Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Enriched forms for Dynamic Queries

Kavya Siddoju*, Soujanya R
Sreenidhi Institute of Science and technology
Hyderabad, India

Abstract— Current scientific directories and internet databases sustain large and also heterogeneous files. These real-world directories contain hundreds or perhaps thousands involving relations and also attributes. Traditional predefined problem forms can't satisfy different ad-hoc concerns from consumers on people databases. This specific paper offers DQF, some sort of novel databases query kind interface, which will be able to dynamically generate query kinds. The essence of DQF should be to capture some sort of user's desire and position query kind components, assisting him/her making decisions. The generation of a query form is an iterative process and it is guided from the user. In each technology, the technique automatically produces ranking provides of kind components along with the user next adds the desired form components in the query kind. The standing of kind components is dependant on the grabbed user desire. A user could also fill the actual query kind and distribute queries to see the problem result at each technology..

Keywords— Data Mining, Query Forms, Query kind components, Ranking components ,user desired components.

I. INTRODUCTION

Many research works concentrate on database interfaces that assist end users to problem the relational data bank without SQL. QBE (Query-By- Example) as well as Query Type are two most favored database querying interfaces. Currently, query forms have been utilized generally in most real-world business or controlled information systems. Current scientific studies and is effective mainly concentrate on how to generate the problem forms. Existing data bank clients as well as tools create great efforts to help you developers design and style and crank out the problem forms, such as EasyQuery, Chilly Fusion, SAP, Microsoft Access etc. They supply visual interfaces for developers to build or customise query forms. The problem of these tools will be that, they are supplied for the particular professional builders who know their listings, not for end-users. Proposed a system which enables end-users in order to customize the prevailing query type at function time. On the other hand, an end-user will not be familiar while using the database. In the event the database schema is extremely large, it will be difficult for them to find suitable database entities and attributes and also to create wanted query forms.

In this specific paper we all propose a new dynamic problem form method which builds the problem forms using the user's need at function time. The device provides a solution for the particular query screen in huge and complex databases. We use F-measure in order to estimate the particular goodness of a query type. F-measure is usually a typical metric to gauge query effects. This metric can also be appropriate for query forms because problem forms are designed to help end users query the particular database. The goodness of a query form is determined by the problem results generated from the query type. Based for this, we status and propose the possible query type components in order that users may refine the particular query type easily. While using proposed metric, we produce efficient algorithms in order to estimate the particular goodness with the paperion as well as selection type components. Here efficiency is essential because DQF is surely an online method where end users often assume quick response.

II. RELATED WORK

While The importance of result diversification has been recognized since early work on information retrieval. The basic premise is that the relevance of a set of documents depends not only on the individual relevance of its members, but also on how they relate to one another. Ideally, the document set should properly account for the interests of the overall user population. Most of the current work on result diversification make at best only a tacit use of the topics of the query or the documents, and diversification happens by way of similarity functions or conditional relevance distributions defined over documents, or through user feedback. We focus on how to diversify search results making explicit use of knowledge about the topics the query or the documents may refer to. A recent user study suggests that efforts of individual users are usually directed towards finding more information on specific topics of interest, rather than an undirected quest for any new information.

One of the early influential works on diversification is that of Maximal Marginal Relevance (MMR) presented by Carbonell and Goldstein. In their work, a tradeoff between novelty (a measure of diversity) and the relevance of search results is made explicit through the use of two similarity functions, one measuring the similarity among documents, and the other the similarity between document and query.

A parameter controls the degree of tradeoff. As there is no categorization of the document or the query, diversification is conducted through the choice of similarity functions. Zhai et al point out that it is in general insufficient to simply return a set of relevant results. The correlations among the results are also important. This work is later formalized, where Zhai and Lafierty propose a risk minimization framework for information retrieval that allows a user to define an arbitrary loss function over the set of returned documents. This function determines the unhappiness of the user for a given set of results. In order to put the theory to practice, however, requires one to specify the loss function. Bookstein and Chen and Karger both consider information retrieval in the context of ambiguous queries. The basic idea in these works is that documents should be selected sequentially according to the probability of the document being relevant conditioned on the documents that come before. Bookstein's method, however, is limited by the fact that it requires explicit user feedback for relevance after every document is selected. Chen and Karger work around this limitation, and propose an objective function that aims to find at least one relevant document for all users. In both cases, the topic of the query is considered only tacitly. Radlinski, Kleinberg, and Joachims propose a learning algorithm to compute an ordering of search results from a diverse set of orderings. By iterating through all documents in each of the positions while holding fixed the documents in the other positions, they attempt to learn a best ranking of documents using user clicks. Their approach naturally produces a diverse set of results, as user feedback through clicks will diminish the value of similar documents. However, the topics of the query are only considered implicitly. [1]

The notion of similarity or distance for categorical data is not as straightforward as for continuous data. The key characteristic of categorical data is that different values that a categorical attribute takes are not inherently ordered. Thus it is not possible to directly compare two different categorical values. The simplest way to find similarity between two categorical attributes is to assign a similarity of 1 if the values are identical and a similarity of 0 if the values are not identical. For two multivariate categorical data points, the similarity between them will be directly proportional to the number of attributes in which they match. This simple measure is also known as the overlap measure in the literature.

Categorical data (also known as nominal or qualitative multi-state data) has been studied for a long time in various contexts. As mentioned earlier, computing similarity between categorical data instances is not straightforward; owing to the fact that there is no explicit notion of ordering between categorical values. To overcome this problem, several data-driven similarity measures have been proposed for categorical data. The behavior of such measures directly depends on the data. In this section we identify the key characteristics of a categorical data set that can potentially affect the behavior of a data driven similarity measure. [2]

Past research on improving data entry is mostly focused on adapting the data entry interface for user efficiency improvements. Several such papers have used learning techniques to automatically fill or predict a top-k set of likely values. For example, Ali and Meeks predicted values for combo-boxes in web forms and measured improvements in the speed of entry; Ecopod generated type-ahead suggestions that were improved by geographic information; Hermens et al. automatically filled leave-of-absence forms using decision trees and measured predictive accuracy and time savings. In these approaches, learning techniques are used to predict form values based on past data, and each measures the time savings of particular data entry mechanisms and/or the proportion of values their model was able to correctly predict. USHER's focus is on improving data quality, and its probabilistic formalism is based on learning relationships within the underlying data that guide the user towards more correct entries. In addition to predicting question values, we develop and exploit probabilistic models of user error, and target a broader set of interface adaptations for improving data quality, including question reordering and re-asking, and widget customizations that provide feedback to the user based on the likelihood of their entries. Some of the enhancements we make for data quality could also be applied to improve the speed of entry.

Data quality assurance is a prominent topic in the science of clinical trials, where the practice of double entry has been questioned and dissected, but nonetheless remains the gold standard. In particular, Kleinman takes a probabilistic approach toward choosing which forms to re-enter based on the individual performance of data entry staff. This *cross-form* validation has the same goal as our approach of reducing the need for complete double entry, but does so at a much coarser level of granularity. It requires historical performance records for each data entry worker, and does not offer dynamic reconfirmation of individual questions. In contrast, USHER's *cross-question* validation adapts to the actual data being entered in light of previous form submissions, and allows for a principled assessment of the tradeoff between cost (of reconfirming more questions) versus quality (as predicted by the probabilistic model). [3]

As a simple example, return to our DBLife example and suppose that some user would like to know for which conferences a researcher named "Widom" has served on the program committee. The user might issue the keyword query "Widom conference," where "Widom" is a data term and "conference" is a schema term. Using Naïve-AND, we would get no forms, since "Widom" does not appear on any forms. Using Naïve-OR, we would ignore "Widom" and get all forms that contain "conference." Using DI-OR, we would find that "Widom" appears in the person table, so the resulting rewritten keyword query would be "Widom person conference," evaluated with OR semantics. Using DI-AND, we would generate two queries: "person conference" and "Widom conference," evaluate each with AND semantics, and return the union of the results. In this case, "Widom conference" would lead to an empty result, but this would not be the case if "Widom" were both a database term and a schema term. [4]

So far, the work that has been done in the area of personalized databases has focused to keyword-based query recommendation systems. In this scenario, a user can interact with a relational database through a web interface that allows him/her to submit keywords and retrieve relevant content. The personalization process is based on the user's keyword queries, those of previous users, as well as an explicit user profile that records the user's preferences with

regards to the content of the database. Clearly, our approach is different from this scenario in several ways. First, the proposed framework is meant to assist users who pose complex SQL queries to relational databases. Moreover, the system does not require from its users to create an explicit profile. This gives a higher level of flexibility to the system, since the same user might have different information needs during different explorations of the database.

Our inspiration draws from the successful application of user-based collaborative filtering techniques, proposed in the Web context. As previously mentioned, this approach cannot be directly applied to the relational database context. The inherent nature of interactive database exploration raises certain implications that cannot be addressed by the straightforward collaborative filtering approach. In this work, we are based on its premises, but extend them in order to apply them in the database environment.

The challenges of applying data mining techniques to the database query logs are also addressed. In this work, the authors outline the architecture of a Collaborative Query Management System targeted at large-scale, shared-data environments. As part of this architecture, they independently suggest that data mining techniques, such as clustering or association rules can be applied to the query logs in order to provide the users with query suggestions. We should stress, however, that contrary to our work, the authors do not provide any technical details on how such a recommendation system could be implemented. [5]

III. PROPOSED SYSTEM

A. Query Form Construction

In this module each query form corresponds to an SQL query template. In the user interface of a query form *F*, *AF* is the set of columns of the result table. *σF* is the set of input components for users to fill. Query forms allow users to fill parameters to generate different queries. *RF* are not visible in the user interface, which are usually generated by the system according to the database schema. For a query form *F*, *RF* is automatically constructed according to the foreign keys among relations in *RF*. Meanwhile, *RF* is determined by *AF* and *σF*. *RF* is the union set of relations which contains at least one attribute of *AF* or *σF*. Hence, the components of query form *F* are actually determined by *AF* and *σF*. As we mentioned, only *AF* and *σF* are visible to the user in the user interface. In this module we focus on the paperion and selection components of a query form. Ad-hoc join is not handled by our dynamic query form because join is not a part of the query form and is invisible for users. As for "Aggregation" and "Order by" in SQL, there are limited options for users. For example, "Aggregation" can only be MAX, MIN, AVG, and so on; and "Order by" can only be "increasing order" and "decreasing order". Our dynamic query form can be easily extended to include those options by implementing them as dropdown boxes in the user interface of the query form.

B. Ranking Metric

Query forms are designed to return the user's desired result. There are two traditional measures to evaluate the quality of the query results *precision* and *recall*. Query forms are able to produce different queries by different inputs, and different queries can output different query results and achieve different *precisions* and *recalls*, so we use *expected precision* and *expected recall* to evaluate the expected performance of the query form. Intuitively, *expected precision* is the expected proportion of the query results which are interested by the current user. *Expected recall* is the expected proportion of user interested data instances which are returned by the current query form. The user interest is estimated based on the user's click-through on query results displayed by the query form.

C. Ranking Paperion

DQF provides a two-level ranked list for paperion components. The first level is the ranked list of entities. The second level is the ranked list of attributes in the same entity. In *Ranking Attributes*, Suggesting paperion components is actually suggesting attributes for paperion. Let the current query form be *Fi*, the next query form be *Fi*+1. Let *AFi* = {*A*1,*A*2, . . . ,*Aj*}, and *AFi*+1 = *AFi* ∪ {*Aj*+1}, is the paperion attribute we want to suggest for the *Fi*+1, which maximizes *FScore(Fi*+1*)*. We mainly consider the following two data-driven approaches to estimate the conditional probability *Pu*. *Workload-Driven Approach*: where conditional probability of *Pu* could be estimated from query results of historic queries. If a lot of users queried attributes *AFi* and *Aj*+1 together on instance *d*, then *Pu* must be high. and *Schema-Driven Approach where* The database schema implies the relations of the attributes. If two attributes are contained by the same entity, then they are more relevant.

D. Ranking Selection

The selection attributes must be relevant to the current papered entities, otherwise that selection would be meaningless. Therefore, the system should first find out the relevant attributes for creating the selection components. For instance, some databases put all attributes of one entity into a relation. Some databases separate all attributes of one entity into several relations. For enriching selection form components of a query form, the set of paperion components *AF* is fixed. Therefore, *FScore (Fi*+1*)* only depends on *σFi*+1 . For the simplicity of the user interface, most query forms' selection components are simple binary relations in the form of "*Aj op cj*", where *Aj* is an attribute, *cj* is a constant and *op* is a relational operator. The *op* operator could be '=', '≥' '≤' and so on. In each cycle, the system provides a ranked list of such binary relations for users to enrich the selection part. Since the total number of binary relations are so large, we only select the best selection component for each attribute.

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.

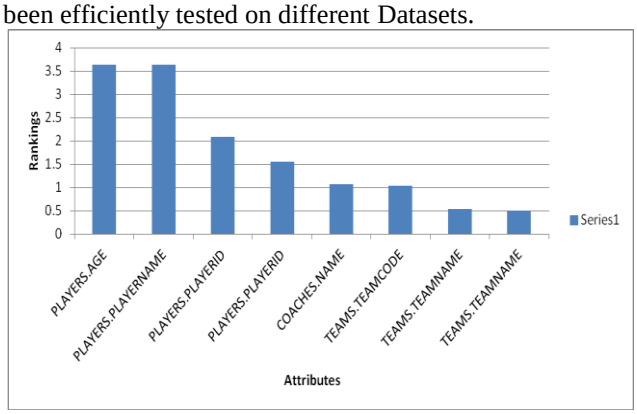

Fig. 1 Attributes Comparison of Query Form

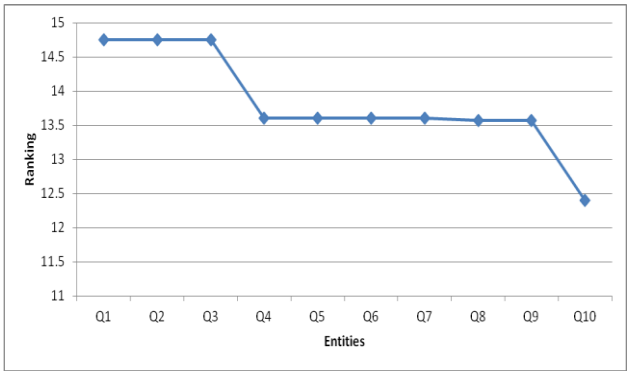

Fig. 2 Ranking Values from Query Form

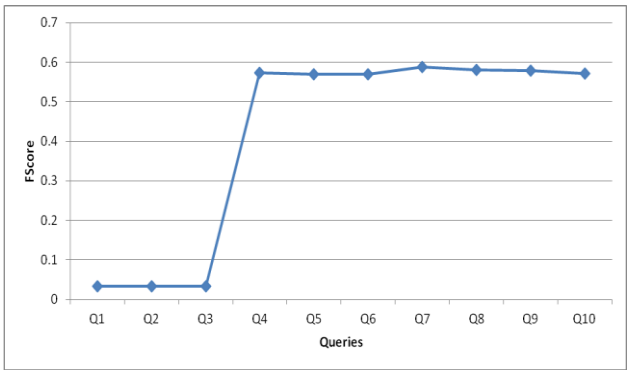

Fig. 3 FScore Values from Query Form

V. CONCLUSIONS

In this particular paper we propose a dynamic query form era approach which in turn helps consumers dynamically generate query sorts. The important idea is with a probabilistic style to status form components determined by user choices. We capture user personal preference using each historical inquiries and runtime feedback such as click-through. Experimental benefits show which the dynamic technique often brings about higher good results rate and simpler query forms weighed against a static technique. The standing of style components also helps it be easier intended for users to customize query forms. As future operate, we can study exactly how our approach might be extended to non relational facts. As money for hard times work, we decide to develop multiple ways to capture this user's interest for the queries aside from the click feedback. For example, we can put in a text-box intended for users to input a few keywords inquiries. The meaning score between your keywords and the query form might be incorporated into the ranking involving form elements at every single step.

REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proc. WSDM*, Barcelona, Spain, Feb. 2009, pp. 5–14.

[2] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *Proc. SDM*, Atlanta, GA, USA, Apr. 2008, pp. 243–254.

[3] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh, "Usher: Improving data quality with dynamic forms," in *Proc. ICDE*, Long Beach, CA, USA, Mar. 2010, pp. 321–332.

[4] E. Chu, A. Baid, X. Chai, A. Doan, and J. F. Naughton, "Combining keyword search and forms for ad hoc querying of databases," in *Proc. ACM SIGMOD*, Providence, RI, USA, Jun. 2009, pp. 349–360.

[5] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Query recommendations for interactive database exploration," in *Proc. SSDBM*, New Orleans, LA, USA, Jun. 2009, pp. 3–18.

[6] M. Jayapandian and H. V. Jagadish, "Automated creation of aforms-based database query interface," *Proc. VLDB*, vol. 1, no. 1, pp. 695–709, Aug. 2008.

[7] M. Jayapandian and H. V. Jagadish, "Expressive query specification through form customization," in *Proc. Int. Conf. EDBT*, Nantes, France, Mar. 2008, pp. 416–427.

[8] M. Jayapandian and H. V. Jagadish, "Automating the design and construction of query forms," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 10, pp. 1389–1402, Oct. 2009.

