QuRVe: Query Refinement for View Recommendation in Visual Data Exploration

Humaira Ehsan[1(✉)], Mohamed A. Sharaf[2], and Gianluca Demartini[1]

[1] The University of Queensland, Brisbane, QLD, Australia
humairaehsan@gmail.com
[2] United Arab Emirates University, Al Ain, Abu Dhabi, UAE
msharaf@uaeu.ac.ae

Abstract. The need for efficient and effective data exploration has resulted in several solutions that automatically recommend interesting visualizations. The main idea underlying those solutions is to automatically generate all possible views of data, and recommend the top-k interesting views. However, those solutions assume that the analyst is able to formulate a well-defined query that selects a subset of data, which contains insights. Meanwhile, in reality, it is typically a challenging task to pose an exploratory query, which can immediately reveal some insights. To address that challenge, this paper proposes to automatically refine the analyst's input query to discover such valuable insights. However, a naive query refinement, in addition to generating a prohibitively large search space, also raises other problems such as deviating from the user's preference and recommending statistically insignificant views. In this paper, we address those problems and propose the novel QuRVe scheme, which efficiently navigates the refined queries search space to recommend the top-k insights that meet all of the analysts's pre-specified criteria.

1 Introduction

Visual data exploration is the rudiment of deriving insights from large datsets. Typically, it involves an analyst performing the following steps: 1) selecting a subset of data, 2) generating different visualizations of that subset of data, and 3) sifting through those visualizations for the ones which reveal interesting insights. Based on the outcome of the last step, the analyst might have to refine their initial selection of data so that the new subset would show more interesting insights. This is clearly an iterative and time-consuming process, in which each selection of data (i.e., exploratory input query) is a springboard to the next one.

Motivated by the need for an efficient and effective visual data exploration process, several solutions have been proposed towards automatically finding and recommending interesting data visualizations (i.e., steps 2 and 3 above) (e.g., [6–8,14,17]). The main idea underlying those solutions is to automatically generate all possible views of the explored data, and recommend the top-k *interesting*

J. Darmont et al. (Eds.): ADBIS 2020, CCIS 1259, pp. 154–165, 2020.
https://doi.org/10.1007/978-3-030-54623-6_14

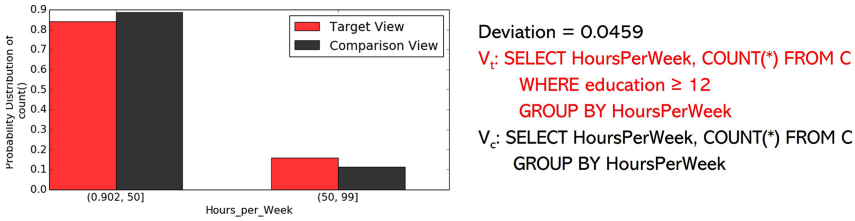

Fig. 1. View on input query Q

views, where the interestingness of a view is quantified according to some *utility* function. Recent work provides strong evidence that a deviation-based formulation of utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [8,14,17,19]. In particular, the deviation-based metric measures the distance between the probability distribution of a visualization over the analyzed dataset (i.e., *target view*) and that same visualization when generated from a comparison dataset (i.e., *comparison view*), where the comparison dataset is typically the entire database. The underlying premise is that a visualizations that results in a higher deviation is expected to reveal insights that are very particular to the analyzed dataset.

Existing solutions have been shown to be effective in recommending interesting views under the assumption that the analyst is "precise" in selecting their analyzed data. That is, the analyst is able to formulate a well-defined exploratory query, which selects a subset of data that contains interesting insights to be revealed by the recommended visualizations. However, such assumption is clearly impractical and extremely limits the applicability of those solutions. In reality, it is typically a challenging task for an analyst to select a subset of data that has the potential of revealing interesting insights. Hence, it is a continuous process of trial and error, in which the analyst keeps *refining* their selection of data manually and iteratively until some interesting insights are revealed. Therefore, in this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. Hence, our goal in this work is not only to recommend interesting views, but also to recommend exploratory queries that lead to such views. To further illustrate the need for such solution, consider the following example.

Example 1. Consider an analyst wants to explore and find interesting insights in the U.S. Census income dataset [1], which is stored in table C. Her intuition is that analyzing the subset of data of those who have achieved a high level of education might reveal some interesting insights. Therefore, she selects that particular subset in which everyone has completed their 12th year of education (i.e., graduated high school) via the query: Q: `SELECT * FROM C WHERE education` $\geq$ `12`. To find the top-k visualizations, she might use one of the existing approaches (e.g., [8,17]), in which all the target and comparison aggre-

gate views are generated and their deviation is computed by using a distance function (e.g., Euclidean distance). Figure 1 shows the top-k visualization recommended by such approaches. Particularly, the figure shows a bar chart in which the x-axis is the dimension `Hours per week`, and the y-axis is the probability distribution of the aggregate function `COUNT`. Such visualization is equivalent to plotting the probability distributions of the target view V_t and the comparison View V_c, which are expressed in SQL in Fig. 1. Hence, the deviation value shown in Fig. 1 is the Euclidean distance between the probability distribution of V_t and V_c. However, Fig. 1 clearly shows that the target and comparison views are almost the same, which is also reflected by the low-deviation value of 0.0459. However, such visualization would still be recommended by existing approaches because it achieves the maximum deviation among all the views generated over the data subset selected by query Q.

The previous example illustrates a clear need for a query refinement solution that is able to automatically modify the analyst's initial input query and recommend a new query, which selects a subset of data that includes interesting insights. To that end, one straightforward and simple approach would involve generating all the possible subsets of data by automatically refining the predicates of the input query. Consequently, for each subset of data selected by each query refinement, generate all possible aggregate views (i.e., visualizations). In addition to the obvious challenge of a prohibitively large search space of query refinements, that naive approach would also lead to visualizations that might appear to be visually interesting but they are irrelevant from the analyst's perspective. Particularly, there are two issues with that approach, and in turn the recommended visualization; 1) *similarity-oblivious*: a blind automated refinement that is oblivious to the analyst's preferences might result in a refined query that is significantly dissimilar from the input query, and 2) *statistical insignificance*: the subset selected by the refined query can be too small and as a result the target views generated from that subset will miss a number of values for the dimension attribute. This leads to views with high deviation values but statistically insignificant.

The two issues mentioned above highlight the need for automatic refinement solutions that are guided by the user's preference and statistical significance, which is the focus of this work. In particular, we propose a novel scheme **QuRVe**, which is particularly optimized to leverage the specific features of the problem for pruning that large search space, as explained in the next sections.

2 Preliminaries

2.1 View Recommendation

Similar to the recent data visualization platforms [8,17], we are given a multi-dimensional dataset $D(\mathbb{A}, \mathbb{M})$, where $\mathbb{A}$ is the set of dimension attributes, $\mathbb{M}$ is the set of measure attributes, and $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$. In a typical visual data exploration session

the user chooses a subset D_S of the dataset D by issuing an input query Q. For instance, consider the query Q: `SELECT * FROM D WHERE T;` . In Q, T specifies a combination of predicates, which selects D_S for visual analysis (e.g., `education` $\geq$ `12` in Ex. 1). A visual representation of Q is basically the process of generating an aggregate view V_i of its result (i.e., D_S), which is then plotted using some visualization methods such as bar charts, scatter plots, etc. Therefore, an aggregate view V_i over D_S is represented by a tuple (A, M, F, b) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, $F \in \mathbb{F}$ and b is the number of bins in case A is numeric. That is, D_S is grouped by dimension attribute A and aggregated by function F on measure attribute M. For instance, the tuple `(Hours per Week, *,COUNT,2)` represents the aggregate view shown in Fig. 1.

Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on deviation based metric (e.g., [8,17]). In particular, it measures the deviation between the aggregate view V_i generated from the subset data D_S vs. that generated from the entire database D, where $V_i(D_S)$ is denoted as *target* view, whereas $V_i(D)$ is denoted as *comparison* view. To ensure that all views have the same scale, each target view $V_i(D_S)$ and comparison view $V_i(D)$ is normalized into a *probability distribution* $P[V_i(D_S)]$ and $P[V_i(D)]$ and it is bounded by the maximum deviation value D_M. Accordingly, the deviation $D(V_i)$, provided by a view V_i, is defined as the normalized distance between those two probability distributions.

$$D(V_i) = \frac{dist(P[V_i(D_S)], P[V_i(D)])}{D_M} \tag{1}$$

Then, the deviation $D(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) [8,9,17,19]. However, to ensure that those top-k recommended views reveal interesting insights, we propose utilizing query refinement techniques, which are explained next.

2.2 Query Refinement

Automatic query refinement is a widely used technique for DBMS testing, information retrieval and data exploration. In a nutshell, in this technique the user provides an initial query and then it is progressively refined to meet a particular objective [13,16,18,20]. In this work, we propose to automatically refine an input exploratory query for the objective of view recommendation. Particularly, as mentioned in Sect. 2.1, the user provides an input query Q, which is progressively refined by automatically enumerating all combinations of predicates for the objective of generating interesting views.

Particularly, we consider queries having selection predicates with range $(<, \leq, >, \geq)$ operators. These predicates are defined on a set of numeric dimension attributes denoted as $\mathbb{P}$. The number of predicates is p, such that $|\mathbb{P}| = p$. Each of this range predicate is in the form $l_i \leq P_i \leq u_i$ where $P_i \in \mathbb{P}$ and l_i and u_i are the lower and upper limits of query Q along predicate P_i. The domain of predicate P_i is limited by a Lower bound L_i and upper bound U_i. A refined query Q_j is generated by modifying the lower and/or upper limits for some of

the predicates in Q. That is, for a predicate $l_i \leq P_i \leq u_i$ in query Q, a refined predicate in Q_j takes the form $l'_i \leq P_i \leq u'_i$. Similar to [2,13], we convert a range predicate into two single-sided predicates. Therefore, $l_i \leq P_i \leq u_i$ is converted to two predicates: $P_i \leq u_i \bigwedge -P_i \leq -l_i$. This allows refinement of one or both sides of the range predicates and this results in the total number of single sided predicates to be $2p$. The set of all of the refined queries is denoted as $\mathbb{Q}$.

A refined query Q_j is obtained by changing one or more predicates $P_i \in T$ to P'_i, which naturally makes the refined query Q_j different from the input query Q. However, a refined query that is significantly dissimilar from its counterpart input query would result in loss of user preference and might be deemed irrelevant to the analysis. Hence, to quantify the change made to transform Q into the refined query Q_j, we define a similarity measure $S(Q, Q_j)$ in terms of the distance between Q_j and Q (i.e., $s(Q, Q_j)$).

$$S(Q, Q_j) = 1 - s(Q, Q_j) \tag{2}$$

While the exact specification of $s(Q, Q_j)$ is deferred to Sect. 4, it is worth pointing out the impact of query refinement on the deviation computation defined in Eq. 1. Particularly, when utilizing refinement, a view V_i can be either generated from the input query, or a refined one. To associate each view with its underlying query, we denote a view as V_{i,Q_j} to specify the i^{th} view generated over the result of query Q_j. Accordingly, Eq. 1 is modified to define the deviation $D(V_{i,Q_j})$ of a view V_{i,Q_j}, as:

$$D(V_{i,Q_j}) = \frac{dist(P[V_i(D_{Q_j})], P[V_i(D)])}{D_M} \tag{3}$$

2.3 Hypothesis Testing

In visual data exploration, it is often the case that an observed high-deviation is actually statistically insignificant. This problem leads to misleading ranking of such views, and in turn inaccurate recommendations [3,4,21]. For instance, in our recent work on the *MuVE* scheme [8,9], we made the following observations:

1. Some of the recommended top-k target views have very few underlying tuples, which lead to higher deviation values. Consequently, such views receive higher rank despite of the lack of real insight.
2. Often the data selected by the exploratory query result in only low-deviation views. Consequently, the top-k recommend views will exhibit low-deviation, as shown in Ex. 1. However, such top-k recommendations are clearly statistically insignificant.

To determine whether the observed difference is statistically significant, we employ the widely used approach hypothesis testing. Hypothesis testing determines if there is enough evidence for inferring that a difference exists between two compared samples or between a sample and population. A difference is called

statistically significant if it is unlikely to have occurred by chance [3]. Hypothesis testing involves testing a null hypothesis by comparing it with an alternate hypothesis. The hypothesis to be tested is called the *null hypothesis*, denoted as H_0. The null hypothesis states that there is no difference between the population and the sample data. The null hypothesis is tested against an *alternate hypothesis*, denoted as H_1, which is what we have observed in the sample data. For instance, in Fig. 1 of Ex. 1, the hypothesis is that *"high school graduates work different number of hours per week (Hours worked is divided into two categories) as compared to the population"*, and this becomes H_1. The corresponding H_0 is that no such difference exits. Likewise, each possible view V_i from each refined query become a H_1, which is to be tested for significance before recommendation.

Depending on the nature of the statistical test and the underlying hypothesis, different null hypothesis statistical tests have been developed, e.g., chi-square test for categorical dimension attributes. Furthermore, after stating H_0 and H_1, the chosen statistical test returns *p-value*. The p-value is the probability of obtaining a statistic at least as extreme as the one that was actually observed, given H_0 is true. Specifically, the p-value is compared against a priori chosen *significance level* α, where the conventionally used significance level is 0.05. Hence, if $pvalue(V_i) \leq \alpha$, then H_0 must be rejected, which means the V_i is statistically significant. Clearly, due to the nature of the statistical test involved, the acceptance or rejection of H_0 can never be free of error. If the test incorrectly rejects or accepts H_0, then an error has occurred. Hypothesis testing can incur the following two types of error: 1) If H_0 is rejected, while it was true, it is called *Type-I error* and 2) If H_0 is accepted, while H_1 was true, it is called *Type-II error*. Type-II error is critical in our case because we do not want to reject views that might be interesting. The probability of Type-II error is specified by a parameter β, which normally has a value $0.10 - 0.20$. An alternate term is power, which is the probability of rejecting a false H_0, therefore, $power = 1 - \beta$. A priori power analysis is employed to determine the minimum sample size that is necessary to obtain the required power. By setting an effect size (ω), significance level (α), and power level (β), the sample size to meet specification can be determined [5].

3 Query Refinement for View Recommendation

In a nutshell, the goal of this work is to recommend the top-k bar chart visualizations of the results of query Q and all its corresponding refined queries $Q_j \in \mathbb{Q}$, according to some utility function. However, that simple notion of utility falls short in capturing the impact of refinement on the input query. In particular, automatic refinement introduces additional factors that impact the level of interestingness, and in turn utility of the recommended views. Accordingly, in our proposed scheme, we employ a weighted multi-objective utility function and constraints to integrate such factors. In particular, for each view V_{i,Q_j}, we evaluate the following components:

1. *Interestingness:* Is the ability of view V_{i,Q_j} to reveal some insights about the data, which is measured using the deviation-based metric $D(V_{i,Q_j})$ (Eq. 3).

2. *Similarity:* Is the similarity between the input query Q, and the refined query Q_j underlying the view V_{i,Q_j}, which is measured as $S(Q, Q_j)$ (Eq. 2).
3. *Statistical Significance:* Is the ability of the refined query Q_j and the view V_{i,Q_j} to generate a statistically significant result, which is captured by checking that the size of the subset selected by Q_j satisfies the constraint $power(Q_j)$, and the significance of the view V_{i,Q_j} satisfies the constraint $pvalue(V_{i,Q_j})$.

To capture the factors and constraints mentioned above, we employ a weighted multi-objective utility function, which is defined as follows:

$$U(V_{i,Q_j}) = \alpha_S \times S(Q, Q_j) + \alpha_D \times D(V_{i,Q_j}) \tag{4}$$

Parameters α_S and α_D specify the weights assigned to each objective in our hybrid utility function, such that $\alpha_S + \alpha_D = 1$. Those weights can be user-defined so that to reflect the user's preference between interestingness and similarity. Also, notice that all objectives are normalized in the range $[0, 1]$.

To fully define the similarity component of our utility function, we revisit Eq. 2 which quantifies the distance between Q and Q_j. In literature, a number of methods have been proposed to measure the distance between two range queries [11,15,16]. Similar to [2,18], we calculate the distance in terms of absolute change in predicate values (Eq. 5). This method provides a reasonable approximation of the change in data selected by the refined query at a negligible cost. Additionally, we normalize it by predicate bounds to accommodate the different scales of various predicates.

$$s(Q, Q_j) = \frac{1}{p} \sum_{i=1}^{p} \frac{|l_i^{Q_j} - l_i^{Q}| + |u_i^{Q_j} - u_i^{Q}|}{2|U_i - L_i|} \tag{5}$$

Definition: Query Refinement for View Recommendation: *G*iven a user-specified query Q on a database D, a multi-objective utility function U, a significance level α, statistical power $1 - \beta$ and a positive integer k. Find k aggregate views that have the highest utility values, from all of the refined queries $Q_j \in \mathbb{Q}$ such that $pvalue(V_{i,Q_j}) \leq \alpha$ and $power(Q_j) > 1 - \beta$.

In short, the premise is that a view is of high utility for the user, if it satisfies the specified constraints, shows high-deviation, and is based on a refined query that is highly similar to the user specified query.

4 Search Schemes

For an input query Q, each possible query refinement of Q can be represented as a point in p-dimensional space, where $|\mathbb{P}| = p$ (please see Sect. 2.2 for more details). Clearly, one of the points in that space is the input query Q itself, and the remaining points belong to the set of refined queries $\mathbb{Q}$. Our high-level goal is to: 1) generate the set $\mathbb{Q}$, 2) compute the utility of all the aggregate views generated from each query in $\mathbb{Q}$, and 3) recommend the top-k views after ranking them based on their achieved utility. To that end, clearly the large size

of $\mathbb{Q}$ and the corresponding aggregate views, together with the complexity of evaluating the statistical significance and utility function of each view, makes the problem highly challenging. Hence, in this section, we put forward various search strategies for finding the top-k views for recommendation.

4.1 The Linear Scheme

Clearly, a naive way to identify the top-k objects is to score all objects based on a scoring function, sort, and return the top-k objects. Accordingly, the Linear scheme is basically an exhaustive and brute force strategy, in which views from all refined queries are generated and ranked according to their utility. As we consider predicates on continuous dimensions, infinite possible values can be assigned to predicates in those refined queries. Therefore, each dimension is discretized with a user specified parameter γ. This divides the range of dimension attribute into $1/\gamma$ equi-width intervals. In this scheme, irrespective of Q, iteratively all refined queries are generated using all combinations of Predicates $P_1, P_2 ... P_p$.

For each query $Q_j \in \mathbb{Q}$, to check the constraint $power(Q_j) < 1 - \beta$, a function *powerTest*(Q_j, ω, β, α) is defined, which returns true value if the constraint is satisfied, else it returns false. The cost of checking this constraint is one database probe, where a `COUNT` query with predicates of Q_j is executed to get the sample size of Q_j. Moreover, for the queries that satisfy the statistical power constraint, all views are generated. Then for each view V_{i,Q_j} the constraint $pValue(V_{i,Q_j}) < \alpha$ is checked. Specifically, for this purpose, another function *significanceTest*(V_{i,Q_j}, α) is defined, which returns a true value if p-value $< \alpha$. Consequently, for each view V_{i,Q_j} that satisfies the constraint, its utility value $U(V_{i,Q_j})$ is computed, and finally the top-k views are returned.

4.2 The QuRVe Scheme

Clearly, the linear search scheme, visits every possible view, therefore, it is very expensive in terms of execution time. In this section, we present the QuRVe scheme, which reduces cost by pruning a large number of views. Notice that our problem of finding top-k views is similar to the problem of top-k query processing, which is extensively studied in various settings [12]. Generally in these settings objects are evaluated by multiple objectives that contribute to the overall score of each object. In terms of efficiency, the best performing techniques for various top-k problem settings are based on the threshold algorithm (TA) [10,12]. TA generates sorted lists of objects on partial scores for every objective, visits the lists in round robin fashion and merges objects from different lists to compute the aggregated scores. Typically, it *early terminates* the search as soon as it has the *top-k* objects.

In our settings, we have a similar configuration i.e., we have two partial scores of a view V_{i,Q_j}, namely: 1) Similarity score $S(Q, Q_j)$, 2) Deviation score $D(V_{i,Q_j})$. These are stored in S_{list} and D_{list}. Conversely, we also have some key differences: 1) for any view V_{i,Q_j} the values of $S(Q, Q_j)$ and $D(V_{i,Q_j})$ are not physically stored and are computed on demand, 2) calculating $D(V_{i,Q_j})$ for

a view is an expensive operation, and 3) the size of the view search space is prohibitively large and potentially infinite.

Obviously, a forthright implementation of TA is infeasible to our problem due to the limitations mentioned before. However, recall that the similarity objective $S(Q, Q_j)$ is the comparison of predicates of Q_j with Q and involves no database probes. Hence, a sorted list S_{list} can be easily generated at a negligible cost. However, populating the D_{list} in a similar fashion is not possible, as it involves expensive database probes. Therefore, to minimize the number of probes and efficiently populate D_{list}, the Sorted-Random (SR) model of the TA algorithm [12] is employed. In the SR model the sorted list (S) provides initial list of candidates and the random list (R) is probed only when required. Accordingly, QuRVe provides S_{list} as the initial list of candidate views, by incrementally generating refined queries in decreasing order of similarity and populating the S_{list}. The views in S_{list} have their partial scores, the final scores are only calculated for the views for which the D_{list} is also accessed. To achieve this, QuRVe maintains the variables U_{Unseen}: Stores the maximum utility of the views that are not probed yet and U_{Seen}: Stores the k^{th} highest utility of a view seen so far. Specifically, to calculate U_{Unseen}, the upper bound on deviation is used. Particularly, consider V_{i,Q_j} as the next view in S_{list} and let the upper bound on its deviation be $D_u(V_{i,Q_j})$. Moreover, let the upper bound on deviation from all views be D_u then $D_u = Max[D_u(V_{i,Q_j})]$. Consequently, $U_{Unseen} = \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u$. As the $D(V_{i,Q_j})$ is the normalized deviation, hence theoretically $D_u = Max[D_u(V_{i,Q_j})] = 1$.

In detail, QuRVe starts with initializations as: (i) there are no views generated yet, therefore $U_{Seen} = 0$, (ii) $U_{Unseen} = D_u$, and (iii) Q has the highest similarity i.e., $S(Q, Q) = 1$, therefore, Q is added to Q_{list} as the first member. Then, the power of the currently under consideration query Q_j is checked by the function *powerTest*(Q_j, ω, β, α). Next, the corresponding views are generated by the function *generateViews*(Q_j) and the statistical significance test is performed on each view by the function *significanceTest*(V_{i,Q_j}, α). The Utility of the views that pass the test is computed. Accordingly, the list *topk* is updated. The utility of k^{th} highest view is copied into U_{Seen}, to maintain the bound on the seen utility values. This completes processing the currently under consideration query Q_j. Later, the next set of neighboring queries are generated. In next iteration, another query Q_j is taken from the Q_{list} in order of the similarity objective value and accordingly the value of U_{Unseen} is updated. The iteration continue, until either there are no more queries to process, or the utility of the remaining queries will be less than the already seen utility (i.e., $U_{Unseen} > U_{Seen}$ is false). If QuRVe terminates because of the first condition that means its cost is the same as Linear search, as the optimization did not get a chance to step in. However, often QuRVe terminates because of the second condition (i.e., $U_{Unseen} > U_{Seen}$ is false) and achieves early termination. The most efficient performance of QuRVe is expected when U_{Unseen} decreases quickly during search and early termination can be triggered.

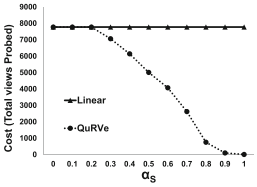

Fig. 2. Impact of α_S and α_D on cost

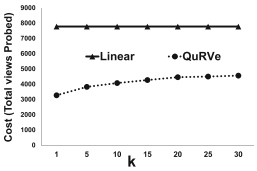

Fig. 3. Impact of k, $\alpha_S = 0.6$

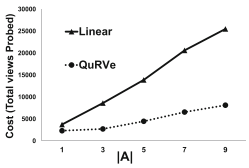

Fig. 4. Impact of dimensions

5 Experimental Evaluation

We perform extensive experimental evaluation to measure both the efficiency and effectiveness of out proposed schemes for top-k view recommendation.

Data Analysis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. We use *CENSUS:* the census income dataset [1]. The dataset has 14 attributes and 48,842 tuples. The independent categorical attributes of the dataset are used as dimensions (e.g., occupation, work class, hours per week, sex, etc.), whereas the observation attributes are used as measures (capital gain, etc.) and the numerical independent attributes are used for predicates (e.g., education, age, etc.). The aggregate functions used are SUM, AVG and COUNT. In the analysis, all the α_S is in the range $[0-1]$, where $\alpha_S + \alpha_D = 1$. In default settings $\alpha_S = 0.5$, $k = 10$ and $\gamma = \frac{1}{2^3}$. For the purpose of statistical significance in experiments we use chi-square goodness of fit test.

Performance: We evaluate the efficiency and effectiveness of the different recommendations strategies in terms of the cost incurred. As mentioned in Sect. 3, the cost of a strategy is the total cost incurred in processing all the candidate views. We use the total views probed as the cost metric. Each experiment is performed with 10 randomly generated input queries, spread around the search space, then average of the cost is taken.

Impact of the α Parameters (Fig. 2): In this experiment, we measure the impact of the α values on cost. Figure 2 shows how the cost of *Linear* and *QuRVe* schemes are affected by changing the values of α_S. In Fig. 2, α_S is increasing while α_D is implicitly decreasing. Notice that the *Linear* scheme has the same cost for all values of α_S, which is as expected since it performs exhaustive search over all views. Therefore, its cost depends on the number of all combinations, irrespective of α. Figure 2 also shows that *QuRVe* has almost the same cost as *Linear* for $\alpha_S = 0$ and $\alpha_S = 0.1$, but outperforms it as the value of α_S increases. This is because in *QuRVe*, the upper bound on deviation is set to the theoretical maximum bound i.e., $D_u = 1$ and when $\alpha_S = 0$, $U_{Unseen} = 0 \times S + 1 \times D_u = 1$, consequently early termination is not possible. On the contrary, as α_S increases, chances of applying early termination based on the similarity value becomes possible. Hence, this prunes many database probes.

Impact of k (Fig. 3): Figure 3 shows that *Linear* is insensitive to the increase in the value of k, because it visits all views and sorts them according to their utility irrespective of the value of k. For $\alpha_S = 0.6$, *QuRVe* performs better than *Linear* for all values of k, due to the early termination feature of *QuRVe* scheme.

Scalability (Fig. 4): The search space of our problem depends on p, γ, $|A|$, $|M|$ and $|F|$. Increasing any of these factors explodes the search space. Consequently, the cost of all schemes increases as there are more views that are visited in search for the top-k views. Figure 4 shows the impact of number of the dimensions on cost. Particularly for this experiment the number of dimensions ($|A|$) are increased from 1 to 9. Figure 4 shows that the increase in the cost of *Linear* is linear with the increase in $|A|$. However, for *QuRVe* the increase in the cost is slow. Particularly, the cost of *QuRVe* increases as the number of dimension are increased from 1 to 3, because the schemes search from more views to generate top-k views. But for $|A| = 5$ the cost decreases this is because a new dimension is added which had views with high deviation and that resulted in increasing the value of U_{Seen} and triggering early termination.

6 Conclusions

Motivated by the need for view recommendation that lead to interesting discoveries and avoid common pitfalls such as random or false discoveries. In this paper we formulated the problem of query refinement for view recommendation and proposed QuRVe scheme. QuRVe refines the original query in search for more interesting views. It efficiently navigates the refined queries search space to maximize utility and reduce the overall cost. Our experimental results show employing the QuRVe scheme offer significant reduction in cost.

Acknowledgement. This work is partially supported by UAE University grant G00003352.

References

1. https://archive.ics.uci.edu/ml/datasets/adult
2. Albarrak, A., Sharaf, M.A., Zhou, X.: SAQR: an efficient scheme for similarity-aware query refinement. In: Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B. (eds.) DASFAA 2014. LNCS, vol. 8421, pp. 110–125. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05810-8_8
3. Bay, S.D., Pazzani, M.J.: Detecting group differences: mining contrast sets. Data Min. Knowl. Disc. **5**(3), 213–246 (2001). https://doi.org/10.1023/A:1011429418057
4. Chung, Y., et al.: Towards quantifying uncertainty in data analysis & exploration. IEEE Data Eng. Bull. **41**(3), 15–27 (2018)
5. Cohen, J.: Statistical Power Analysis for the Behavioral Sciences, revised edn. Academic Press, Cambridge (1977)
6. Demiralp, Ç., et al.: Foresight: recommending visual insights. PVLDB **10**(12), 1937–1940 (2017)

7. Ding, R., et al.: Quickinsights: quick and automatic discovery of insights from multi-dimensional data. In: SIGMOD, pp. 317–332 (2019)
8. Ehsan, H., et al.: MuVE: efficient multi-objective view recommendation for visual data exploration. In: ICDE, pp. 731–742 (2016)
9. Ehsan, H., et al.: Efficient recommendation of aggregate data visualizations. IEEE Trans. Knowl. Data Eng. **30**(2), 263–277 (2018)
10. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
11. Kadlag, A., Wanjari, A.V., Freire, J., Haritsa, J.R.: Supporting exploratory queries in databases. In: Lee, Y.J., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 594–605. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24571-1_54
12. Marian, A., et al.: Evaluating top- k queries over web-accessible databases. ACM Trans. Database Syst. **29**(2), 319–362 (2004)
13. Mishra, C., Koudas, N.: Interactive query refinement. In: EDBT (2009)
14. Sellam, T., et al.: Ziggy: characterizing query results for data explorers. PVLDB **9**(13), 1473–1476 (2016)
15. Telang, A., et al.: One size does not fit all: toward user and query dependent ranking for web databases. IEEE Trans. Knowl. Data Eng. **24**(9), 1671–1685 (2012)
16. Tran, Q.T., et al.: How to conquer why-not questions. In: SIGMOD (2010)
17. Vartak, M., et al.: SEEDB: efficient data-driven visualization recommendations to support visual analytics. PVLDB **8**(13), 2182–2193 (2015)
18. Vartak, M., et al.: Refinement driven processing of aggregation constrained queries. In: EDBT (2016)
19. Wang, C., et al.: Efficient attribute recommendation with probabilistic guarantee. In: KDD, pp. 2387–2396 (2018)
20. Wu, E., et al.: Scorpion: explaining away outliers in aggregate queries. PVLDB **6**(8), 553–564 (2013)
21. Zhao, Z., et al.: Controlling false discoveries during interactive data exploration. In: SIGMOD (2017)