Version of Record: https://www.sciencedirect.com/science/article/pii/S2214579621000551
Manuscript_baa2adae098a27e6945b1f4e6cc32cb0

ViewSeeker: An Interactive View Recommendation Framework

Xiaozhong Zhang[a], Xiaoyu Ge[a], Panos K. Chrysanthis[a], Mohamed A. Sharaf[b]

[a] *Department of Computer Science, University of Pittsburgh, USA*
[b] *Department of Computer Science, United Arab Emirates University, UAE*

Abstract

View recommendations have emerged as a powerful tool to assist data analysts in exploring and understanding big data. Existing view recommendation approaches proposed a variety of utility functions in selecting useful views. However, the suitability of the utility functions and their tunable parameters for an analysis is usually dependent on the analysis context, such as the user, the data and the analysis task. In order to provide *context-aware* view recommendation, we formulate a new *Interactive View Recommendation* (IVR) paradigm, where the system interacts with the user to discover the utility functions that are most suitable in the current analysis context. We further develop an IVR framework, coined `ViewSeeker`, which leverages user feedback on intelligently selected example views to discover the most suitable utility functions. Finally, we implemented a prototype of `ViewSeeker` and verified its efficiency and effectiveness using two real-world datasets.

1. Introduction

The ubiquitously available information sources and the advancements in data storage and acquisition techniques have led to an aggressive increase in the data volumes available for data analysis. One major challenge in utilizing these abundantly available data is discovering insights from them effectively and efficiently. Examples of an "insight" include the structure, patterns, and causal relationships. To explore these massive and structurally complicated datasets, data analysts often utilize visual data analysis tools, such as Tableau, Qlik, Lyra, Amazon Quicksight, Microsoft Power BI, Google Fusion Tables etc. [1]. However, the effectiveness of these tools depends on the user's expertise and experience. Coming up with a visualization that shows interesting trends/patterns is a non-trivial issue. Commonly, the analyst needs to examine the relationships among various attributes and consider various aggregate functions before any useful visualizations can be discovered. This approach is typically ad-hoc, labor-intensive, and not scalable, especially for high-dimensional databases.

To address such shortcomings, several methods for recommending visualizations have recently been proposed (e.g., [2–8]). These methods automatically generate all possible

*All authors are corresponding authors.
Email addresses: `xiz151@pitt.edu` (Xiaozhong Zhang), `xig34@pitt.edu` (Xiaoyu Ge), `panos@cs.pitt.edu` (Panos K. Chrysanthis), `msharaf@uaeu.ac.ae` (Mohamed A. Sharaf)

March 22, 2021

© 2021 published by Elsevier. This manuscript is made available under the Elsevier user license
https://www.elsevier.com/open-access/userlicense/1.0/

views of data, and *recommend* the *top-k interesting* views, according to some utility function (e.g., deviations, data variance, usability) that measures the interestingness of the view. However, the suitability of the utility functions and their tunable parameters for an analysis is usually dependent on the analysis context, such as the user, the data and the analysis task. Thus, existing view recommendation methods with a-priori defined utility functions cannot adapt to the analysis context.

In order to provide *context-aware* view recommendation, we formulate a new *Interactive View Recommendation* (IVR) paradigm, in which the system interacts with the user to determine the most suitable utility functions. Specifically, an IVR system interacts with the user in an iterative fashion, and uses the user feedback to gradually refine its estimation of the suitability of the utility functions, with the goal of determining the most suitable utility functions in the current analysis context.

There are a number of ways to determine the utility function suitability from user feedback, such as using decision rules or machine learning. In this paper, we propose an IVR framework, called `ViewSeeker`, which adopts the machine learning model-based approach. `ViewSeeker` supports two forms of adaptation: *utility function tuning* and *utility function integration*[1]. Utility function integration uses *active-learning* technique [13] to select example views for labeling and to predict the contribution of each utility function in a multi-objective "ideal" utility function in the current analysis context. Utility function tuning is an additional functionality of `ViewSeeker`, which interacts with the user to determine the most suitable parameters of a utility function in the current analysis context.

To verify the effectiveness and efficiency of `ViewSeeker`, we implemented a prototype system and experimentally evaluated it using a dataset of diabetic patients [14] and a census dataset from the U.S. labor force survey [15].

To summarize, the contributions of this paper are the following:

1. Formulate the new human-in-the-loop, *Interactive View Recommendation* (IVR) paradigm for context-aware view recommendations.
2. Propose *ViewSeeker*, an IVR framework with deviation-based utility function tuning and integration functionality, with the goal of efficiently discovering the ideal utility function that is most suitable in the current analysis context.
3. Implement a prototype system of the proposed `ViewSeeker`, and verify its effectiveness using two real-world datasets.

Outline The rest of the paper is structured as follows. Section 2 introduces the IVR paradigm and the `ViewSeeker` framework. Section 3 presents the utility function tuning functionality of `ViewSeeker`. Section 4 presents the utility function integration functionality of `ViewSeeker`. Section 5 describes our experimental evaluation. Section 6 discusses related works and Section 7 presents our conclusions.

[1]Utility function tuning is the main contribution of this paper whereas utility function integration combines and extends the functionality presented in our previous work [9–12].

2. Problem Formulation

In this section, we first discuss how views can be constructed through SQL queries and explain how the interestingness of a view may be captured through a predefined utility function. Then, we formally present the proposed *Interactive View Recommendation (IVR)* paradigm, and our IVR framework `ViewSeeker` that address the *Context-Aware View Recommendation (CoVR)* problem, which motived our work.

2.1. Views & Data Visualization Recommendation

In the context of structured databases, a view (i.e., histogram or bar chart) essentially represents an SQL query with a group-by clause over a database D [2, 6]. Under the typical multi-dimensional data models, data can be modeled as a set of measure attributes $M = \{m_1, m_2, m_3, ...\}$ and a set of dimension attributes $A = \{a_1, a_2, a_3, ...\}$. The measure attributes (e.g., the number of items sold) are the set of attributes that contain measurable value and can be aggregated. The dimensional attributes (e.g., brand, year, color, size) are the set of attributes on which measure attributes are viewed. Finally, let $F = \{f_1, f_2, f_3, ...\}$ be the set of standard SQL aggregate functions that are applied to generate an aggregate query/view with a group-by clause. Thus, we can represent each view v_i as a triple (a, m, f), such that one aggregate function f is applied on dimension attribute a over the corresponding measure attribute m. Consequently, the View Space (VS), i.e., the total number of possible views is:

$$VS = |A| \times |M| \times |F| \tag{1}$$

Clearly, VS can be large, especially for high-dimensional data. In order to recommend the set of k most interesting views from a large number of views, utility scores are required to rank the views. To compute such utility scores, existing literature has proposed several utility functions (UFs). Some commonly used UFs include *deviation* [2], *accuracy* [5], *usability* [5] and *p-value* [16]. The typical view recommendation problem can be defined as follows:

Definition 1. (*View Recommendation Problem*) Given a database D, a user-specified query Q, a set of results R produced by Q, a UF $u()$, and the number of the preferred view recommendations k. Find the top-k views $\{v_1, v_2, ..., v_k\}$ constructed from R that have the highest utilities according to $u()$ among all possible views.

2.2. Context-Aware View Recommendation

The above definition of a typical view recommendation problem assumes that the UF $u()$ is defined a priori [17–20]. However, clearly such a predefined static UF cannot adapt to the different contexts encountered during different data analysis session. That is, the most suitable utility function at a time depends on the specific analysis context, with respect to the user, the data and the analysis task. For example, consider an initial data analysis when the user first explores the data. UFs recommending views that show interesting data characteristics such as skewed data distributions or significant attribute correlations, would be suitable to help the user get familiar with the data. However, for targeted analysis task, in which the user tries to find an explanation to a specific problem (e.g., a drop in sales),

then UFs recommending views that show peculiarities (e.g., views with uncommon trends, views with outliers, etc.) would be suitable to help the user discover potential explanations. Hence, it is clear that a predefined UF $u()$ is highly unlikely to always be the most suitable UF in all the different data analysis contexts.

Additionally, even for a single specific context, the interestingness of a view is hardly captured by a single UF, and is often determined by a combination of multiple UFs. For example, both data characteristics and visual quality affect the view interestingness at the same time. A view that may display interesting data patterns but is too cluttered to be easily understood would not be insightful. Neither would a view that is easy to understand but shows no interesting patterns. As such, in this work, we posit that a predefined single or composite UF $u()$ cannot always accurately capture the different aspects of the view interestingness in all analysis contexts.

In light of the above observations, we formalize the *Context-Aware View Recommendation (CoVR)* problem.

Definition 2. (*Context-Aware View Recommendation*) Given a database D, a user-specified query Q, a set of results R produced by Q, a set of n possible UFs $U = \{u_1(), u_2(), ..., u_n()\}$, and the number of the preferred view recommendations k. Find the ideal UF $u^*()$, which can be any combination of the functions in U and is most suitable in the current analysis context, and accordingly recommend the top-k views $\{v_1, v_2, ..., v_k\}$ constructed from R based on $u^*()$.

Since the number of possible UFs in U could be large, it is clearly more preferable and more effective to allow the ideal UF $u^*()$ to be discovered automatically by the system than hand-crafted by the user.

2.3. Interactive View Recommendation

In order to discover the ideal UF $u^*()$ in the CoVR problem, information about the analysis context needs to be collected accurately and quickly. In our work, we advocate for an *Interactive View Recommendation* paradigm where the system collects the context information and discovers the ideal UF $u^*()$ through *user interaction*, namely based on user answers to interactive questions.

Definition 3. (*Interactive View Recommendation*) Given a database D, a user-specified query Q, a set of results R produced by Q, a set of n possible UFs $U = \{u_1(), u_2(), ..., u_n()\}$, and the number of the preferred view recommendations k. Find the ideal UF $u^*()$, which can be any combination of the functions in U, interactively based on user feedback, and accordingly recommend the top-k views $\{v_1, v_2, ..., v_k\}$ constructed from R based on $u^*()$.

`ViewSeeker` [9–12] is our proposed realization of the IVR paradigm. `ViewSeeker` is a framework that uses machine learning techniques to discover the ideal UF $u^*()$ and supports two forms of adaptation: *UF Tuning* and *UF Integration*. *UF Integration* is the main functionality of `ViewSeeker`, which uses active learning to select example views for user labeling and learns the ideal UF $u^*()$ based on user feedback on the example views. As an IVR framework, `ViewSeeker` currently supports a variety of example view selection strategies,

user feedback types, and view interestingness prediction models, which can be chosen based on user analysis expertise, user data familiarity, etc.

UF Tuning is `ViewSeeker`'s second form of adaptation, which is useful when the most suitable parameters of a UF $u()$ vary greatly in different analysis contexts. During *UF Tuning*, `ViewSeeker` interacts with the user to tune the parameters of a UF $u()$ so that the parameters most accurately capture the data analysis context. In this paper, we focus on tuning the widely used *deviation-based* UFs [2, 21]. Deviation-based UFs adopt a *reference* parameter representing common scenarios for a view, and the view interestingness is measured by the distance between the view and the reference. A predefined reference is not likely to capture common scenarios for all views accurately in all analysis contexts. For instance, a reference in which gender ratio is at 1:1 is likely to capture the common scenario for gender ratio among kindergarten kids, but not kindergarten teachers. UF tuning can be used in such cases to properly set the reference, such that it most accurately captures the common scenarios for the views.

Clearly, IVR is a much more challenging problem than the traditional view recommendation problem, and `ViewSeeker` faces a search space much larger than the traditional view search space VS (Eq. 1). In particular, `ViewSeeker`'s search space includes in addition to VS, the search spaces of UF tuning and UF integration. In the following two sections, we discuss UF Tuning and UF Integration, which form the two phases of `ViewSeeker`, in detail.

3. ViewSeeker - Utility Function Tuning

As stated above, `ViewSeeker` addresses the CoVR problem in two phases, each of which provide a different level of adaptation. UF Tuning is the first and optional phase of `ViewSeeker`, in which the system interactively customize the individual UFs so that they most accurately capture the user's intention. `ViewSeeker` currently supports the tuning of the *deviation*-based UFs [2, 5, 7, 21], which are widely used UFs in view recommendation works. `ViewSeeker` tunes the deviation-based UFs by customizing the *reference* in the calculation of the UFs to improve the accuracy of their view interestingness estimation.

3.1. Deviation-based utility function

We first introduce how deviation may be measured based on a view. For clarity, we call each original view a target view v_i^T, which is represented as a triple (a, m, f) applied to a subset of the data D_Q that is produced by a given user query Q (as discussed in Section 2.1). In order to define the deviation, we create a helper view called the reference view v_i^R for each target view, which represents common scenarios for the target view. One way to create the reference view is to visualize the results of grouping the data in the whole database D with the same set of triple (a, m, f) used by the target view. An example of a target view with its reference view under this approach is illustrated in Figure 1. The target view on the right (black) shows the player 3-point attempt rate of a selected NBA team and the reference view on the left (gray) shows the player 3-point attempt rate of all teams in the league in a particular year [22]. The comparison of the two views shows that the selected

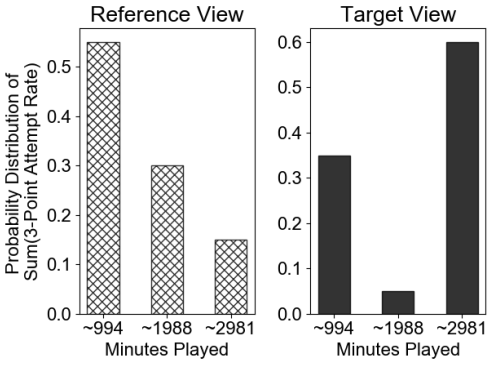


Figure 1: A target view and its corresponding reference view.

team outperformed the league average and could explain why it won the championship [5, 6]. The difference between the two views can be expressed in term of *deviation*.

Deviation measures the difference between the target view and the reference view with an underlying assumption that the greater the difference, the higher the utility is. In the case of histograms or bar charts, measuring the difference between two views v_i^T and v_i^R essentially equals measuring the distance between the two normalized probability distributions $P(v_i^T)$ and $P(v_i^R)$. The conversion from a view to a probability distribution is illustrated in Eq. 2. Specifically, we normalize each view v_i by individually dividing the aggregate value of each bin in v_i by the sum of the aggregated values of all bins in v_i, such that the sum of aggregated values of all bins in v_i would become 1.

$$P(v_i) = \langle \frac{g_1}{G}, \frac{g_2}{G}, ..., \frac{g_b}{G} \rangle \tag{2}$$

where $P(v_i)$ is the probability distribution after normalization; g_i are individual values in each bin; $G = \sum_{i=1}^{b} g_i$ is the sum of the values in all bins; and b is the number of bins in the dimension attribute a.

After normalizing v_i^T and v_i^R, the utility score $u(v_i)$ of a view v_i computed from deviation can be defined as:

$$u(v_i) = DT(P(v_i^T), P(v_i^R)) \tag{3}$$

where DT is the distance function that measures the distance between two distributions (e.g., Euclidean, Earth Movers Distance).

3.2. Reference View Generation

For a target view v_i^T, there are two commonly used approaches to generate a reference view v_i^R that represents common scenarios for v_i^T, which we refer to as the *Global* approach and the *Local* approach.

The Global approach relies on the global information (i.e., the whole dataset D) to generate the reference view. As illustrated in the NBA example above (Figure 1), for a target view v_i^T, the reference view v_i^R under this approach is generated by applying the aggregate query (i.e., the triple (a, m, f)) of v_i^T on the whole dataset D. The Global approach is a

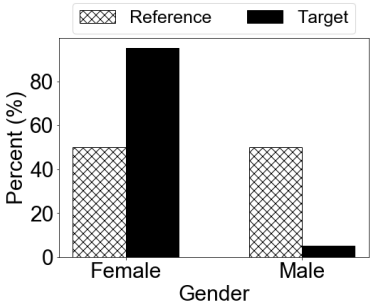


Figure 2: A view of kindergarten teacher gender ratios.

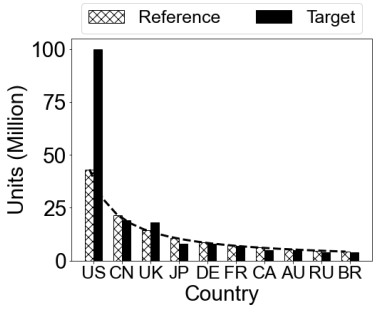


Figure 3: A view of unit sales by country.

widely used data-driven approach. The state-of-the-art view recommendation work SeeDB [2] adopts the Global approach as its default reference specification.

The Local approach relies on the local information (i.e., the target view v_i^T) to generate the reference view. For a target view v_i^T, the Local approach first sorts the aggregate values in v_i^T in descending order to form a value sequence $X = \{x_1, x_2, ..., x_n\}$. Then it fits a descending power-law line (i.e., exponential function) to X, and uses the fitted line as the reference view for v_i^T. Figure 3 illustrates an example of the reference view v_i^R (the dashed line) generated by the Local approach on the values of the unit sales by country in v_i^T. The state-of-the-art view recommendation work QuickInsights [16, 21, 23] adopts the Local approach as its reference specification for bar charts.

Since the reference views in both approaches are predefined, they cannot always accurately capture the common scenarios for the target view, causing inaccurate reference view generation, and in turn inaccurate view interestingness estimation. We use the below two example to illustrate the issue.

For the Global approach, consider the case in Figure 2. The target view v_i^T shows the gender ratio of kindergarten teachers D_Q, and the reference view v_i^R shows the gender ratio of the whole dataset D. The Global approach would recommend this view because there is a large deviation between v_i^T and v_i^R. However, since it is well known that currently female teachers constitute a dominant proportion of kindergarten teachers, this view could very likely be uninteresting to the user. On the contrary, the Global approach would not recommend this view if the gender ratio for kindergarten teachers is close to 1:1, because the deviation between v_i^T and v_i^R would be very small in this case. However, such a view would very likely interest the user because it contradicts the user's prior knowledge.

For the Local approach, consider again the case inspired by [23] in Figure 3. The target view v_i^T shows the unit sales by country, and the reference view v_i^R is generated by the fitted power-law line (the dashed line) on the values in v_i^T. The Local approach would recommend this view because there is a large deviation between v_i^T and v_i^R. However, if the user is well aware of the large sales gap between US and the other countries, this view could become uninteresting to the user. On the contrary, the Local approach would not recommend this view if the US has a lower unit sales (e.g., 50M), because the deviation between v_i^T and v_i^R would be small in this case. However, such a view would very likely interest the user because

the user would expect a larger sales gap based on their prior knowledge.

From the above two examples, we see that the predefined references in the Global and Local approaches may not always accurately capture the common scenarios for the target views. Therefore, we propose a novel approach called *Expectation Acquisition and Propagation* (EAP), which leverages user domain knowledge to generate reference views v_i^R that capture more accurately the common scenarios for the target views v_i^T.

3.3. Expectation Acquisition & Propagation

In order to generate reference views v_i^R that capture more accurately user prior knowledge (i.e., expectation) for target views v_i^T, we propose the two-step *Expectation Acquisition & Propagation* (EAP) algorithm.

In the Expectation Acquisition (EA) step, `ViewSeeker` displays several example target views to the user, and asks the user to specify their expectation for the target views. The user can specify their expectation by providing the expected aggregated values for the groups directly on each target view. Ideally, the user should be presented with each target view, and asked to specify their expectation. However, such an EA-only approach is obviously not scalable. Therefore, we propose the second Expectation Propagation (EP) step to reduce user labeling effort. Specifically, in the EP step, `ViewSeeker` generates user expectation estimates (i.e., reference views) v_i^R for the unlabeled target views v_i^T based on the specified expectations in the EA step and recommends the v_i^T's that deviate largely from the corresponding v_i^R's.

The hypothesis under EAP is that the attribute correlations in user domain knowledge about D_Q (i.e., the data subset) is similar to the attribute correlations in D_Q, therefore EAP can use the attribute correlations in D_Q and user expectation specified on a target view to estimate user expectation for other target views.

In the following subsections, we will use an example data subset to illustrate the EA step and the EP step, which can be further divided into the EPaD process (Expectation Propagation across Dimensions) and the EPaM process (Expectation Propagation across Measures). In describing these steps, we will use the following notation. Recall that each target view is the result of applying a triple (a, m, f) on the query subset D_Q. In order to distinguish between different target views, we replace the subscript i with the actual triple for v_i^T, such that the v_i^T with the triple (a, m, f) is now noted as $v_{a,m,f}^T$, and the corresponding reference view is now noted as $v_{a,m,f}^R$. Further, we use the superscript R for any reference view that is either specified directly by the user or estimated by the EAP algorithm, and the superscript T for any view that is generated directly from the query subset D_Q.

3.3.1. Expectation Acquisition

Consider an example data subset D_Q from certain labor force survey as show in Table 1. The data subset has two dimensions a_1 (Education/Edu) and a_2 (Occupation/Occ), two measures m_1 (Wage Income/WI) and m_2 (Total Income/TI), and contains six records.

Assume that EA selects a target view $v_{a_1,m_1,AVG}^T$ ($v_{Edu,WI,AVG}^T$ shown in Table 2) for user expectation labeling. (The example selection strategy will be discussed in Section 3.3.4). Further, assume that the user indicates that this view is interesting because the user is

Table 1: Example Data Subset D_Q

Education	Occupation	Wage Income	Total Income
Bachelor	Engineer	190	250
Bachelor	Engineer	210	270
Master	Engineer	300	360
Master	Scientist	100	120
PhD	Scientist	140	155
PhD	Scientist	160	175

Table 2: $v^T_{Edu,WI,AVG}$

Education	AVG(Wage Income)
Bachelor	200
Master	200
PhD	150

Table 3: $v^R_{Edu,WI,AVG}$

Education	AVG(Wage Income)
Bachelor	200
Master	200
PhD	300

surprised that PhD holders have a lower wage income than that of Bachelor and Master degree holders. The user also indicates their expectation that PhD holders should have a wage income of at least 300. In other words, the EA step has elicited the user's expectation for the target view $v^R_{a_1,m_1,AVG}$ ($v^R_{Edu,WI,AVG}$) as shown in Table 3.

3.3.2. EP Across Dimensions (EPaD)

Recall that the goal of EP is to use the user expectation specified in the EA step (e.g., $v^R_{a_1,m_1,AVG}$) to generate user expectation estimate (i.e., reference views) v^R_i for other target views v^T_i, so that target views v^T_i that deviate largely from the corresponding reference views v^R_i would be recommended. EPaD assumes that correlation between dimension attributes in user domain knowledge is similar to the corresponding correlation in the data subset, such that EP could use the latter and the specified user expectation on certain target views to estimate user expectation for other target views.

The overall workflow of EPaD, for example from a_1 (Education) to a_2 (Occupation) can be divided in three steps. If we assume that EP needs to generate user expectation estimate $v^R_{a_2,m_1,AVG}$ ($v^R_{Occ,WI,AVG}$) from $v^R_{a_1,m_1,AVG}$, the reference view generation steps are:

$$v^R_{a_1,m_1,AVG} \xrightarrow{\text{Step 1}} v^R_{a_1,m_1,SUM} \xrightarrow{\text{Step 2}} v^R_{a_2,m_1,SUM} \xrightarrow{\text{Step 3}} v^R_{a_2,m_1,AVG}. \tag{4}$$

The intuition of the need of the intermediate transformation to $v^R_{a_1,m_1,SUM}$ is because the number of records in each group of a_1 represents the impact of this group on downstream views in the path, such that a larger group will have a larger impact. However, this impact information is not reflected by the average aggregate view. So EP needs to multiply the average view by the corresponding counts of the groups to form a sum view to capture the impact of the groups. The details of each estimation step are as follows:

Step 1 ($v^R_{a_1,m_1,SUM}$ generation) The specified user expectation (i.e., reference view) can be represented as a vector as $v^R_{a_1,m_1,AVG} = (g_1, g_2, ..., g_p)$, where p is the number of groups in the

dimension a_1. In the example dataset, $v^R_{a_1,m_1,AVG} = v^R_{Edu,WI,AVG} = (200, 200, 300)$. Then, EP first generates $v^R_{a_1,m_1,SUM}$ using Eq. 5.

$$v^R_{a_1,m_1,SUM} = v^R_{a_1,m_1,AVG} \odot v^T_{a_1,m_1,COUNT} = (g_1c_1, g_2c_2, ..., g_pc_p) \tag{5}$$

where $v^T_{a_1,m_1,COUNT} = (c_1, c_2, ..., c_p)$ is a helper view which contains the counts for each group in a_1, and $\odot$ is the element-wise multiplication operator. In the example dataset, $v^T_{a_1,m_1,COUNT} = v^T_{Edu,WI,COUNT} = (2, 2, 2)$, so the corresponding calculation is:

$$v^R_{Edu,WI,SUM} = v^R_{Edu,WI,AVG} \odot v^T_{Edu,WI,COUNT} = (200, 200, 300) \odot (2, 2, 2) = (400, 400, 600)$$

The corresponding result is shown in Table 4.

Step 2 ($v^R_{a_2,m_1,SUM}$ generation) In order to generate $v^R_{a_2,m_1,SUM}$, EP would create a helper view on D_Q with two grouping attributes a_1 and a_2, which can be represented as a matrix as shown in Eq. 6.

$$V^T_{a_1,a_2,m_1,SUM} = \begin{bmatrix} s_{11} & \dots & s_{1q} \\ \vdots & \ddots & \vdots \\ s_{p1} & \dots & s_{pq} \end{bmatrix} \tag{6}$$

where p and q are the group numbers for dimension a_1 and a_2 respectively and s_{pq} is the aggregate result for the 2-attribute group pq. The corresponding matrix in the example dataset is:

$$V^T_{Edu,Occ,WI,SUM} = \begin{bmatrix} 400 & 0 \\ 300 & 100 \\ 0 & 300 \end{bmatrix}$$

Then, EP would normalize each row in $V^T_{a_1,a_2,m_1,SUM}$ to get $V^T_{a_1,a_2,m_1,SUM,normed}$ as shown in Eq.7.

$$V^T_{a_1,a_2,m_1,SUM,normed} = \begin{bmatrix} \frac{s_{11}}{G_1} & \dots & \frac{s_{1q}}{G_1} \\ \vdots & \ddots & \vdots \\ \frac{s_{p1}}{G_p} & \dots & \frac{s_{pq}}{G_p} \end{bmatrix} \tag{7}$$

where $G_i = \sum_{j=1}^{q} s_{iq}$ for $i = 1...p$. The corresponding matrix for the example dataset is:

$$V^T_{Edu,Occ,WI,SUM,normed} = \begin{bmatrix} 400/400 & 0/400 \\ 300/400 & 100/400 \\ 0/300 & 300/300 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 \\ 0.75 & 0.25 \\ 0.0 & 1.0 \end{bmatrix}$$

Then, EP can generate $v^R_{a_2,m_1,SUM}$ using Eq. 8.

$$v^R_{a_2,m_1,SUM} = v^R_{a_1,m_1,SUM} \cdot V^T_{a_1,a_2,m_1,SUM,normed} = (g_1c_1, g_2c_2, ..., g_pc_p) \cdot \begin{bmatrix} \frac{s_{11}}{G_1} & \dots & \frac{s_{1q}}{G_1} \\ \vdots & \ddots & \vdots \\ \frac{s_{p1}}{G_p} & \dots & \frac{s_{pq}}{G_p} \end{bmatrix} = (t_1, t_2, ..., t_q) \tag{8}$$

Table 4: $v^R_{Edu,WI,SUM}$

Education	SUM(Wage Income)
Bachelor	400
Master	400
PhD	600

Table 5: $v^R_{Occ,WI,SUM}$

Occupation	SUM(Wage Income)
Engineer	700
Scientist	700

Table 6: $v^R_{Occ,WI,AVG}$

Occupation	AVG(Wage Income)
Engineer	233
Scientist	233

Table 7: $v^T_{Occ,WI,AVG}$

Occupation	AVG(Wage Income)
Engineer	233
Scientist	133

The corresponding calculation for the example dataset is:

$$v^R_{Occ,WI,SUM} = v^R_{Edu,WI,SUM} \cdot V^T_{Edu,Occ,WI,SUM,normed} = (400, 400, 600) \cdot \begin{bmatrix} 1.0 & 0.0 \\ 0.75 & 0.25 \\ 0.0 & 1.0 \end{bmatrix} = (700, 700)$$

In other words, EP splits the income sum in each group in a_1 (Education) and sums up the splits by the groups in a_2 (Occupation). The corresponding result is shown in Table 5.

Step 3 ($v^R_{a_2,m_1,AVG}$ generation) Finally, EP generates $v^R_{a_2,m_1,AVG}$ using Eq. 9.

$$v^R_{a_2,m_1,AVG} = v^R_{a_2,m_1,SUM} \oslash v^T_{a_2,m_1,COUNT} = (t_1, t_2, ..., t_q) \oslash (c_1, c_2, ..., c_q) = (\frac{t_1}{c_1}, \frac{t_2}{c_2}, ..., \frac{t_q}{c_q}) \quad (9)$$

where $v^T_{a_2,m_1,COUNT} = (c_1, c_2, ..., c_q)$ is another helper view, which contains tuple counts for the groups in a_2, and $\oslash$ is the element-wise division operator. For the example dataset, $v^T_{a_2,m_1,COUNT} = v^T_{Occ,WI,COUNT} = (3, 3)$ and the corresponding calculation is:

$$v^R_{Occ,WI,AVG} = v^R_{Occ,WI,SUM} \oslash v^T_{Occ,WI,COUNT} = (700, 700) \oslash (3, 3) = (233, 233)$$

The result is shown in Table 6. The corresponding target view $v^T_{Occ,WI,AVG}$ generated directly from the dataset is shown in Table 7. It can be seen that there is a large deviation between the estimated user expectation for scientist wage income (i.e., 233) and the actual value (i.e., 133), which means that the target $v^T_{Occ,WI,AVG}$ could be interesting to the user, the user could expect the scientist to have a higher wage income.

EP for views with aggregate function SUM and COUNT works similar as above with aggregate function AVG. EP between views with the aggregate function SUM can be performed using Eq. 8, with the difference that $v^R_{a_1,m_1,SUM}$ is directly specified by the user in this case. EP between views with the aggregate function COUNT can be performed by changing the aggregate functions in the partial process from $v^R_{a_1,m_1,SUM}$ to $v^R_{a_2,m_1,SUM}$ above (i.e., Eq. 6 - 8) to COUNT, such that the EP proceeds as shown in Eq. 10.

$$v^R_{a_2,m_1,COUNT} = v^R_{a_1,m_1,COUNT} \cdot V^T_{a_1,a_2,m_1,COUNT,normed} \quad (10)$$

where $v^R_{a_1,m_1,COUNT}$ is specified directly by the user.

To summarize, EPaD (EP across dimensions) leverages the correlation between dimension attributes to propagate specified expectation on certain target view to other target views, such that the target views that deviate largely from the corresponding expectation estimates (i.e., reference views) would be recommended.

3.3.3. EP Across Measures (EPaM)

In this section, we will introduce the EP across measures (EPaM) process. EPaM assumes that correlation between measure attributes in user domain knowledge is similar to the corresponding correlation in the data subset, such that EP could use the latter and the specified user expectation on certain target views to estimate user expectation for other target views.

Consider, same as in EPaD, EA has elicited user expectation $v^R_{a_1,m_1,AVG}$ ($v^R_{Edu,WI,AVG}$), and EP needs to use it to generate user expectation estimate $v^R_{a_1,m_2,AVG}$ ($v^R_{Edu,TI,AVG}$). This is EP across dimensions from m_1 (Wage Income) to m_2 (Total Income).

The overall EP process is described in Eq. 11.

$$v^R_{a_1,m_2,AVG} = v^T_{a_1,m_2,AVG} \oplus (v^R_{a_1,m_1,AVG} \ominus v^T_{a_1,m_1,AVG}) \oslash \sigma^T_{a_1,m_1} \odot r^T_{a_1,m_1,m_2} \odot \sigma^T_{a_1,m_2} \tag{11}$$

where $v^T_{a_1,m_1,AVG}$ and $v^T_{a_1,m_2,AVG}$ are the target views, $\oplus$ and $\ominus$ represent element-wise addition and subtraction respectively, $\sigma^T_{a_1,m_1}$ contains the standard deviations of m_1 for each group in a_1, $\sigma^T_{a_1,m_2}$ contains the standard deviations of m_2 for each group in a_1, and $r^T_{a_1,m_1,m_2}$ contains the Pearson correlation coefficients between m_1 and m_2 for each group in a_1.

It can be seen from Eq. 11 that the EP process proceeds in the same way for each group in a_1, so for simplicity, we will introduce the EP process for an example group g_1 (PhD) in a_1 (Education). Therefore, Eq. 11 becomes:

$$v^R_{g_1,m_2,AVG} = v^T_{g_1,m_2,AVG} + (v^R_{g_1,m_1,AVG} - v^T_{g_1,m_1,AVG})/\sigma^T_{g_1,m_1} * r^T_{g_1,m_1,m_2} * \sigma^T_{g_1,m_2} \tag{12}$$

where subscript g_1 represents the group.

The corresponding calculation for the example dataset is:

$$v^R_{PhD,TI,AVG} = v^T_{PhD,TI,AVG} + (v^R_{PhD,WI,AVG} - v^T_{PhD,WI,AVG})/\sigma^T_{PhD,WI} * r^T_{PhD,WI,TI} * \sigma^T_{PhD,TI}$$

We discuss the calculation step by step in the follow. First, EP starts with:

$$\Delta^R_{PhD,TI,AVG} = (v^R_{PhD,WI,AVG} - v^T_{PhD,WI,AVG})/\sigma^T_{PhD,WI} = (300 - 150)/10 = 15$$

where $v^R_{PhD,WI,AVG}$ and $v^T_{PhD,WI,AVG}$ are the third rows in Table 3 and 2 respectively. $\sigma^T_{PhD,WI}$ is the standard deviation for wage income of PhD in Table 1.

Then EP proceeds with:

$$\Delta^R_{PhD,TI,AVG} = \Delta^R_{PhD,WI,AVG} * r^T_{PhD,WI,TI} = 15 * 1.0 = 15$$

where $r^T_{PhD,WI,TI}$ is the Pearson correlation coefficient between WI and TI for PhD in Table 1.

Table 8: $v^R_{Edu,TI,AVG}$

Education	AVG(Total Income)
Bachelor	260
Master	240
PhD	315

Table 9: $v^T_{Edu,TI,AVG}$

Education	AVG(Total Income)
Bachelor	260
Master	240
PhD	165

Finally, EP finishes with:

$$v^R_{PhD,TI,AVG} = v^T_{PhD,TI,AVG} + \Delta^R_{PhD,TI,AVG} * \sigma^T_{PhD,TI} = 165 + 15 * 10 = 315$$

where $v^T_{PhD,TI,AVG}$ and $\sigma^T_{PhD,TI}$ are the mean and standard deviation for total income of PhD in Table 1.

In other words, the EP process first calculates the difference between $v^R_{PhD,WI,AVG}$ and $v^T_{PhD,WI,AVG}$, then propagates the difference to another measure TI based on correlation between WI and TI (i.e., $r^T_{PhD,WI,TI}$), finally adds the estimated difference to $v^T_{PhD,TI,AVG}$ to form expectation estimate $v^R_{PhD,TI,AVG}$ for TI. The standard deviations $\sigma^T_{PhD,WI}$ and $\sigma^T_{PhD,TI}$ are in place to remove the influence of the scale differences between the two measures.

After estimating user expectation for other education groups, EP would generate the reference view $v^R_{Edu,TI,AVG}$ as shown in Table 8, and the corresponding target view $v^T_{Edu,TI,AVG}$ is shown in Table 9. We see that the average total income for PhD in $v^T_{Edu,TI,AVG}$ (i.e., 165) is much lower than that in the user expectation estimate $v^R_{Edu,TI,AVG}$ (i.e., 315), indicating that this view might be interesting to the user because the user could expect a higher total income for PhD.

EP between views with the aggregate function SUM can be done in a similar fashion as shown in Eq. 13.

$$v^R_{a_1,m_2,SUM} = v^T_{a_1,m_2,SUM} \oplus (v^R_{a_1,m_1,SUM} \ominus v^T_{a_1,m_1,SUM}) \oslash \sigma^T_{a_1,m_1} \odot r^T_{a_1,m_1,m_2} \odot \sigma^T_{a_1,m_2} \quad (13)$$

where $v^R_{a_1,m_1,SUM}$ is specified by the user.

EP between views with the aggregate function COUNT directly copies the user expectation, as shown in Eq. 14.

$$v^R_{a_1,m_2,COUNT} = v^R_{a_1,m_1,COUNT} \quad (14)$$

where $v^R_{a_1,m_1,COUNT}$ is specified by the user.

To summarize, EPaM (EP across measures) leverages the correlation between measure attributes to propagate specified expectation on certain target view to other target views, such that the target views that deviate largely from the corresponding expectation estimates (i.e., reference views) would be recommended.

3.3.4. Example Selection Strategy

As discussed, Expectation Propagation (EP) can be done across both dimensions and measures, which means that ideally, EP can estimate user expectations for all the target views by asking the user to label one target view for each aggregate function. Therefore, the

Algorithm 1 `ViewSeeker` - Deviation-based Utility Function Tuning

Require: A data subset D_Q specified by a query
Ensure: Customized deviation-based utility functions $C = \{u_1(), u_2(), ..., u_n()\}$
1: Target view set $T \leftarrow generateViews(D_Q)$
2: Reference view set $R \leftarrow \{\}$
3: **for all** $f \in \{COUNT, SUM, AVG\}$ **do**
4: Choose one example $v^T_{a,m,f}$ from T for expectation labeling to get $v^R_{a,m,f}$
5: $R \leftarrow R \cup \{v^R_{a,m,f}\}$
6: **for all** $a' \in A$ **do**
7: **if** $a' \neq a$ **then**
8: Expectation propagation from $v^R_{a,m,f}$ to $v^R_{a',m,f}$
9: $R \leftarrow R \cup \{v^R_{a',m,f}\}$
10: **end if**
11: **for all** $m' \in M$ **do**
12: **if** $m' \neq m$ **then**
13: Expectation propagation from $v^R_{a',m,f}$ to $v^R_{a',m',f}$
14: $R \leftarrow R \cup \{v^R_{a',m',f}\}$
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: $C \leftarrow generateCustomizedFunctions(T, R)$
20: Return C

strategy to select the example view for labeling in the Expectation Acquisition (EA) step is critical to the effectiveness of the EAP algorithm. In our work, we have adopted a simple yet effective example selection strategy. Specifically, for each aggregate function, EA chooses the view with the dimension attribute that has the largest group count among all dimension attributes as the example view under the assumption that a target view with more groups provides more useful information about user expectation. The example selection strategy is captured in Eq. 15.

$$a_{selected} = \underset{a}{\text{argmax}}(|a|) \tag{15}$$

where $|\cdot|$ represents the group count operator. If multiple views have the dimension attribute with the largest group count, EP selects the first view based on view attribute order in the table.

3.4. Runtime Complexity of Deviation-based UF Tuning

Algorithm 1 shows `ViewSeeker`'s four steps to tune deviation-based utility functions: i) Generation of all target views based on D_Q (Line 1); ii) Selection of an example target view for each aggregate function and elicitation of user expectations on the view (Line 3-5); iii) Generation of user expectation estimate for other target views (Line 6-18); and iv) Formulation of the customized deviation-based UFs $C = \{u_1(), u_2(), ..., u_n()\}$ (Line 19-20). The customized deviation-based UFs C are passed to `ViewSeeker`'s second phase to use them to calculate the deviation scores for each target view v^T_i based on the difference between v^T_i and the corresponding reference view v^R_i.

Table 10: UF Tuning Time Complexity

Part #	Part Name	# Operations
1	View generation	$AMFN_Q$
2	Helper view/matrix generation	$AAMFN_Q$
3	EP across dimension attributes	$AFGG$
4	EP across measure attributes	$AMFG$

To analyze the runtime complexity of deviation-based UF Tuning phase of `ViewSeeker`, we have divided the EAP process into four parts and evaluated the time complexity in terms of number of operations for each part. Our analysis was based on Algorithm 1 and the EAP equations, and the cost for each part is summarized in Table 10. The meanings of the symbols in the table are: A is the number of dimension attributes, M is the number of measure attributes, F is the number of aggregate functions, N_Q is the number of records in the query subset D_Q, G is the average number of groups in each dimension attribute.

For the *view generation* part, `ViewSeeker` needs to scan the query subset to generate a view for each (a, m, f) triple, which results in a complexity of $AMFN_Q$. For *helper view/matrix generation*, the dominating part is the generation of the helper matrices. Since each matrix has two dimension attributes, the total number of matrices would be $AAMF$, thus the complexity is $AAMFN_Q$. *EP across dimensions* needs to propagate user expectation to AF views, and EP to each view has a complexity of GG, resulting in a total complexity of $AFGG$. *EP across measures* needs to propagate user expectation to AMF views, and EP to each view has a complexity of G, resulting in a total complexity of $AMFG$.

We compare the theoretical complexities of the above parts to the experimentally measured runtime costs in Section 5.

3.5. Tuning of Other Utility Functions

Although the idea of tuning the UF based on analysis context is the same for all the UFs, the specific ways to tune the UFs are different. EAP is suitable for the tuning of the UFs whose calculation involves a reference representing common scenarios for the target view, for example, the deviation-based UFs [2, 21].

Other UFs evaluating other aspects of the view interestingness may require other methods for tuning. For example, UFs measuring *visual quality* of the view [8, 17] may incorporate context information such as user visual literacy to improve accuracy. Tuning of other UFs is part of our future work.

4. ViewSeeker - Utility Function Integration & View Recommendation

UF Integration is the second phase of `ViewSeeker`, in which the system leverages user interaction to *integrate* the various UFs, including the customized ones in the previous phase, to discover the ideal utility function $u^*()$ that is most suitable in the current analysis context.

Algorithm 2 shows the overall workflow of `ViewSeeker`. The UF Tuning phase (Line 1) is optional and occurs first if the user is interested in generating customized UFs, specifically

Algorithm 2 `ViewSeeker`

Require: The data subset D_Q specified by a query
Ensure: The view interestingness estimator IE
1: Customized utility functions $C \leftarrow tuneUtilityFunctions(D_Q)$
2: Unlabeled example set $U \leftarrow generateExamples(D_Q, C)$
3: Labeled example set $L \leftarrow$ obtain initial set of example labels
4: $IE \leftarrow$ initialize view interestingness estimator IE using L
5: $FE \leftarrow$ initialize example informativeness estimator FE using L
6: **loop**
7: Choose one x from U using FE
8: Solicit user's label on x
9: $L \leftarrow L \cup \{x\}$
10: $U \leftarrow U - \{x\}$
11: $IE \leftarrow$ refine IE using L
12: $FE \leftarrow$ refine FE using L
13: $T \leftarrow$ recommend top views using IE
14: **if** the user is satisfied with T or the user wants to stop **then**
15: Break
16: **end if**
17: **end loop**
18: Return the most recent IE

the currently supported customized deviation-based UFs. The UF Integration phase follows the UF Tuning phase and has three main stages.

Stage 1: *View Generation*, in which `ViewSeeker` generates the unlabeled example set (Line 2). The customized UFs are used in this stage to generate the internal learning representation of the examples.

Stage 2: *Initial Example Acquisition*, in which `ViewSeeker` acquires user labels on some examples to initialize the *view interestingness estimator (IE)* and the *example informativeness estimator (FE)* (Line 3-5). The concept of the *informativeness* of an example is adopted from *active learning* [13] to refer to the benefit of the label on an example to the improvement of IE, such that the most informative example could be selected for user labeling.

Stage 3: *Interactive View Recommendation*, in which `ViewSeeker` interacts with the user to refine the estimators and provides view recommendation based on IE predictions (Line 6-18). We elaborate on the three stages of the UF Integration phase next.

4.1. View Generation

In this first stage, `ViewSeeker` generates the unlabeled example set U, unless the target view set T from the UF Tuning phase is available, in which case `ViewSeeker` reuses it as U.

`ViewSeeker` also generates an internal representation for each view as the learning representation for the following stages. The features in the internal representation for a view are the UF scores. For the UFs that `ViewSeeker`'s UF Tuning currently does not support, `ViewSeeker` uses their default calculation methods to generate the scores without tuning. If customized deviation-based UFs were generated by the UF Tuning phase, ViewSeeker uses the customized UFs to calculate the UF scores.

Table 11: Technology Summary for UF Integration Phase

Example form	Feedback type	IE model	FE strategy	IE ranking score
Single view	Binary	Binary classifier	Least confident	$P_{IE}(y = positive\|v)$
Single view	Likert-scale	Multi-class classifier	Least confident	$\sum_{i=1}^{C}(w_i \times P_{IE}(y = c_i\|v))$
Single view	Real number	Regression model	QBC	$s_{IE}(v)$
View pair	Pairwise comparison	Learning-to-rank	QBC	$s_{IE}(v)$

4.2. Initial Example Acquisition

In the second stage, `ViewSeeker` would acquire an initial set of user feedback on several randomly selected examples to initialize the view *interestingness estimator* IE and the *example informativeness estimator* FE. `ViewSeeker` currently supports two example forms and four types of user feedback. Both example form and feedback type can be selected based on user analysis expertise and user data familiarity.

The first example form displays a single view at a time for labeling, while the second example form displays a pair of views at a time for labeling.

For the example form of a single view, `ViewSeeker` offers three feedback types: *binary, Likert-scale,* and *real number*. The binary feedback asks the user to indicate if the example view is interesting or not. The Likert-scale feedback asks the user to label the example view on a scale of 1 to 5, with 5 being most interesting and 1 being least interesting. The real number feedback requires the user to label the example view with a real number between 0.0 and 1.0, with 1.0 being most interesting and 0.0 being least interesting.

For the example form of a pair of views, the supported feedback type is pairwise comparison. Given a pair of example views, the pairwise comparison feedback asks the user to indicate if the first view is more interesting or less interesting than the second view.

After the user labeling to some examples, `ViewSeeker` use the labels to initialize IE and FE.

4.3. Interactive View Recommendation

In the third stage, `ViewSeeker` interacts with the user in an iterative fashion to refine the IE and uses it to provide view recommendation. It can be seen from Algorithm 2 that there are four steps in each iteration: 1) Example Selection: `ViewSeeker` selects examples from U based on the informativeness estimate from FE (Line 7). 2) User Label Acquisition: `ViewSeeker` acquires user labels on the examples (Line 8). 3) Estimator Refinement: `ViewSeeker` uses the labeled example set L to refine IE and FE (Line 9-12). 4) View Recommendation: `ViewSeeker` uses the latest IE to make view recommendations (Line 13-16). If the user is satisfied with the view recommendation in Step 4, then the loop stops; otherwise, the loop starts from Step 1 again.

Each feedback type requires a different set of methods and technologies for IE model, FE strategy, and IE ranking algorithm. These are summarized in Table 11 and discussed below.

4.3.1. Example Selection

`ViewSeeker` uses FE for example selection in Step 1. The example informativeness prediction of FE is based on the view interestingness prediction of IE. The view interestingness estimator IE is in the form of a machine learning model, and `ViewSeeker` uses different model types for different feedback types.

For binary feedback, the IE is in the form of a binary classifier, such as the logistic regression model. For Likert-scale feedback, the IE is in the form of a multi-class classifier, such as the decision tree model. For real number feedback, the IE is in the form of a regression model, such as the linear regression model. For pairwise comparison feedback, the IE is in the form of a learning-to-rank model [24].

The example informativeness estimator FE is essentially a strategy to utilize the prediction from IE (or a committee of IEs) to estimate the informativeness of the unlabeled examples, such that the most informative example could be selected for user labeling. This strategy is named *query strategy* in active learning literature [13]. `ViewSeeker` again uses different query strategies for different feedback types.

For binary and Likert-scale feedback, `ViewSeeker` supports any query strategies that are suitable for classification problem, such as the *uncertainty sampling* strategy [13]. For real number feedback, `ViewSeeker` supports any query strategies that are suitable for regression problem, such as the *query-by-committee* strategy [13]. For pairwise comparison feedback, `ViewSeeker` supports any query strategies that are suitable for the learning-to-rank problem. The following are the default example query strategy for each feedback type.

An example query strategy for binary or Likert-scale feedback is the *least confident* strategy under the category of uncertainty sampling strategies. The least confident strategy selects the view whose class label the IE is least confident about, according to Eq. 16.

$$v^*_{LC} = \underset{v}{\operatorname{argmax}}\, 1 - P_{IE}(\hat{y}|v) \tag{16}$$

where v^*_{LC} is the example to select, and $\hat{y} = argmax_y P_{IE}(y|v)$ is the class label with the highest posterior probability under the model IE. In other words, the most informative example for a binary classifier is the example whose positive class posterior probability is closest to 0.5. And the most informative example for a multi-class classifier is the example whose highest posterior probability in any class is the lowest.

An example query strategy for the real number feedback is the *query-by-committee strategy* (QBC). QBC builds a committee of IEs, and estimates the informativeness of the example based on the disagreement among the committee members. The members of the committee are built with slightly different hyper-parameters or trained with slightly different training sets, such that they would provide different predictions for the same example.

Recall that for real number feedback, the IE is in the form of a regression model. Therefore the committee for real number feedback is a committee of regression models. Each regression model in the committee c_i can be trained with the labeled set L and predict a score $s_{c_i}(v)$ for a view v. One way to measure the disagreement among the committee members is to measure the variance of their predictions, as shown in Eq. 17.

$$v^*_{QBC} = \underset{v}{\operatorname{argmax}}\, Var(\langle s_{c_1}(v), s_{c_2}(v), ..., s_{c_C}(v)\rangle) \tag{17}$$

where $Var(\cdot)$ is the variance operator, and C is the number of committee members.

QBC can also be used for the pairwise comparison feedback as proposed by our previous work [12]. Each learning-to-rank model in the committee for the pairwise comparison feedback c_i can be trained with the ordered view pairs in the labeled set L and predict a ranking score $s_{c_i}(v)$ for a view v. One way to measure the disagreement among the committee members for an example (i.e., a pair of view (v_1, v_2)) is to measure the variance of their predictions for the ranking score gap between the two views (i.e., $s_{c_i}(v_1) - s_{c_i}(v_2)$) as shown in Eq. 18.

$$(v_1, v_2)^*_{QBC} = \underset{(v_1,v_2)}{\operatorname{argmax}} Var(\langle s_{c_1}(v_1) - s_{c_1}(v_2), s_{c_2}(v_1) - s_{c_2}(v_2), ..., s_{c_C}(v_1) - s_{c_C}(v_2)\rangle) \quad (18)$$

4.3.2. Estimator Refinement

After the example selection and the following user label acquisition for the examples, `ViewSeeker` would use the latest labeled example set L to refine the IE and FE. Specifically, `ViewSeeker` would use the vector representation of the labeled views as the input features and the user labels as the input labels to train a new IE and a new FE. Note that, for the QBC query strategy, `ViewSeeker` would usually keep a separate IE as the formal IE for view recommendation outside the FE, which is in the form of a committee of IEs.

4.3.3. View Recommendation

The last step in each user interaction loop is view recommendation. The interestingness estimator IE uses different methods to generate view interestingness estimate for different feedback types.

For binary feedback, IE uses the positive class probability $P_{IE}(y = positive|v)$ as the interestingness estimate for a view v because it reflects the classifier's belief of v's interestingness.

For Likert-scale feedback, IE uses the weighted sum of the weight and probability of each class as the interestingness estimate for a view v as shown in Eq. 19:

$$I = \sum_{i=1}^{C} (w_i \times P_{IE}(y = c_i|v)) \quad (19)$$

where I is the interestingness estimate, C is the class number, w_i and $P_{IE}(y = c_i|v)$ are the weight and probability of each class.

We use linear spacing to define the class weights, such that for a 5-class classifier, the 5 classes have weights of 0, 0.25, 0.5, 0.75, and 1.0 for class 1 to 5, respectively. It can be seen that I would be a real number between 0.0 and 1.0. Similar to binary feedback, our definition of the interestingness estimate changes in the same direction as the model's belief of v's interestingness. For example, if the probability of class 3 decreases and the probability of class 4 increases, it means that the model's belief of v's interestingness increases, and so does the interestingness estimate.

For real number and pairwise comparison feedback, the IE's predicted score $s_{IE}(v)$ can be directly used as the interestingness estimate for the view v.

If the user is satisfied with the view recommendation, then the user interaction stops; otherwise, the `ViewSeeker` would start a new loop by selecting new examples for user labeling.

5. Experimental Evaluation

In this section, we present two sets of experiments. The first set of experiments is to evaluate the UF Integration phase without the involvement of the UF Tuning phase. The second set of experiments is to evaluate the combined effectiveness of the UF Tuning and the UF Integration phases. We built a `ViewSeeker` platform in Python and the experiments were performed on a Core i5 server with 8GB of RAM. All our experiment settings are listed in Table 12.

5.1. Evaluation of Utility Function Integration

5.1.1. Experimental Settings

Datasets We used two datasets: the DIAB dataset and the CENSUS dataset. The DIAB dataset is a real-world dataset of diabetic patients [14]. We removed the attributes that have a large amount of missing data. After preprocessing, the data set has 100 thousand records, 7 dimension attributes, and 8 measure attributes. The CENSUS dataset contains microdata from the U.S. labor force survey [15]. We removed the "not in universe" records and records with zero income. After preprocessing, the dataset has 100 thousand records, 5 dimension attributes, and 5 measure attributes. All measure attributes in the two datasets are normalized to a real number range between 1 and 100.

Query Simulation We use the following steps to generate each query subset D_Q. 1) Randomly select a dimension attribute a, 2) Randomly select a group g in a, 3) Select all records in g as D_Q. For example, if a is gender and g is female, then D_Q would be all female records. We used the above method to generate 10 random query subsets D_Q, and the reported results are the average from running 10 different experiments using those 10 query subsets.

Individual Utility Functions In our experiment, we have used eight individual utility functions. The first five utility functions are deviation-based UFs: Kullback-Leibler divergence (KL), Earth Mover Distance (EMD), L1 distance (L1), L2 distance (L2), and the maximum deviation in any individual bin (MAX_DIFF). The remaining three utility functions represent the *usability* [5], *accuracy* [5], and *p-value* [16]. Usability refers to the quality of the visualization in terms of providing the analyst with an understandable, uncluttered representation, which is quantified via the relative bin width metric. Accuracy refers to the ability of the view to accurately capture the characteristics (i.e., distribution) of the analyzed data, which is measured in terms of *Sum Squared Error* (SSE). The p-value is a statistical term defined as "the probability of obtaining a result equal to or more extreme than what is observed, with the given null hypothesis being true" [25]. In the problem of view recommendation, the *null hypothesis* refers to the reference view, and the *extremeness of the results* refers to the interestingness of the target views.

Table 12: Testbed Parameters

Experiment Set	UF Integration	UF Tuning and Integration
Total number of records ($\|D\|$)	$100K$ (DIAB), $100K$ (CENSUS)	
Average query set size ($\|D_Q\|$)	$43K$ (DIAB), $47K$ (CENSUS)	$36K$ (DIAB), $39K$ (CENSUS)
Average reference set size ($\|D_R\|$)	N/A	$36K$ (DIAB), $39K$ (CENSUS)
Difference ratio between D_Q and D_R	N/A	10%, 20%, 30%
Number of dimension attributes (A)	7 (DIAB), 5 (CENSUS)	
Number of measure attributes (M)	8 (DIAB), 5 (CENSUS)	
Number of aggregate functions	5	3 (COUNT, SUM, AVG)
Total view count	280 (DIAB), 125 (CENSUS)	168 (DIAB), 75 (CENSUS)
Number of individual utility functions	8	
Feedback type for UF integration	Real number	
View interestingness estimator	Linear regressor	
Example informativeness estimator	Query-by-committee	
Number of views presented per iteration	1	
Evaluation metrics	Top-k accuracy	EEE, Runtime, Top-k accuracy
The number of views to recommend (k)	5,10,15,20	
Runs for each configuration	10 (with different D_Q)	10 (with different D_Q and D_R)

It should be noted that, in general, users may customize the UFs, including adding new ones, for a personalized analysis. The current set of UFs mentioned above are selected to illustrate the effectiveness of `ViewSeeker`.

Since this set of experiments does not involve UF tuning, the default calculation methods of the individual UFs were used to generate the learning representation of the views. The reference views in the deviation-based UFs are calculated using the Global approach as discussed in Section 3. All UF scores were normalized to a range between 0.0 and 1.0 across all views to avoid learning and prediction bias due to the range difference in the original UF scores.

User Simulation We simulated different data analysis contexts, where each context is associated with an *ideal utility function* (IUF). Each IUF represents how the user perceives the interestingness of views under each simulated context. That is, IUF acts as the ground truth $u()$ for that context. Similar to all our utility functions, the output of each IUF is a real number between 0.0 and 1.0, with 0.0 being not interesting and 1.0 being very interesting. We designed 11 diverse IUFs that included 3 single-component UFs and 8 multi-component (i.e., multi-objective) composite UFs (Table 13). We chose the components in multi-component IUFs carefully such that they represent different characteristics of the candidate views. For example, EMD measures the absolute differences across the bins; KL-divergence measures the relative entropy between the two distributions; Usability represents the visual quality of a view, etc.

Evaluation Metrics We evaluated the performance of `ViewSeeker` in the aspect of recommendation accuracy. Specifically, we measured the number of labeled examples needed for the IE to reach an 100% recommendation accuracy. Here we define the accuracy as the size of the intersection between the top-k views recommended by `ViewSeeker` and the top-k views recommended by the IUF. For two sets of top-k views V^p and V^* produced by

Table 13: Simulated Ideal Utility Functions

#	Ideal Utility Functions
1	1.0 * KL
2	1.0 * EMD
3	1.0 * MAX_DIFF
4	0.5 * EMD + 0.5 * KL
5	0.5 * EMD + 0.5 * L2
6	0.5 * EMD + 0.5 * p-value
7	0.3 * EMD + 0.3 * KL + 0.4 * MAX_DIFF
8	0.3 * EMD + 0.3 * L2 + 0.4 * MAX_DIFF
9	0.3 * EMD + 0.3 * p-value + 0.4 * MAX_DIFF
10	0.3 * EMD + 0.3 * KL + 0.4 * Usability
11	0.3 * EMD + 0.3 * KL + 0.4 * Accuracy

`ViewSeeker` and the IUF, respectively, the accuracy is calculated as: $\frac{|V^p \cap V^*|}{k}$.

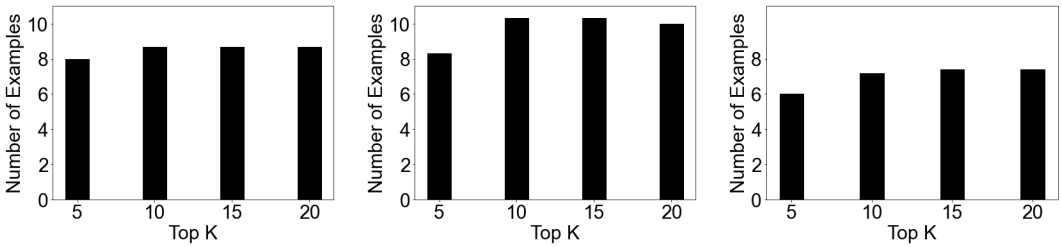
5.1.2. Experimental Results

Figures 4 and 5 show the number of example views that need to be labeled in order for the view interestingness estimator IE to reach an 100% accuracy in the top-k recommended views. Here, the x-axis is the k in top-k (i.e., the number of views on which the accuracy calculation is based), and the y-axis is the number of example views presented, capturing the user effort required for the accuracy to reach 100%.

Specifically, Figures 4a and 5a are for one-component IUFs (i.e., average result over IUF 1-3 in Table 13). Figures 4b and 5b are for two-component composite IUFs (i.e., average result over IUF 4-6). Finally, Figures 4c and 5c are for three-component composite IUFs (i.e., average result over IUF 7-11).

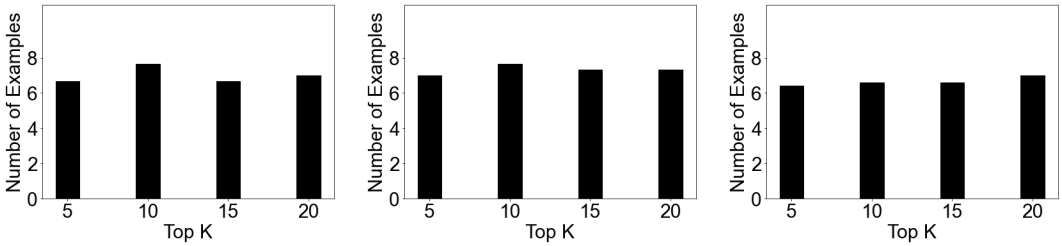
From these results, we can observe that `ViewSeeker` is extremely effective in discovering the set of ideal top-k views: for k ranging from *5-20*, on average only *6-11* labels were required before `ViewSeeker` reached an accuracy of 100% for both DIAB and CENSUS datasets. Clearly, this indicates that only a small amount of user effort is needed before a satisfactory set of results can be obtained by `ViewSeeker`'s UF integration functionality.

To the best of our knowledge, currently there are no existing solutions that provide the same functionality as `ViewSeeker` to act as a baseline for performance evaluation. However, to further study the performance of `ViewSeeker`, we compare it to a set of alternative baselines that are based on predefined UFs, as shown in Figure 6. Particularly, while `ViewSeeker` tries to accurately estimate the ideal utility function for a certain context, each of those baselines assumes that ideal utility function to be one of the eight predefined individual utility functions discussed in the previous section (e.g., KL, EMD, L1, etc). Figure 6 shows the average accuracy achieved by those baselines vs. `ViewSeeker` across all of our 11 IUFs when evaluated on both the DIAB and CENSUS datasets. As the figure shows, `ViewSeeker` outperforms all baselines with an average accuracy improvement of 86.9% over the baseline average, and 24.6% over the best performing baselines (i.e., EMD and L1). This demonstrates the `ViewSeeker`'s capability to achieve a high recommendation accuracy improvement against predefined UFs with a very small amount of additional user effort.



(a) Single-Component u*() (b) Two-Component u*() (c) Three-Component u*()

Figure 4: Recommendation accuracy for DIAB dataset with different ideal utility functions u*().



(a) Single-Component u*() (b) Two-Component u*() (c) Three-Component u*()

Figure 5: Recommendation accuracy for the CENSUS dataset with different ideal utility functions u*().

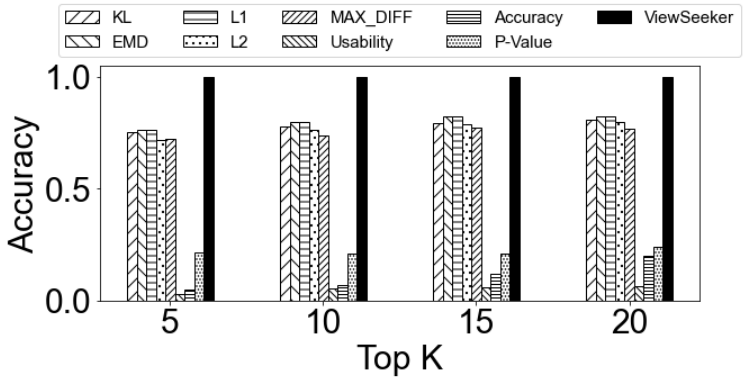


Figure 6: Maximum achievable accuracy by baselines and ViewSeeker

5.2. Evaluation of Utility Function Tuning and Integration

5.2.1. Experimental Settings

Datasets We used the same datasets as the previous experiment set.

User Simulation for UF Tuning Assume that the whole dataset is called D, and the data subset specified by the query (i.e., query set) is called D_Q. To simulate the user expectation, we generated a new data subset called the *reference set* D_R, which represents the user's prior knowledge of D_Q, such that views generated from D_R represent user's expectation for the corresponding views generated from D_Q. We generate D_Q and D_R in the following way: 1) Randomly select a dimension attribute a. 2) Randomly select a group g in a. 3)

Select all records in g as a sample S. 4) Randomly select two subsets of records D_Q and D_R from S, such that $|D_Q| = |D_R|$ and $D_Q \cup D_R = S$.

The *difference ratio* between D_Q and D_R is used to describe the degree of difference between the two subsets and is defined as $\frac{|D_Q - D_R|}{|D_Q|}$.

The two partially overlapping subsets D_Q and D_R from a group in a dimension represent two subsets that have similar data characteristics but are not identical. This is usually the situation for user prior knowledge about certain group of records, such that the user's expectation of the group aggregate characteristics may agree with most but not all of the group aggregate characteristics expressed by the records retrieved by the query.

After generating D_Q and D_R, for each view v_i (i.e., a (a, m, f) triple), two views can be generated: a *target view* (v_i^T) which is generated by applying (a, m, f) on D_Q, and an *ideal reference view* (v_i^{IR}) which is generated by applying (a, m, f) on D_R simulating the user's expectation for v_i^T.

We used the above method to generate a random query subsets D_Q and a random reference set D_Q in each experiment, and the reported results are the average from running 10 different experiments using 10 different query subsets and reference subsets.

User Simulation for UF Integration We used the same individual UFs and IUFs as in the previous experiment set to simulate the user. The ground truth scores for deviation-based UFs are calculated based on the target views (v_i^T) and the corresponding ideal reference views (v_i^{IR}).

Reference View Generation Models The three models for reference view generation are the Global, Local, and `ViewSeeker`, which were discussed in Section 3. The Global and Local are baselines in the experiments. For a target view v_i^T, each model uses its own approach to generate the reference view v_i^R and uses the v_i^R to calculate its own scores for the deviation-based UFs.

Evaluation Metrics We evaluated `ViewSeeker`'s performance in multiple aspects. For UF tuning, we measured the system effectiveness using *expectation estimate error (EEE)*, and the system efficiency using system runtime. For the combined process of UF tuning and integration, we used the view recommendation accuracy for effectiveness evaluation.

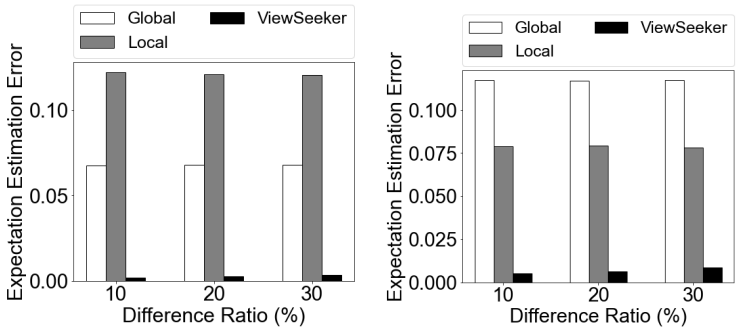
We first give the definition of EEE as follow. As mentioned above, for a target view v_i^T, each model generates its own reference view v_i^R, and EEE measures the distance between v_i^R and the corresponding ideal reference view v_i^{IR} using Equation 20.

$$EEE_i = DT(P(v_i^R), P(v_i^{IR})) \tag{20}$$

where DT is the distance function and $P(\cdot)$ is the view normalization operator as introduced in Section 2. The $L1$-norm distance function was used as DT in the experiments. The final EEE score is the average over the EEE scores of all views.

We measured the runtime for the four parts of the UF Tuning phase as mentioned in Section 3.4. We fixed the size of the whole dataset D, the query set D_Q, and the reference set D_R, and focused on the influence of the data dimensionality on the system runtime.

For view recommendation accuracy, our evaluation metric is the Top-k accuracy as used in the previous experiment set. We measured the accuracy after 10 labeled examples when



(a) DIBA dataset (b) CENSUS dataset

Figure 7: Expectation Estimate Error comparison for different reference generation models.

the model learning stabilized for most of the configurations.

5.2.2. Experimental Results

Expectation Estimate Error Figure 7 shows the EEE comparison results. For the DIAB dataset, the three models have achieved an average EEE of 6.8% (Global), 12.1% (Local), and 0.3% (`ViewSeeker`) across all difference ratios. For the CENSUS dataset, the three models have achieved an average EEE of 11.7% (Global), 7.9% (Local), and 0.7% (`ViewSeeker`).

It can be seen that `ViewSeeker` consistently estimates the ideal reference views v_i^{IR} significantly more accurately than the two baselines. Specifically, `ViewSeeker` achieved a relative EEE reduction of 96.0% against Global and 97.8% against Local for the DIAB dataset. Further `ViewSeeker` achieved a relative EEE reduction of 94.2% against Global and 91.4% against Local for the CENSUS datasets.

Scalability The runtime result for different parts of the UF Tuning phase is shown in Table 14. We have tested different data dimensionality by using different number of dimension attributes A and number of measure attributes M (as defined in Section 3.4. We used the CENSUS dataset and simulated different data dimensionality by duplicating dimension and measure attributes of the dataset. The result was averaged over all possible views that can be labeled.

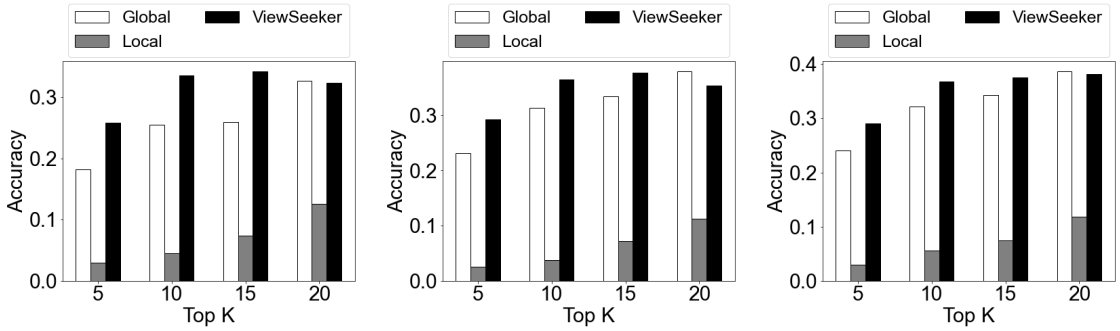
The first and second parts are view generation and helper view/matrix generation, which could be offline calculation. The third and four parts combined is the interactive EP process, which requires interactive system response time.

The runtime result verified our complexity analysis in Table 10 in Section 3.4, such that part 1 (view generation) and part 3&4 (interactive EP) has a runtime proportional to AM, and part 2 (helper view/matrix generation) has a runtime proportional to AAM. Besides, we can see that the runtime of part 3&4 (interactive EP) is at the level of millisecond and would remain interactive (i.e., sub-second) even for dataset with hundreds of dimension and measure attributes.

Recommendation Accuracy Figures 8 and 9 show the view recommendation accuracy comparison results. For the DIAB dataset, the three models have achieved an average

Table 14: Runtime in UF Tuning for CENSUS dataset

Part	Name	$A = M = 5$	$A = M = 10$	$A = M = 20$
1	View generation	0.53s	3.00s	19.12s
2	Helper view/matrix generation	2.61s	29.43s	243.45s
3&4	EP across dimensions and measures	0.62ms	1.54ms	6.11ms



(a) Difference ratio = 10% (b) Difference ratio = 20% (c) Difference ratio = 30%

Figure 8: Recommendation Accuracy comparison for DIAB dataset with various difference ratios between D_Q and D_R.

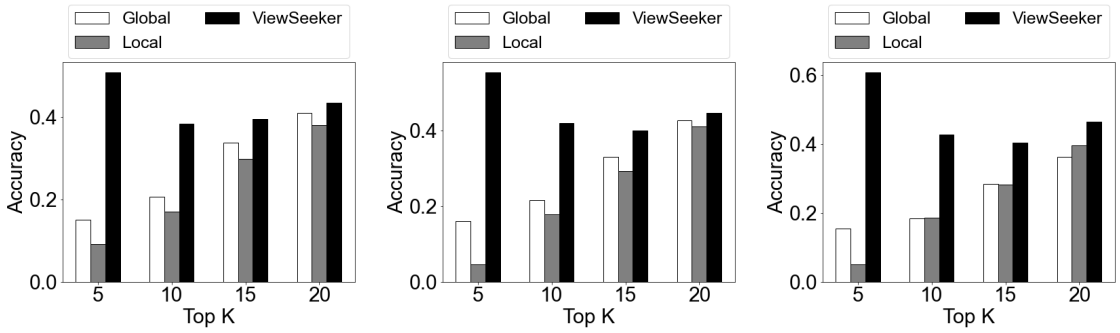
accuracy of 29.7% (Global), 6.7% (Local), and 33.8% (`ViewSeeker`) across all simulated IUFs and difference ratios. For the CENSUS dataset, the three models have achieved an average accuracy of 26.9% (Global), 23.2% (Local), and 45.4% (`ViewSeeker`).

Again, we see that `ViewSeeker` consistently performs significantly better than the two baselines. Specifically, `ViewSeeker` achieved a relative accuracy improvement of 14.4% against Global and 410.3% against Local for the DIAB dataset. And `ViewSeeker` achieved a relative accuracy improvement of 69.6% against Global and 95.5% against Local for the CENSUS datasets.

The user effort for the UF Tuning phase is one example view for each aggregate function and the number of groups for user expectation labeling in each example view is 26 for DIAB and 20 for CENSUS. This indicates that only a small amount of user effort is required in the UF Tuning phase for the `ViewSeeker` to achieve a significant improvement in recommendation accuracy against the Global and Local approaches.

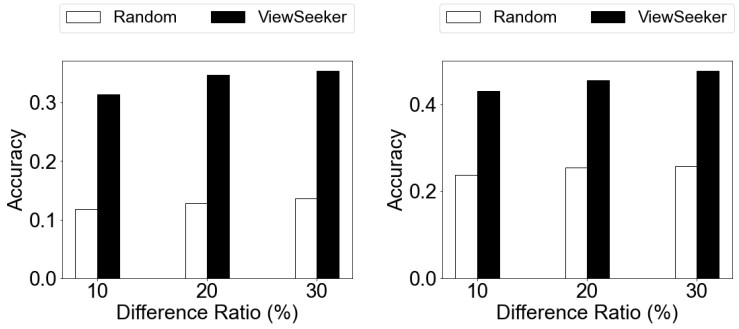
Example Selection Strategy Lastly, a set of experiments were conducted to evaluate the effectiveness of `ViewSeeker`'s example selection strategy in the UF Tuning phase. As discussed in Section 3.3.4, for each aggregate function, `ViewSeeker` selects the view with the dimension attribute that has the highest group count as the example view in the UF Tuning phase. We compare our designed example selection strategy with a baseline strategy, which selects a random example view for each aggregate function in the UF Tuning phase.

Figure 10 shows the comparison result for average accuracy across all IUFs and Top-k's. It can been seen that our designed example selection strategy consistently outperformed the baseline, with average accuracy improvements of 165.2% and 81.5% for the DIAB and CENSUS datasets, respectively. This result illustrates the effectiveness of the designed



(a) Difference ratio = 10% (b) Difference ratio = 20% (c) Difference ratio = 30%

Figure 9: Recommendation Accuracy comparison for CENSUS dataset with various difference ratios between D_Q and D_R.



(a) DIAB Dataset (b) CENSUS Dataset

Figure 10: Recommendation Accuracy comparison for different query strategies in UF Tuning phase.

example selection strategy in selecting informative examples in the UF Tuning phase.

6. Related Works

Expectation Propagation techniques estimate user expectation for the visualizations and have been studied by several works (e.g., [26, 27]). The cube exploration work [26] estimates the user expectation of views based on the values in the previously visited views using the maximum entropy principle. The data mining work [27] allows the user to provide expectation in spatio-temporal summaries (i.e., views) and uses a graph approach to propagate user expectation to other views. Both works focus on user expectation estimation in views of other data subsets with the same dimension and measure as the labeled view, while the EAP algorithm in `ViewSeeker` is designed to estimate user expectation in views of the same data subset with different dimensions and measures than the labeled view.

View Recommendation techniques automatically generate all possible views of data, and recommend the top-k interesting views, according to some utility function (e.g., [2, 5, 7, 8, 17–21]). A key difference among these works is the proposed utility functions. Works such as Voyager [17] and DeepEye [8] recommend the visualizations based on visual quality utility functions such as visual expressiveness and perceptual effectiveness. Other works

such as SeeDB [2] and QuickInsights [21] adopt deviation-based utility functions where the view interestingness is measured by the distance between the target view and the reference view. Some recent works such MuVE [5] and DiVE [7] have proposed multi-objective utility functions which capture different aspects of the view interestingness (e.g., visual quality, deviation, diversity) at the same time. The key difference between `ViewSeeker` and all prior work is that all previous works use predefined view utility functions and do not discover the most suitable utility function in the current analysis context.

Interactive Visualization Tools have been extensively studied for the past few years [3, 4, 28–32]. Unlike visualization recommendation frameworks, such as `ViewSeeker` that recommend visualization automatically by searching through the entire view spaces, traditional interactive visualization tools require the user to manually specify the views to be generated. Recently, a few interactive visualization tools have attempted to automate part of the data analysis and visualization process. Two works that are close to our work are VizDeck [3] and the work [31]. VizDeck [3] uses the data characteristics of the attributes in the view as the view representation and leverages user voting up/down labels to learn a linear model to rank the views. However, VizDeck displays all possible views available for labeling in a scrollable dashboard without any application of query strategy to select the most informative views for the user to label. The work [31] leverages user binary labels to learn a binary classification model for view relevance prediction. `ViewSeeker` differs from [31] by learning the ideal utility function to rank and get the top-k views.

Data Exploration techniques that aim to efficiently extract knowledge from data [33] are complementary to our work. In particular, *example-driven* data exploration approaches [34, 35] share the same philosophy as `ViewSeeker` and reply on user feedback on examples to refine the exploratory queries. `ViewSeeker` is well suited to such situations and can enhance example-driven data exploration by creating visualizations that illustrate interesting patterns during the construction of the exploratory queries.

7. Conclusion

In this work, we claim that traditional view recommendation lacks analysis context adaptability, and frame this as the *Context-Aware View Recommendation problem* (CoVR). Recognizing that any effective solution of CoVR should directly involve the users, we advocate for a human-in-the-loop paradigm, called *Interactive View Recommendation* (IVR), in which the system interacts with the user to discover the most suitable utility function (UF) in the current analysis context.

We further present the first IVR solution, coined `ViewSeeker`, which supports two forms of adaptation, *UF Tuning* and *UF Integration*, operating as its first and second phase, respectively. UF Tuning implements a novel *Expectation Acquisition and Propagation* (EAP) algorithm that elicits user expectation of common scenarios for example views and estimates user expectation for other views to improve the accuracy of deviation-based utility functions. UF Integration employs active learning techniques to select informative example views for feedback (i.e., labeling) and utilizes the labels to discover the most suitable combination of UFs.

Our extensive set of experiments using two real-world datasets verify the effectiveness and efficiency of the UF Tuning and UF Integration phases of `ViewSeeker`. Specifically, they show that the EAP algorithm can estimate the user expectation much more accurately than the baselines, and can maintain interactive responsive time even for datasets with hundreds of dimensions. They also show that `ViewSeeker` (i.e., the combination of UF Tuning and UF Integration) can achieve high recommendation accuracy with minimum user effort and has an average accuracy improvement of 147.5% against the alternative baselines.

References

[1] X. Qin, Y. Luo, N. Tang, G. Li, Making data visualization more efficient and effective: a survey, VLDB J. 29 (1) (2020) 93–117.

[2] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, N. Polyzotis, SEEDB: efficient data-driven visualization recommendations to support visual analytics, VLDB 8 (13) (2015) 2182–2193.

[3] A. Key, B. Howe, D. Perry, C. R. Aragon, Vizdeck: self-organizing dashboards for visual analytics, in: ACM SIGMOD, 2012, pp. 681–684.

[4] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, J. Heer, Profiler: integrated statistical analysis and visualization for data quality assessment, in: ACM AVI, 2012, pp. 547–554.

[5] H. Ehsan, M. A. Sharaf, P. K. Chrysanthis, Muve: Efficient multi-objective view recommendation for visual data exploration, in: IEEE ICDE, 2016.

[6] H. Ehsan, M. A. Sharaf, P. K. Chrysanthis, Efficient recommendation of aggregate data visualizations, IEEE Trans. Knowl. Data Eng. 30 (2) (2018) 263–277.

[7] R. Mafrur, M. A. Sharaf, H. A. Khan, Dive: Diversifying view recommendation for visual data exploration, in: ACM CIKM, 2018, pp. 1123–1132.

[8] Y. Luo, X. Qin, N. Tang, G. Li, Deepeye: Towards automatic data visualization, in: IEEE ICDE, 2018.

[9] X. Zhang, X. Ge, P. K. Chrysanthis, M. A. Sharaf, Viewseeker: An interactive view recommendation tool, in: BigVis Workshop, 2019.

[10] X. Zhang, X. Ge, P. K. Chrysanthis, Leveraging data-analysis session logs for efficient, personalized, interactive view recommendation, in: IEEE CIC, 2019.

[11] X. Zhang, X. Ge, P. K. Chrysanthis, Interactive view recommendation with a utility function of a general form, in: HILDA, 2020.

[12] X. Zhang, X. Ge, P. K. Chrysanthis, Evaluating query strategies for different feedback types in interactive view recommendation, in: IEEE IV, 2020.

[13] B. Settles, Active learning literature survey, Tech. rep., University of Wisconsin-Madison Department of Computer Sciences (2009).

[14] Diabetes data set (2019).
URL https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008/

[15] S. Flood, M. King, R. Rodgers, S. Ruggles, R. Warren, Integrated public use microdata series, current population survey: Version 7.0 [dataset], in: Minneapolis, MN: IPUMS, 2020.
URL https://doi.org/10.18128/D030.V7.0

[16] B. Tang, S. Han, M. L. Yiu, R. Ding, D. Zhang, Extracting top-k insights from multi-dimensional data, in: ACM SIGMOD, 2017.

[17] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, J. Heer, Voyager: Exploratory analysis via faceted browsing of visualization recommendations, IEEE Trans. Vis. Comput. Graph. 22 (1) (2016) 649–658.

[18] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, J. Heer, Voyager 2: Augmenting visual analysis with partial view specifications, in: ACM CHI, 2017.

[19] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, J. Heer, Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco, IEEE Trans. Vis. Comput. Graph. 25 (1) (2019) 438–448.

[20] B. Mutlu, E. E. Veas, C. Trattner, Vizrec: Recommending personalized visualizations, ACM Trans. Interact. Intell. Syst. 6 (4) (2016) 31:1–31:39.
[21] R. Ding, S. Han, Y. Xu, H. Zhang, D. Zhang, Quickinsights: Quick and automatic discovery of insights from multi-dimensional data, in: ACM SIGMOD, 2019, pp. 317–332.
[22] NBA, - http://www.basketball-reference.com.
[23] Quickinsights - insight types specification, https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Insight-Types-Specification.pdf (2020).
[24] T. Liu, Learning to Rank for Information Retrieval, Springer, 2011.
[25] M. Krzywinski, N. Altman, Significance, p values and t-tests, in: Nature methods, 2013.
[26] S. Sarawagi, User-adaptive exploration of multidimensional data, in: VLDB, 2000, pp. 307–316.
[27] H. J. Hamilton, L. Geng, L. Findlater, D. J. Randall, Efficient spatio-temporal data mining with genspace graphs, J. Appl. Log. 4 (2) (2006) 192–214.
[28] J. D. Mackinlay, P. Hanrahan, C. Stolte, Show me: Automatic presentation for visual analysis, IEEE Trans. Vis. Comput. Graph. 13 (6) (2007) 1137–1144.
[29] C. Stolte, P. Hanrahan, Polaris: A system for query, analysis and visualization of multi-dimensional relational databases, in: IEEE INFOVIS, 2000.
[30] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, J. Goldberg-Kidon, Google fusion tables: web-centered data management and collaboration, in: ACM SIGMOD, 2010.
[31] M. Behrisch, F. Korkmaz, L. Shao, T. Schreck, Feedback-driven interactive exploration of large multi-dimensional data supported by visual classifier, in: IEEE VAST, 2014.
[32] A. Satyanarayan, J. Heer, Lyra: An interactive visualization design environment, Comput. Graph. Forum 33 (3) (2014) 351–360.
[33] D. Mottin, M. Lissandrini, Y. Velegrakis, T. Palpanas, New trends on exploratory methods for data analytics, VLDB 10 (12) (2017) 1977–1980.
[34] X. Ge, Y. Xue, Z. Luo, M. A. Sharaf, P. K. Chrysanthis, REQUEST: A scalable framework for interactive construction of exploratory queries, in: IEEE Big Data, 2016.
[35] K. Dimitriadou, O. Papaemmanouil, Y. Diao, Explore-by-example: an automatic query steering framework for interactive data exploration, in: ACM SIGMOD, 2014.