Matrix Factorization Based Query Recommendation

Visak Paul[1], **Sreena Sreedhar**[2]

[1]M.Tech Student, Department of Computer Science and Engineering, Ilahia College of Engineering and Technology, Muvattupuzha, Kerala, India

[2]Assistant Professor, Department of Information Technology, Ilahia College of Engineering and Technology, Muvattupuzha, Kerala, India

Abstract: ***Database exploration is always a tedious task for the person who lacks skill in writing complex SQL queries. In order to aid such people, SQL recommendations are provided with the help of an interactive query recommendation system. The recommendations will be based on the current query, queries previously submitted by the user and the queries submitted by other users to the system. Based on this, the recommendation engine recommends the recommendation query to the user. The user can use this query as a template to formulate the query he wanted or he can submit the same. The recommended query will be like the query the user may want to write. The recommendation users the general concept of collaborative filtering method in which the recommendations will be based on the relationships between the queries submitted and the interests of the user. The use matrix factorization further improves the recommendation accuracy and thereby a better result for the user.***

Keywords: recommender systems, matrix factorization, query recommendation, collaborative filtering

1. Introduction

Query Recommendation aims at suggesting sql queries to the users who lack expertise in formulating sql queries. For performing recommendations, a sql query recommendation system is developed. The recommendation system continuously monitors the users behavior of querying from the database and make recommendations based on this.

Sql query recommender system's concept is based on a the concept on web recommender systems. If users A and B have posed similar queries, then the other queries of B may be of interest to user A and vice versa. In other words, recommending the queries of user B in order to help user A in their exploration of the database. In particular, to implement this idea through Collaborative Filtering, a well known, mature technique that has been used in Web recommender systems. However, the transfer of this approach to the database context introduces several technical challenges. First, SQL is a declarative language, and hence syntactically different queries may reflect the same information need. The recommended queries are relevant to the user's information needs and can be submitted directly or be further refined. In other words, the user can use them as "templates" for query formulation instead of having to compose new ones

Recommender system addresses these challenges by employing a closed-loop approach. Specifically, the recommender system framework decomposes each query into Basic elements that capture the essence of the query's logic. These elements are used to compute similarities between users, as well as a signature of the user's querying behavior (and, to some extent of the user's information needs). Recommendations are generated by mining queries from the system log that match well with the signature. Hence, the user is presented with queries that match her querying behavior, and are likely to be more intuitive than purely synthetic ones.

2. System Details

2.1 Recommender Systems

Recommender systems have become a vital tool for attracting and keeping users on commercial websites. Their utility is supported by research as well as common practice. The task of a recommender system can be abstractly described as follows. Consider a matrix in which rows correspond to users and columns correspond to items. Each value in this matrix represents a user's revealed or stated preference (if any) for an item: for example, whether he purchased a book, how many times he listened to a song, or what rating he gave to a movie. Because the item set is typically far larger than a single user can consume and evaluate, this matrix is "sparse:" only a small fraction of entries are filled in. A recommender system takes this matrix as input, along with any available metadata about users (such as demographics) and items (such as item categories). The goal of the system is to extrapolate users "true" preferences over the full item set.

Recommender systems can be classified as content-based, collaborative and hybrid. Content-based systems identify relationships between items based on metadata alone and recommend items which are similar to the user's past transactions. Collaborative filtering identifies relationships between items based on the preferences of all users. Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a person chosen randomly.

2.2 Recommender Framework

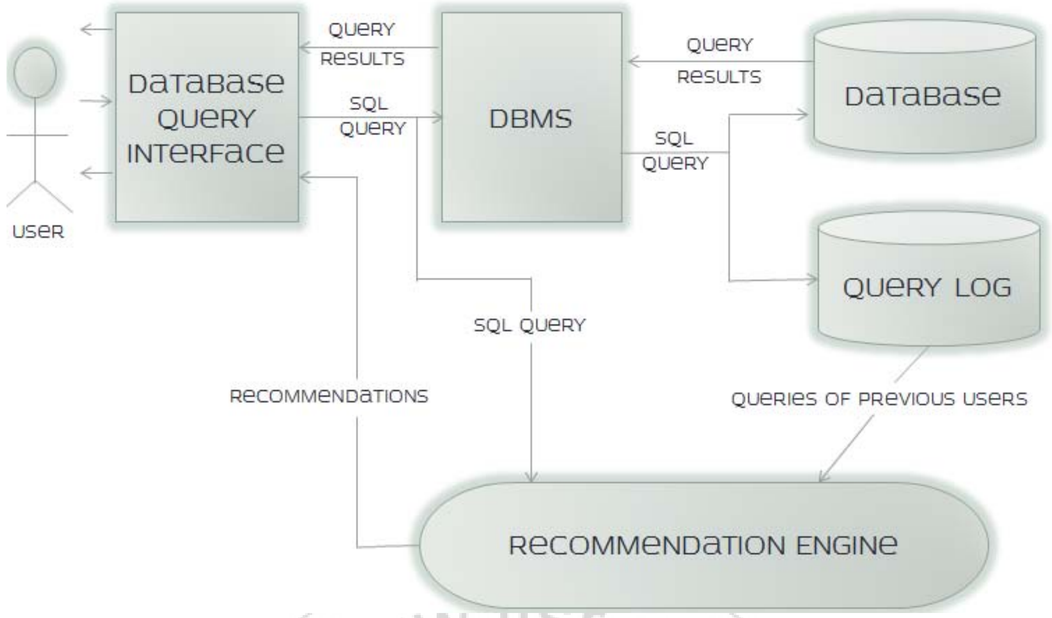


Figure 1: Framework workflow

The abstract framework is essentially a workflow, as depicted in Figure 1. The active user's queries are forwarded to both the DBMS and the Recommendation Engine. The DBMS processes each query and returns a set of results. At the same time, the query is stored in the Query Log. The Recommendation Engine combines the current user's input with information gathered from the database interactions of past users, as recorded in the Query Log, and generates a set of query recommendations that are returned to the user. Consider a setting where users explore a relational database through a sequence of SQL queries. The goal of the exploration is to discover interesting information or verify a particular hypothesis. The queries are formulated based on this goal and reflect the user's overall information need. As a consequence, the queries posted by a user during one "visit" (commonly called session) to the database are typically correlated, in that the user formulates the next query in the sequence after having inspected the results of previous queries.

Let assume that each user has a single session with the database. This assumption can be lifted in a straightforward manner at the expense of more complicated notation. Given a user *i*, let Q_i denote the set of SQL queries that the user has posed so far in a single session. Introduced the notion of a session summary to summarize the characteristics of the queries posed in the session. This summary captures the parts of the database accessed by the user and incorporates a metric of importance for each part. Contrary to Web recommender systems, where the users are represented by the items they visit/rate/purchase, in the context of relational databases, several ways to model the session summaries exist. For instance, a crude summary may contain the names of the relations that appear in the queries of the user, and the importance of each relation can be measured as the number of queries that reference it. On the other extreme, a detailed summary may contain the actual results inspected by the user, along with an explicit rating of each result tuple. In what follows, use S_i to represent the session summary for user *i*. User $i = 0$ will always represent the current user (for whom recommendations are generated), whereas $i = 1,\dots,n$ represents past users of the system. In a slight abuse of notation, use S_i to represent both the session summary and user *i*. To generate recommendations for current user S_0 , the framework first computes a "predicted" summary S^{pred} . This summary captures the predicted degree of interest of S_0 with respect to different query characteristics, including those that already appear in his/her queries, as well as new ones that have not been used yet.

Overall, the framework consists of the following components: (a) a model for session summaries, (b) a method to compute the session summaries $S_0 ,\dots,S_n$, (c) a method to compute S^{pred}, and (d) a method to select queries based on S^{pred}.

2.3 Tuple-Based Query Recommendations

In this instantiation of the framework [2], the session summary S_i is represented as a weighted vector, where every coordinate corresponds to a distinct database tuple. Assume that the total number of tuples in the database, and as a consequence the length of the vector, is T. The weight $S_i[\tau]$ represents the importance of a given tuple $\tau \in T$ in session S_i, and is non-zero only if τ is a witness for at least one query in the session. The intuition is that S_i captures the tuples in the base tables that are touched by the queries in the user's session. Hence, sessions that contain equivalent queries will map to the same summary.

2.4 Fragment-Based Query Recommendations

The fragment-based instantiation of the recommender system framework works in a similar manner to the tuple-based one [1]. The two main differences lie in the representation of the session summaries and the formulation of similarities. More specifically, the coordinates of the session summaries correspond to fragments of queries instead of witnesses.

Identify as fragments the following syntactical features of the queries in the session: attribute references, table's references, join and selection predicates. At a high level, the idea behind this approach is to recommend queries whose syntactical features match the queries of the current user.

3. Proposed System

3.1 Matrix Factorization Model

Matrix factorization models [4] map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Accordingly, each item *i* is associated with a vector $q_i \in R^f$, and each user u is associated with a vector $p \in R^f$. For a given item *i*, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user *u*, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors, again, positive or negative. The resulting dot product, $q_i^T p_u$ captures the interaction between user *u* and item *i,* the user's overall interest in the item's characteristics. This approximates user *u*'s rating of item *i*, which is denoted by r_{ui}, leading to the estimate $r = q_i^T p_u$. The major challenge is computing the mapping of each item and user to factor vectors $q_i, p_i \in R^f$. After the recommender system completes this mapping, it can easily estimate the rating a user will give to any item by using equation. Such a model is closely related to singular value decomposition (SVD), a well-established technique for identifying latent semantic factors in information retrieval. Applying SVD in the collaborative filtering domain requires factoring the user-item rating matrix. This often raises difficulties due to the high portion of missing values caused by sparseness in the user-item ratings matrix. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier systems relied on imputation to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, inaccurate imputation might distort the data considerably. Hence, more recent works suggested modeling directly the observed ratings only, while avoiding overfitting through a regularized model. To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings parameters, whose magnitudes are penalized.

4. Proposed System

The fragment-based approach clearly captures information at a coarser level of detail, and hence it is expected to miss interesting correlations between users. For instance, two distinct selection predicates will be mapped to different fragments even if they are satisfied by the same tuples in the base tables. It is therefore expected that the basic tuple-based approach yields better results in terms of precision. This, however, comes with a cost; the tuple-based approach constructs large (and relatively dense) summaries and, most importantly, requires real-time calculations of the similarities between the session summary S_0 of the current user and these of past users. On the other hand, the big advantage of the fragment-based approach is that it can be implemented very efficiently; the space of fragments grows slowly allowing for a scalable system, the summaries are very sparse enabling faster similarity calculations and, most importantly, the fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated, leveraging all the advantages of item-to-item collaborative filtering. A comparable response time is achieved when the tuple-based instantiation employs MinHash synopses.

4.1 SQL Query Preprocessing

Because of the large and excessive amount of slightly dissimilar queries existing in the query logs, in order to relax them to increase their cardinality, and thus the probability of finding similarities between different user sessions, query preprocessing must done. The intuition is that if two users query the same table and attributes, using slightly different filtering conditions, the algorithm should consider them as similar. In essence, all the WHERE clauses are relaxed by converting the numerical data and string literals to generic string representations. For example, all strings are replaced by STR, all hexadecimal numbers by HEXNUM and all decimals by NUM. A similar generalization is also followed for lists or ranges of numbers and strings. The mathematical and set comparators are also replaced by string equivalents, for example "=" is replaced by EQU and "≤" by COMPARE. Each distinct fragment is assigned a numerical identifier, used in the query and session vector representation. For each new fragment not previously recorded in the query log, recommender system generates a new identifier. Such updates occur in real-time, as the current user posts a query including new fragments. In the case of the WHERE clause, only the joins and the filter conditions are stored. Because of the generalization, the fragments in the WHERE clause are not differentiated based on their actual values, but rather based on the attributes used for filtering.

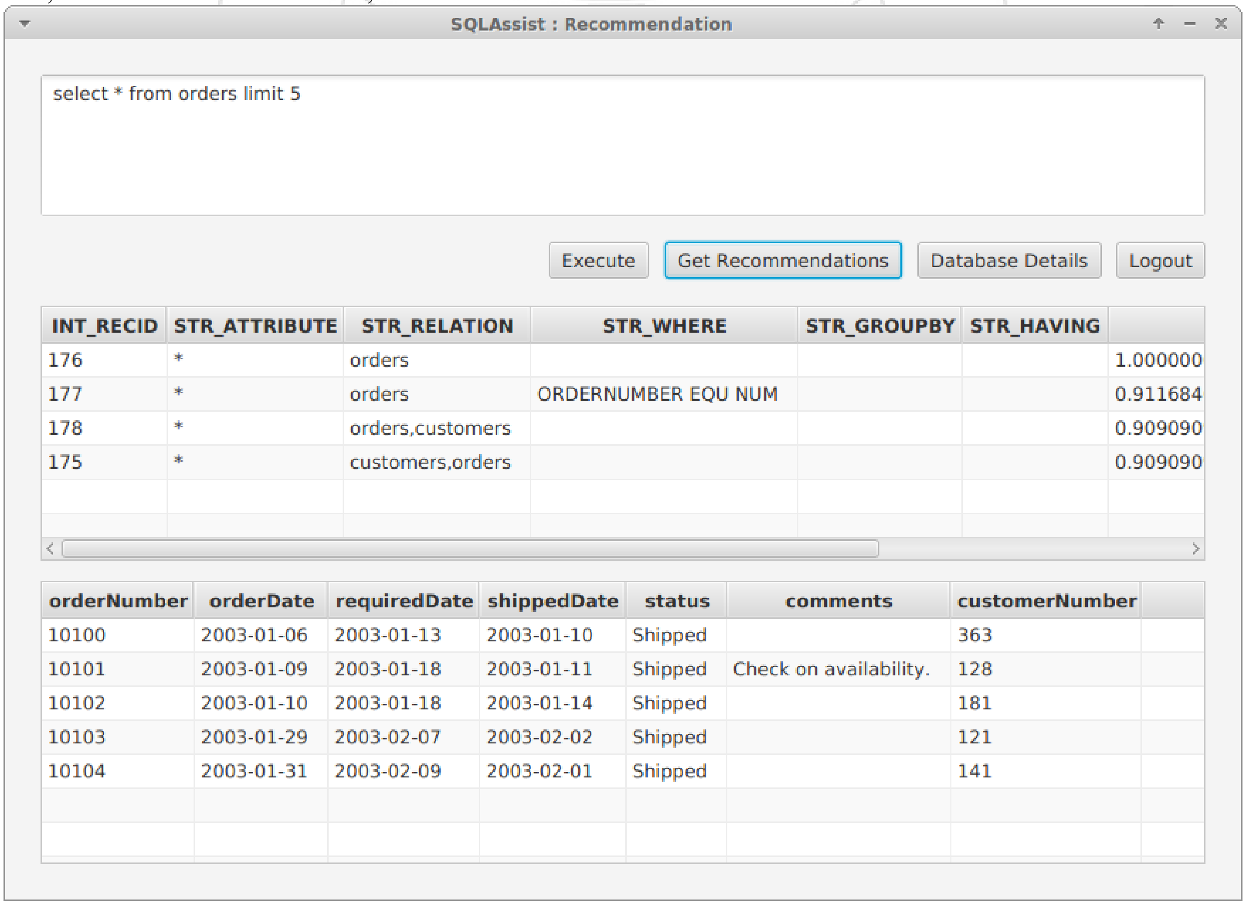
4.2 Recommender System Prototype

A prototype of the proposed recommender system is implemented that supports the recommendation engines using Java and runs on top of a standard relational DBMS(MySQL) to store the query logs and the similarities. The database query interface module is built using JavaFX. The recommendation engine module is also built using Java. Once a user logs in the system, he is able to access the database. The user can author and submit a SQL query. Recommender system sends the request to the database, and presents the user with the results. At the same time, the system records the active user's queries, creating an implicit user profile. This user profile is used as input to the algorithm, along with the predictive model to generate real-time, personalized query recommendations. For each recommended query, the user is able to examine a sample of the results that will be retrieved, in order to decide whether it addresses her needs, prior to actually submitting it to the DBMS. Recommender system continuously monitors the user's querying behavior and finds matching patterns in the system's query log, in an attempt to identify previous users with similar information needs. Subsequently, recommender system uses these "similar" users and their queries to recommend queries that the current user may find interesting. In this recommender system framework, the active user's session is represented by a set of query fragments. The active user's queries are forwarded to both the DBMS and the Recommendation Engine. The DBMS processes each query and returns a set of results. At the same time, the query is stored in the Query Log. The Recommendation Engine combines the current user's input with information gathered from the database interactions of past users, as recorded in the Query Log, and generates a set of query recommendations that are returned to the user. The recorded fragments are used to identify similar query fragments in the previously recorded sessions, which are in turn assembled in potentially interesting queries for the active user.

At all times, the active user is able to: (a) formulate a query from scratch, (b) select a recommended query and submit it as it is, or (c) select a recommended query and edit it before submitting it to the database. Moreover, the interface allows the user to browse the database schema, review and re-submit queries that were posed during his recent history.

4.3 Singular Value Decomposition

In a collaborative filtering problem, the connections that do not exist (user *i* has not rated item *j*, person *x* has not friended person *y*) are generally treated as missing values to be predicted, rather than as zeros. That is, if user *i* hasn't rated item *j*, we want to guess what he might rate it if he had rated it. If person *x* hasn't friended *y*, we want to guess how likely it is that he'd want to friend him. The recommendations are based on the reconstructed values. When taking the SVD of the social graph (e.g., plug it through svd()), basically we are inputing zeros in all those missing spots. That this is problematic is more obvious in the user-item-rating setup for collaborative filtering. If we had a way to reliably fill in the missing entries, we wouldn't need to use SVD at all. The recommendations are given based on the filled in entries. If we don't have a way to do that, then we shouldn't fill them before we do the SVD. Let the matrix A be such that rows are the users and the columns are the items that the user likes.. One way to think of SVD is as follows: SVD finds a hidden feature space where the users and items they like have feature vectors that are closely aligned. So, when compute $A = U \times s \times V$, the U matrix represents the feature vectors, corresponding to the users in the hidden feature space and the V matrix represents the feature vectors corresponding to the items in the hidden feature space. Now, if two vectors from the same feature space and ask to find if they are similar, the simplest thing that we can do for accomplishing that is Dot product. So, if we want to see user *i* likes item *j*, all need to do is take the dot product of the *i*th entry in U and *j*th entry in V. Of course, dot product is by no means the only thing to apply, we can use any similarity measure that is applicable. Figure 2 shows the recommender system interface.



```
select * from orders limit 5
```

INT_RECID	STR_ATTRIBUTE	STR_RELATION	STR_WHERE	STR_GROUPBY	STR_HAVING	
176	*	orders				1.000000
177	*	orders	ORDERNUMBER EQU NUM			0.911684
178	*	orders,customers				0.909090
175	*	customers,orders				0.909090

orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber
10100	2003-01-06	2003-01-13	2003-01-10	Shipped		363
10101	2003-01-09	2003-01-18	2003-01-11	Shipped	Check on availability.	128
10102	2003-01-10	2003-01-18	2003-01-14	Shipped		181
10103	2003-01-29	2003-02-07	2003-02-02	Shipped		121
10104	2003-01-31	2003-02-09	2003-02-01	Shipped		141

Figure 2: Recommender System Interface

5. Conclusion

In this paper, we proposed a recommender system framework that aims to generate useful SQL query recommendations to users of relational databases. By using our recommender system, users can generate a set of recommendations for the query he submitted and he can select a suitable sq query from the recommendations. This reduces the time to formulate such complex SQL queries and also helps the less experienced users. The use of matrix factorization based recommender system improves the recommendation.

References

[1] Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, Naushin Shaikh, "QueRIE: Collaborative Database Exploration", IEEE Trans. Knowl. Data Eng. 26(7): 1778-1790,2014
[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Collaborative filtering for interactive database exploration," in Proc. of the 21st Intl. Conf. on Scientific and Statistical Database Management (SSDBM'09),2009.
[3] Shameem Ahamed Puthiya Parambath, "Matrix Factorization Methods for Recommender Systems", Umea University, Sweden, 2013
[4] Yehuda Koren, Robert M. Bell, Chris Volinsky, "Matrix Factorization Techniques for Recommender Systems." IEEE Computer, 42(8): 30-37, 2009
[5] Noam Koenigstein, Parikshit Ram, Yuval Shavitt, "Efficient Retrieval of Recommendations in a Matrix Factorization Framework", CIKM'12, Proceedings of the21st ACM international conference on information and knowledge management, 535-544
[6] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mit-tal, D. On, N.Polyzotis, and J. S. V. Varman, "SQL QueRIE Recommendations," in Proc. of the 36th Intl. Conf. on Very Large DataBases (VLDB 2010), 2010.

Author Profile

Visak Paul received the Bachelor of Technology degree in Computer Science and Engineering from Mahatma Gandhi University, Kerala in 2012. He is currently doing Master of Technology degree in Computer Science and Engineering with specialization in Information Systems from Mahatma Gandhi University, Kerala.

Sreena Sreedhar is an Assistant Professor at the Information Technology Department, Ilahia College of Engineering and Technology, Muvattupuzha, Kerala.