and content filtering. In *SIGIR*, pages 550–551, 2004.

[3] M. K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.

[4] K. C.-C. Chang, B. He, C. Li, M. Patil, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, 2004.

[5] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[6] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 31(3):1134–1168, 2006.

[7] S. Gauch and M. S. et. al. User profiles for personalized information access. In *Adaptive Web*, pages 54–89, 2007.

[8] B. He. Relevance feedback. In *Encyclopedia of Database Systems*, pages 2378–2379, 2009.

[9] S.-W. Hwang. *Supporting Ranking For Data Retrieval*. PhD thesis, University of Illinois, Urbana Champaign, 2005.

[10] G. Koutrika and Y. E. Ioannidis. Constrained optimalities in query personalization. In *SIGMOD Conference*, pages 73–84, 2005.

[11] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, pages 131–142, 2005.

[12] X. Luo, X. Wei, and J. Zhang. Guided game-based learning using fuzzy cognitive maps. *IEEE Transactions on Learning Technologies*, PP(99):1– 1, 2010.

[13] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions of Database Systems*, 29(2):319–362, 2004.

[14] A. Penev and R. K. Wong. Finding similar pages in a social tagging repository. In *WWW*, pages 1091–1092, 2008.

[15] X. Shi and C. C. Yang. Mining related queries from web search engine query logs using an improved association rule mining model. *J. Am. Soc. Inf. Sci. Technol.*, 58:1871–1883, October 2007.