Predicting Your Next OLAP Query Based on Recent Analytical Sessions

Marie-Aude Aufaure[1], Nicolas Kuchmann-Beauger[1], Patrick Marcel[2], Stefano Rizzi[3], and Yves Vanrompay[1]

[1] MAS Laboratory, École Centrale Paris, France
[2] Université Francois Rabelais de Tours, France
[3] DISI, University of Bologna, Italy

Abstract. In Business Intelligence systems, users interact with data warehouses by formulating OLAP queries aimed at exploring multidimensional data cubes. Being able to predict the most likely next queries would provide a way to recommend interesting queries to users on the one hand, and could improve the efficiency of OLAP sessions on the other. In particular, query recommendation would proactively guide users in data exploration and improve the quality of their interactive experience. In this paper, we propose a framework to predict the most likely next query and recommend this to the user. Our framework relies on a probabilistic user behavior model built by analyzing previous OLAP sessions and exploiting a query similarity metric. To gain insight in the recommendation precision and on what parameters it depends, we evaluate our approach using different quality assessments.

Keywords: OLAP, Query recommendation, User modeling.

1 Introduction

Online Analytical Processing (OLAP) systems allow users to explore and analyze large volumes of data by formulating queries on multidimensional cubes. Despite the flexibility, usability, and efficiency of modern OLAP systems, the huge number of possible aggregations and selections that can be operated on multidimensional data may make the user experience disorientating and frustrating, so that users may need a long time to achieve their analysis goals.

The approach we propose in this paper to tackle this problem is based on a prediction of the most likely queries the user will submit next. We start by observing that OLAP workloads tend to show different patterns depending on the specific analytical tasks the user is performing. Our goal is to learn these patterns and represent them in a probabilistic model of the user's behavior; to this end, we analyze the query logs of a user, we cluster queries using a similarity metric, and we derive a Markov-based model of the user behavior. Using this model, we are able to predict the most likely queries the user will formulate next given the current query. Our approach takes a step towards improving the quality of the user experience with OLAP systems in different ways. First, given the current query in the OLAP session, a set of most probable next queries can be proactively recommended to the user to guide her in analyzing the data and prevent her from "getting lost" among multidimensional data. Secondly, query recommendation could allow the average length of OLAP sessions to be reduced because users are driven towards their analysis goal and can reach them in less steps. Indeed, if a user is given a choice between different future queries, she might be able to skip some steps and more quickly arrive at the expected results. Thirdly, the OLAP cache manager can exploit the predicted queries for estimating the benefits of a cached object for future queries. Using predictions, the cache manager can also prefetch objects that are likely to be of interest for future queries, which results in a reduction in latency time perceived by the user.

The rest of the paper is organized as follows. Section 2 gives an overview of related work in OLAP query prediction and recommendation. Section 3 describes formally the query model we incorporate in our approach. Then, Section 4 presents the different steps for recommendation, i.e., query clustering and query prediction based on a user behavior model, together with the query similarity metric we adopt. In Section 5 we experimentally evaluate our approach, while in Section 6 we conclude and give directions for future research.

2 Related Work

Recommending items of interest and queries has been intensively researched in the domains of Information Retrieval [1] and search engines [2]. Recently, in the database community, there has been an increasing interest in leveraging past queries or query answers to assist interactive relational database exploration [3–9]. The approaches proposed include past query browsing and/or searching [7], query completion [6] and query recommendation [5, 8, 9]. Noticeably, automatic query recommendation approaches either rely on the query answer and database instance, which may lead to efficiency problem, or treat sessions as sets of queries, overlooking the intrinsic sequential nature of the exploratory process. A framework for recommending OLAP queries has been presented in [10], whose authors group queries according to a finer similarity measure and then recommend queries by matching logged sessions to the current session. To the best of our knowledge, PROMISE is the first system applying predictive caching to multidimensional queries aimed at reducing the execution time of OLAP queries within a session [11]. PROMISE integrates a Markov model [12] where each state corresponds to a discrete point of the timescale (i.e., only one state is active at time t); a transition between two states corresponds to the probability to reach the next state given the previously visited states. While [10] bases its recommendations on a similarity metric and does not use a probabilistic model, PROMISE probabilistically represents next queries, mainly for query prefetching and not for recommendation. Besides, PROMISE use a coarse approach to group queries (grouping by similar group by set, measure set and slicer). Our system tries to combine both approaches by providing a probabilistic approach using Markov models where the states are clusters of queries, grouped according to a finer similarity measure. Prediction models like the one used in PROMISE predict multidimensional queries based either on already visited queries (from query logs) or on resources provided by experts during a conceptual modeling process [11]. Sarawagi's work [13, 14] is a different, somewhat orthogonal approach, in the sense that the goal is not to model the user's querying behavior. Instead, users are led to the "most surprising" unvisited parts of the cube, whatever their past behavior was. Finally, Sarawagi leverages the query answers and the cube instance, while the present work requires to know only the query expression.

3 Query Model

We consider in the following a multidimensional schema $\mathcal{M} =< L, H, M >$ as defined in [15], where L is a finite set of levels, H a finite set of hierarchies (each including a subset of levels), and M a finite set of measures. We work with a basic form of OLAP query centered on a single multidimensional schema and characterized by an aggregation and a selection expressed through a conjunctive predicate. To be independent of the details related to logical design of multidimensional schemata and to specific query plans, we express queries using an abstract syntax. An OLAP query on schema $\mathcal{M}$ is thus defined as a triple $q =< g, P, Meas >$ where g is the query group-by set (including one level for each hierarchy in H), $P = \{c_1, \ldots, c_n\}$ is a set of Boolean clauses, one for each hierarchy, whose conjunction defines the selection predicate for the query, and $Meas \subset M$ is the measure set whose values are returned by the query. IPUMS is a public database storing census microdata for social and economic research (Minnesota Population Center, 2008). Its CENSUS multidimensional schema has five hierarchies as shown in figure 1, namely *RACE*, *TIME*, *SEX*, *OCCUPATION*, and *RESIDENCE*, and measures *AvgIncome*, *AvgCostGas*, *AvgCostWtr*, and *AvgCostElect*.

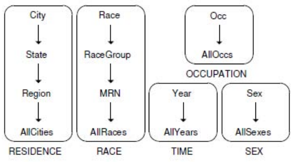

Fig. 1. Roll-up orders for the hierarchies in the CENSUS schema

Then, a query expressed as "Income per occupations and state in 2013" would have the following abstract representation:

$$q = \begin{bmatrix} < \mathsf{State}, \mathsf{AllRaces}, \mathsf{Year}, \mathsf{Occ}, \mathsf{AllSexes} > \\ \{\mathrm{TRUE}_{\mathsf{RESIDENCE}}, \mathrm{TRUE}_{\mathsf{RACE}}, (\mathsf{Year} = 2013), \mathrm{TRUE}_{\mathsf{OCCUPATION}}, \mathrm{TRUE}_{\mathsf{SEX}}\} \\ \{\mathsf{AvgIncome}\} \end{bmatrix}$$

We give the MDX formulation of the query for illustration purposes below. MDX is a de-facto standard for querying multidimensional databases. Some of its distinguishing features compared to SQL are the possibility of returning query results that contain tuples with different aggregation levels and the possibility of specifying how the results should be visually arranged into a multidimensional representation.

```
SELECT
    NON EMPTY {[Measures].[AvgIncome]} ON COLUMNS,
    NON EMPTY Hierarchize(Crossjoin({  [Residence].[State],
                                       [Race].[AllRaces],
                                       [Occupation].[Occ],
                                       [Sex].[AllSexes] }))
    DIMENSION PROPERTIES
    PARENT_UNIQUE_NAME ON ROWS
FROM ( SELECT {[Time].[Year].&[2013]}
         ON COLUMNS FROM [CENSUS])
```

4 Query Prediction for Recommendation

This section presents our approach for query prediction for recommendation. First we give an overview of the proposed architecture and we outline the query similarity metric we adopt, then we explain the clustering and prediction steps needed for recommendation. A sketch of the functional architecture we propose is shown in Figure 2. After the user has formulated a query, the query processing component is in charge of processing it and getting the results back from the data warehouse. Each query issued by the user in a session will also be stored in the query log. Based on the information available in the query log, the clustering & learning module is responsible for dynamically determining the user behaviour model (learning step) from the recognized clusters representing similar queries (clustering). The user's behaviour model, learned and updated by the clustering and learning module, will then be used by the prediction module. It should be noted that clustering and learning the user behavior model are done at regular times offline, and then the model is consulted at runtime to predict the next user query. Finally, the prediction module, guided by the user's current query, is based on the results from the discovery process previously stored in the past history of user queries. From this data, the prediction module is able to recommend the

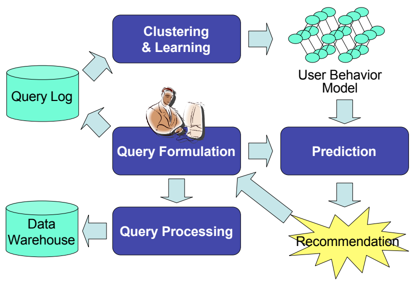

Fig. 2. Functional architecture for query recommendation

next query. From the user's behaviour model inferred by the learning module, the prediction module will determine the user's most likely future query. In order to recommend queries to users, we build a behaviour model that predicts the most likely next queries. The approach consists of 2 steps, the first being the clustering of similar queries based on a query similarity metric. In the second step we treat these clusters as states of a Markov chain model and compute the probability of the most likely next state (i.e. cluster of queries). In the clustering process, similar queries are grouped into clusters, as a way to reduce the size of the history log and to be able to recommend similar queries. These clusters are interpreted as states of a state machine, and the transition probabilities from one state to another are calculated based on the history. The interpretation of the current OLAP session as a trajectory of states allows to anticipate most probable future states. In our approach, this process consists of estimating the probabilities of moving from one state to other possible future states. The query finally recommended is then the one that is most similar to the current query and that is a member of the most likely next state, given the state the current query belongs to. The state the current query belongs to is the cluster whose members are on average most similar to the current query, again computed with the query similarity metric. So the recommended query is the most similar query in the most likely next state.

4.1 Query Similarity Metric

As shown in the previous section, our approach relies on a metric to compute the similarity between OLAP queries both for clustering and predicting. To be used in our context, a similarity metric must meet two requirements: First, efficiency is required because, during the prediction step, the current query q has to be matched to the closest state in the Markov model, so the similarity between the current query and all the states in the model must be computed. On the other hand, multidimensional databases store huge volumes of data, and as a consequence OLAP queries can return large result sets. Extensional computation of query similarity (i.e., made by comparing query results like in [16] and [17]) can thus pose serious efficiency problems. For this reason we use a metric that computes query similarity at the intensional level, i.e., by only looking at the query expressions. Second, the space of possible queries on a multidimensional schema is large, there is little probability that two queries are really identical. So, the metric we adopt should return a score and not a Boolean value like in [18].

We adopt as a metric for computing the similarity between two queries q and q' the one defined as σ_{que} in [15]:

$$\sigma_{\text{que}}(q,q') = \alpha \cdot \sigma_{\text{sel}}(q,q') + \beta \cdot \sigma_{\text{meas}}(q,q') + \gamma \cdot \sigma_{\text{gbs}}(q,q') \tag{1}$$

where σ_{sel}, σ_{meas}, and σ_{gbs} represent respectively the *selection*, *measure* and *group-by set* similarities as defined in [15], and $\alpha, \beta, \gamma \in [0,1]$ are parameters to be experimentally determined. A priori, the three terms in the equation are not equally important. Based on a set of tests made with users, in [15] it is argued that the selection predicate is the most important in determining similarity between OLAP queries, followed by the group-by set; the least significant term is the set of measures to be returned.

As an example we take two queries on the CENSUS schema specified as follows:

$$q_1 = \begin{bmatrix} < \mathsf{State, RaceGroup, Year, Occ, AllSexes} > \\ \{\mathrm{TRUE}_{\mathsf{RESIDENCE}}, \mathrm{TRUE}_{\mathsf{RACE}}, (\mathsf{Year} = 2005), \mathrm{TRUE}_{\mathsf{OCCUPATION}}, \mathrm{TRUE}_{\mathsf{SEX}}\} \\ \{\mathsf{AvgCostWtr,\ AvgCostElect}\} \end{bmatrix}$$

$$q_2 = \begin{bmatrix} < \mathsf{State, RaceGroup, Year, Occ, AllSexes} > \\ \{\mathrm{TRUE}_{\mathsf{RESIDENCE}}, \mathrm{RaceGroup = Chinese}, (\mathsf{Year} = 2005), \mathrm{TRUE}_{\mathsf{OCCUPATION}}, \mathrm{TRUE}_{\mathsf{SEX}}\} \\ \{\mathsf{AvgCostWtr,\ AvgCostElect}\} \end{bmatrix}$$

For these queries, only differing in selection predicate, $\sigma_{\mathrm{que}}(q1, q2) = 0.95$, taking for simplicity equal weights for α, β and γ.

4.2 Clustering

The first step of our approach is the clustering of user's queries. During different analysis sessions, a user often expresses similar (but not identical) queries; for instance, she may formulate queries with the same group-by set but on different slices of data (which means, with different selection predicates). Indeed, as the query log contains a trace of all queries formulated by each user, it is very likely that some of them will be similar. The clustering algorithm is inspired by standard density-based clustering methods (k-means with dynamic number of clusters), but we define the clustering space as being build using the similarity metric introduced in the previous section. The input to this step are the abstract representations of the queries previously issued by the user, stored in the log. The goal is to determine query clusters in such a way that the queries in the same cluster are similar (in the sense of Section 4.2) to each other, and that queries in different clusters are not similar to each other. We start with randomly selecting a number of queries from the query logs, which will serve as the seeds for the clusters (Listing 1.1 line 3). This random selection of queries ensures that the clustering mechanism takes into account the density of queries over the query space introduced by the query similarity metric. Then, we assign each query q in the log (Listing 1.1 line 4) to the cluster whose queries on average have highest similarity with q (Listing 1.1 lines 6-8). To avoid that clusters grow unlimited in size, a cluster split rate is defined; when a cluster reaches a given number of member queries, the cluster is split in two (Listing 1.1 line 9).

```
1. int cluster_split_rate = t;
2. List<Query> queries = read_in(ipums_log.txt);
3. List<Cluster> clusters = select_random_seeds(nbOfSeeds);
4. for each Query q in queries{
5.   for each Cluster c in clusters{
6.      compute_average_similarity(q,c);}
7.      Cluster selected_c =
8.            assign_query_to_closest_cluster(clusters,q);
9.      if (selected_c.size()>t) selected_c.split();}
```

Listing 1.1. Clustering similar queries

4.3 Learning the User Behavior Model

We model the querying behavior of each user in the form of a Markov chain, where each query cluster (determined as explained in Section 4.3) is a state. The series of states satisfies the Markov property, i.e., the probability of reaching a state in the future, given the current and past states, is the same probability as that given only the current state. This means that past states give no information about future states. More precisely, if the system is in state x at time n, the probability that it moves to state y at time $n+1$ depends only on the current state x and not on past states. The transition probability distribution can then be represented as a matrix P, called a *transition matrix*, whose (i,j)-th element is defined as follows:

$$P_{ij} = Pr(X_{n+1} = j | X_n = i)$$

The initial probability $Pr(X_{n+1} = j | X_n = i)$ is $\frac{1}{m}$, where m is the number of states that can follow the current state. In our case, each state corresponds to a cluster of queries, giving as value for m the initial number of clusters that were identified in the clustering step.

The probability $Pr(X_{n+1} = j | X_n = i)$ could be updated by counting how often query q_j is preceded by query q_i and dividing this number by the total number of queries that were observed as following query q_i. This means however that the past is as important as the present. In the OLAP context, the series of queries a user performs will typically evolve over time. So, if the log includes for instance the queries performed by the user during the last six months, it is reasonable to have more recent queries having relatively more influence on the user behavior model than older ones. To this end, the transition probability function should be updated in such a way that recent transitions have more relevance than older ones, which we do using an exponential smoothing method:

$$P_{ij} = \rho \times x_j + (1 - \rho) P'_{ij}$$

where P'_{ij} represents the old probability and $x_j \in \{0, 1\}$ is the value for the choice taken at query q_i with respect to query q_j. If $x_j = 1$ then q_j was executed after q_i, if $x_j = 0$ it was not. Using this method, the sum of all outgoing probabilities remains 1, as required for a transition probability matrix. The *learning rate* $\rho \in [0, 1]$ is a real number that controls how important recent observations are compared to history. If ρ is high, the present is far more important than history; in this setting, the system will adapt quickly to the behavior of the user, which can be necessary in a rapidly changing environment or when the system is deployed and starts to learn. In a rather static environment, ρ can be set low. In conclusion, by incorporating the learning rate, we make sure the user behavior model is dynamic and evolves together with changing habits or preferences of users.

The algorithm for learning the user behavior model is shown below. Based on the clusters of queries that were obtained in the clustering step (Listing 1.2 line 1) and on the query log (Listing 1.2 line 2), a transition probability matrix (Listing 1.2 line 3) is constructed expressing the probability of going from each cluster to each other cluster. For this, each query q_i and q_{i+1} in the log is considered (Listing 1.2 line 4). We check to which clusters q_i and q_{i+1} belong to (Listing 1.2 line 5-6) and update the probabilities in the transition probability matrix accordingly (Listing 1.2 line 7).

```
1. List<Cluster> states = get_query_clusters();
2. List<Query> queries = read_in(ipums_log.txt);
3. float[][] transition_matrix = new float[nb][nb];
4. for each Query q(i) and q(i+1) in queries
5.   Cluster p = assign_query_to_cluster(q(i), states);
6.   Cluster q = assign_query_to_cluster(q(i+1), states);
7.   update_probs (transition_matrix, p, q);
```

Listing 1.2. Learning the user behavior model

Once the user behavior model has been constructed, it can be exploited to recommend a next query to the user. Given the current user query, we identify to which state (i.e. cluster of queries) in the Markov model this query is closest to by computing the average similarity between the current query and each state. This similarity is equal to the average of the similarity between the current user query and the queries contained in the state. Since the similarity metric we use computes its value intensionally and not extensionally, this is computationally feasible. Once we have identified the appropriate state, the Markov model gives us the most probable next state by looking in the transition probability matrix for the highest probability on the row corresponding to the matching state. In this most probable next state, we retrieve the query that is most similar to the current query, and propose this one as the query predicted for recommendation. For example, given the current user query is query q_2 introduced in section 4.2, we identify the most probable next cluster from the transition probability matrix. This most likely next cluster turns out to have probability 0.36, which value gives us good confidence in the correctness of the prediction (as will be discussed in the next section). In this cluster, we select the query q_3 most similar to q_2 for recommendation, which is shown below. Query q_3 differs from q_2 by adding the *SumCostWtr* predicate.

$$q_3 = \begin{bmatrix} < \mathsf{State, RaceGroup, Year, Occ, AllSexes} > \\ \{\mathrm{TRUE}_{\mathsf{RESIDENCE}}, \mathrm{RaceGroup} = \mathrm{Chinese}, (\mathsf{Year} = 2005), \mathrm{TRUE}_{\mathsf{OCCUPATION}}, \mathrm{TRUE}_{\mathsf{SEX}}\} \\ \{\mathsf{AvgCostWtr,\ AvgCostElect,\ SumCostWtr}\} \end{bmatrix}$$

5 Experiments and Evaluation

We evaluated our approach using a synthetic dataset of log traces of MDX queries, with the goal of answering the following questions: 1) How does the error rate evolve and is influenced by the cluster split rate (i.e. the average size of clusters). 2) How much does the error rate improve when using a threshold for the prediction probability P in considering to recommend.

To the best of our knowledge, available public dataset logs like the one used by [19] do not correspond to an OLAP query log in the sense that they do not have a multidimensional schema, and the SQL queries in these logs cannot be seen as OLAP queries. In particular, the vast majority of them does not have grouping and aggregation, and therefore do not fit our OLAP query model. Therefore, we opted for generating synthetic datasets for experimentation. The synthetic dataset consists of query session logs that were generated considering a specific policy, using the IPUMS CENSUS multidimensional schema. In a first step, random queries were generated and grouped together according to their similarity. This allows for a broad coverage of the space of possibe queries since there is a distance between the groups. Each group corresponds thus to one type of session. In a second step, we generate a number of sessions for each group, queries in the group acting as seeds. For this we select at random a query A and a query B in a the group, A being the start query of the session and B being the last query of the session. The shortest OLAP path (series of OLAP operations) between A and B is calculated and each OLAP operation is translated into one OLAP query, as such generating the session. While the specific queries contained in the path from A to B are fixed, variance was introduced in the order of the queries to obtain more realistic sessions. We used 75% of this dataset for training purposes to build the user behavior model, and 25% as a testing set to perform the evaluation.

In order to gain insight in the performance of the query prediction in terms of correctness, we propose in this section a set of metrics. The evaluation procedure for our approach consists in requesting a prediction (based on the current query) and comparing its correctness with the actual next query. By doing this for a series of queries, we get an overall view of prediction correctness. First, we compare the predicted and actual query incorporating the similarity measure $\sigma_{\text{que}}(q, q')$ defined in section 4, taking for q_1 the predicted query q_{pred}, and for q_2 the actual next query q_{act}. Then, we define the probability of correctness P, which is the probability of the most likely prediction, i.e. the query which has the highest probability of being executed next according to the prediction model. In statistics, this corresponds to the confidence one has in a classification.

The performance of the prediction approach can then be assessed by using the following metric:

$$S = \frac{1}{n} \sum_{t=1}^{n} \pi_t$$

where π_t equals $\sigma_{\text{que}}(q, q')$ at time t. Since the main goal of predicting the next query is to be able to proactively execute it, we consider that even if the predicted and actual next query are not exactly the same, largely similar operations will be performed and facts prefetched. Therefore, we do not choose the give π_t a value of 0 or 1 exclusively, but to allow for an inexact match between queries by using the value $\sigma_{\text{que}}(q, q')$. A variant of this metric is the thresholded S, being S_t, where only predictions are taken into account that have a probability P above a threshold (which we fixed at 0.3). Figure 3 shows how S and S_t evolve according to the cluster split rate.

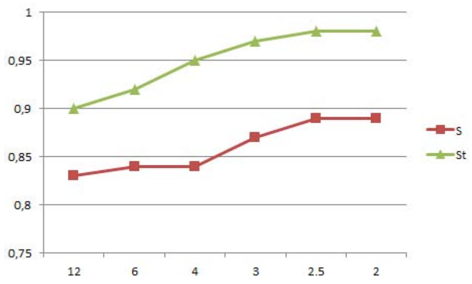

Fig. 3. S and St in function of cluster split rate

In addition we evaluate the recommendation mechanism by using the following metrics:

- *Standard error rate* E_s*:* This corresponds to an unweighted error rate being the proportion of incorrect predictions over all predictions performed.
- *Error rate with threshold* E_t*:* The same as the standard error rate, taking into account only the predictions which have a probability of correctness above a threshold.
- *Coverage C:* This metric is defined as the proportion of items (i.e. queries) for which it is possible to do a recommendation. In our case, we consider it possible to perform a recommendation in case the probability of correctness of the predicted query is above a threshold.

It should be noted that P, the probability of the prediction, can be used to take a decision on whether to effectively execute the predicted query proactively or not. By setting a threshold for P, only queries that are predicted with rather high confidence can be executed, minimizing the risk of executing a wrong query and thus wasting resources. The threshold for P is thus useful in deciding on the quality of an individual concrete prediction. On the other hand, the metric E_t tells something on the general performance in terms of correctness of the prediction approach.

Experiments (see figure 4) show that for a cluster split rate of 2.5 the error rate E_s is 0.44, which is too high for prefetching purposes but could be considered for recommendation purposes. However, if we introduce a threshold T of 0.3 on the probability P of the predictions, the error rate drops to 0.12, which is more acceptable. It should be noted that 26% of predictions were done with a probability of 0.3 or higher. The value of the treshold T is of great importance and influences the success of prediction. When setting it too low (e.g. T=0.2), the error rate increases significantly by 10 to 20%, depending on the number of clusters. When setting T too high (e.g. 0.4), the number of predictions satisfying a P above the treshold decreases dramatically. Moreover, the figure shows that at a cluster split rate of 2.5, E_t stabilizes for lower values of the split rate. The only difference is that when the split rate goes to 2, a more and more lower proportion of predictions satisfies the threshold T, which means the coverage goes down, as can be seen in figure 4. For example, at split rate 2.5, 24% of predictions satisfies T, while at split rate 2.0, this is only 15%. This leads to a choice of balance between E_t and C because the lower E_t, the lower also the ability to do recommendations. Since minimizing E_t is most important to avoid giving erroneous recommendations to the user, and since E_t stabilizes at a split rate of 2.5 while having a coverage of 24%, we identify this point as optimal.

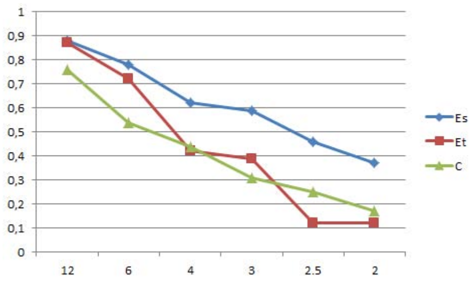

Fig. 4. Es, Et and C in function of cluster split rate

6 Conclusion and Future Work

In this paper, we proposed an approach to query recommendation that combines a probabilistic user behavior model with a query similarity metric. Instead of only relying on the similarity of queries to do a recommendation, the incorporation of the probability of a predicted query allows to define a threshold to decide on the trust one can have in the prediction. Introducing the threshold allows to avoid faulty predictions, improving the quality of experience for the user and avoiding waste of computational resources, while keeping the coverage at an acceptable level. Preliminary evaluation of the approach on a synthetic dataset makes us confident that the recommendation mechanism can provide added value to users in guiding them through their OLAP sessions. As for future work, our goal is to perform an evaluation involving real OLAP users, introducing subjective metrics to gain insight in how appropriate the recommendation is perceived by users. Also, the Markov-based user behavior model will be extended to include in the prediction process not only the current user query, but also characteristics of the whole current OLAP session. Finally, taking into account the n previous queries in doing prediction (higher-order Markov chains) will be investigated as an extension to our approach.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. on Knowl. and Data Eng. 17(6), 734–749 (2005)

2. Baeza-Yates, R., Hurtado, C.A., Mendoza, M.: Query recommendation using query logs in search engines. In: Lindner, W., Fischer, F., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 588–596. Springer, Heidelberg (2004)
3. Khoussainova, N., Balazinska, M., Gatterbauer, W., Kwon, Y., Suciu, D.: A Case for A Collaborative Query Management System. In: CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7. Online Proceedings (2009)
4. Stefanidis, K., Drosou, M., Pitoura, E.: You May Also Like results in relational databases. In: Proceedings International Workshop on Personalized Access, Profile Management and Context Awareness: Databases, Lyon, France (2009)
5. Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query Recommendations for Interactive Database Exploration. In: Winslett, M. (ed.) SSDBM 2009. LNCS, vol. 5566, pp. 3–18. Springer, Heidelberg (2009)
6. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: SnipSuggest: Context-Aware Autocompletion for SQL. PVLDB 4(1), 22–33 (2010)
7. Khoussainova, N., Kwon, Y., Liao, W.-T., Balazinska, M., Gatterbauer, W., Suciu, D.: Session-based browsing for more effective query reuse. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 583–585. Springer, Heidelberg (2011)
8. Drosou, M., Pitoura, E.: Redrive: result-driven database exploration through recommendations. In: Macdonald, C., Ounis, I., Ruthven, I. (eds.) CIKM, pp. 1547–1552. ACM (2011)
9. Sellam, T., Kersten, M.: Meet Charles, big data query advisor. In: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9. Online Proceedings (2013)
10. Giacometti, A., Marcel, P., Negre, E.: A framework for recommending OLAP queries. In: Proc. DOLAP, Napa Valley, CA, pp. 73–80 (2008)
11. Sapia, C.: PROMISE: Predicting query behavior to enable predictive caching strategies for OLAP systems. In: Kambayashi, Y., Mohania, M., Tjoa, A.M. (eds.) DaWaK 2000. LNCS, vol. 1874, pp. 224–233. Springer, Heidelberg (2000)
12. Howard, R.: Dynamic programming and Markov processes. Technology Press of Massachusetts Institute of Technology (1960)
13. Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-driven exploration of OLAP data cubes. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 168–182. Springer, Heidelberg (1998)
14. Sarawagi, S.: User-adaptive exploration of multidimensional data. In: Proc. VLDB, Cairo, Egypt, pp. 307–316 (2000)
15. Aligon, J., Golfarelli, M., Marcel, P., Rizzi, S.: Similarity measures for OLAP sessions. International Journal of Knowledge and Information Systems (to appear, 2013)
16. Giacometti, A., Marcel, P., Negre, E.: Recommending multidimensional queries. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 453–466. Springer, Heidelberg (2009)
17. Drosou, M., Pitoura, E.: ReDRIVE: result-driven database exploration through recommendations. In: Proc. CIKM, Glasgow, UK, pp. 1547–1552 (2011)
18. Yang, X., Procopiuc, C., Srivastava, D.: Recommending join queries via query log analysis. In: Proc. ICDE, Shanghai, China, pp. 964–975 (2009)
19. Chatzopoulou, G., Eirinaki, M., Koshy, S., Mittal, S., Polyzotis, N., Varman, J.S.V.: The querie system for personalized query recommendations, 55–60 (2011)