Leveraging Interactive User Feedback for Personalized Data Visualization Recommendation

by

Xiaozhong Zhang

Bachelor of Engineering, Tongji University, 2009

Master of Science, SUNY Buffalo, 2015

Submitted to the Graduate Faculty of

the Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2023

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Xiaozhong Zhang

It was defended on

December 15th 2022

and approved by

Prof. Panos K. Chrysanthis, PhD, Chair

Prof. Adriana Kovashka, PhD, Examiner

Prof. Alexandros Labrinidis, PhD, Examiner

Prof. Mohamed A. Sharaf, PhD, External Examiner

Dissertation Director: Prof. Panos K. Chrysanthis, PhD, Chair

Leveraging Interactive User Feedback for Personalized Data Visualization Recommendation

Xiaozhong Zhang, PhD

University of Pittsburgh, 2023

Data visualization is a powerful tool to communicate information clearly and efficiently, and is widely used in data analysis to support data-driven decision making. However, selecting a good visualization (i.e., view) that is both usable and useful is not a trivial task, because the search space of visualization configurations (e.g., variables to visualize, data transformations, visual encodings) is prohibitively large. In order to aid the user, a variety of utility functions have been proposed to estimate the utility of the views, so as to provide view recommendations, such as usability and usefulness. Different utility functions assess different aspects of a view's utility and can also be combined to form multi-objective utility functions.

Traditional view recommendation systems typically do not consider user preferences of utility functions or user expectation of view values, leading to further suboptimal recommendation. A predefined utility function is not likely to fit all users' needs at all times. User expectation also plays a role in the utility of a view. For example, if a peculiar pattern is expected by the user, then they will not find it very interesting in a recommended view. In order to support effective view recommendation for a given analysis, in this thesis, we propose a new paradigm, called *Interactive View Recommendation (IVR)*, in which the system iteratively interacts with the user to elicit user preference information and expectations to formulate the most suitable utility function. IVR has two main objectives: *high recommendation quality* and *low user interaction effort*. However, these two objectives are usually in conflict, making the IVR problem even more challenging than traditional view recommendations.

As a proof-of-concept, we propose ViewSeeker, an active learning-based IVR framework and prototype system, which intelligently selects informative questions for user feedback to achieve a satisfactory recommendation quality while minimizing user interaction effort. ViewSeeker implements three main functionalities: (1) utility function preference learning, (2) user expectation learning, and (3) external knowledge base utilization. Simulated and real user studies were con-

ducted to evaluate ViewSeeker's effectiveness in leveraging interactive user feedback for personalized view recommendation, showing that with acceptable user effort, ViewSeeker can outperform current non-IVR recommenders.

Table of Contents

List of Tables

List of Figures

Preface

I would like to express my most sincere thanks to everyone who has made my journey to Ph.D. possible and joyful.

First of all, I want to express my deepest gratitude to my advisor, Prof. Panos K. Chrysanthis, who has always been supportive both academically and personally. Prof. Chrysanthis has not only taught me the way to conduct high-quality research and teaching, but also encouraged and guided me through many difficult moments during my graduate study. I would also like to thank my thesis committee members, namely Prof. Adriana Kovashka, Prof. Alexandros Labrinidis, and Prof. Mohamed A. Sharaf, who have given me invaluable guidance and help in my research.

I also want to thank my lab colleagues and alumni, especially Xiaoyu Ge, who has guided me through many projects and provided visionary ideas and insightful comments. Others to thank include: Rakan Alseghayer, Constantinos Costa, Nick Katsipoulakis, Brian Nixon, Daniel Petrov, Thao Pham, Anatoli Shein, and many more.

I dedicate this work to my parents, Fujing and Xiaofen. Their belief and support are indispensable to me and have helped me through many hardships during my study. Thank you!

1.0 Introduction

"People don't know what they want until you show it to them."

– Steve Jobs

1.1 Motivation

In the current era of Big Data, data visualization is a powerful tool to communicate information clearly and efficiently and is widely used in data analysis to support data-driven decision making. However, since the search space of visualization configurations (e.g., variables to visualize, data transformations, visual encodings) is prohibitively large, selecting a good visualization (i.e., view) that is both *usable* and *useful* is not a trivial task, even for the expert users.

In order to aid the user, a variety of utility functions have been proposed to estimate the utility of the views, so as to provide view recommendations. Different utility functions assess different aspects of the view utility, such as *usability* (e.g., visual quality [33, 15]), *usefulness* (e.g., pattern peculiarity [32, 6]), and *diversity* of a set of views [18]. Furthermore, multiple utility functions can be combined to form a multi-objective utility function that assesses different aspects of the view utility at the same time [7, 18, 6].

However, state-of-the-art view recommendation systems usually rely only on predefined utility functions to rank the views without taking into account utility function preferences from the user, leading to suboptimal recommendations, because a predefined utility function would not likely meet all users' interests at all times. For example, SeeDB [32] uses a predefined utility function based on the L1 distance between a view and a reference, and does not allow the user to change to other utility functions. Other works such as Voyager [33] and DeepEye [15] reply on perceptual-principle-guided partial orders between view encodings to rank the views, which essentially serve as a predefined utility function collectively and offer no adaptability to perceptual differences between users.

Furthermore, the calculations of the utility functions in state-of-the-art view recommenders

are usually also predefined, resulting in further suboptimal recommendations. For example, as mentioned above, the L1 distance calculation in SeeDB involves a view and a reference. The reference represents the common case scenario for the view and has a predefined generation method. However, different users might have different prior knowledge and expectations about the common case scenario for the view, which are not likely to be captured by a predefined generation method. QuickInsights [6] also uses a difference measure between a view and a reference as the utility function. However, it suffers from the same issue with a predefined reference generation method.

The above observations have pointed to the need for a more effective view recommendation approach in which the system would acquire user information, such as user utility function preference and common case scenario expectation, and adjust the utility function accordingly to realize a personalized view recommendation. Under the assumption of no previous available knowledge about the user, the above-mentioned user information would need to be collected through interactive user feedback. This essentially makes the above approach a *user-in-the-loop* view recommendation approach, which is the main focus of this thesis.

1.2 Hypothesis & Approach

The underlying hypothesis of this thesis is that:

"A user-in-the-loop approach for view recommendations can improve the view recommendation quality by tailoring them to the user's interest."

User-in-the-loop implies interactivity and continues dialogue between the users and the system. To realize this and test our hypothesis, we propose a new paradigm, called *Interactive View Recommendation* (IVR), which interactively elicits user preference and expectation information to customize the view recommendation to the user's current analysis goals.

IVR mainly has two objectives: *high recommendation quality* and *low user interaction effort*. However, these two objectives are usually in conflict. To improve the recommendation quality requires more user information through more user feedback. To reduce user interaction effort will reduce user information at hand thus worsening the recommendation quality. To achieve a

personalized recommendation while balancing the above two objectives makes the IVR problem even more challenging than the traditional view recommendation problem.

To effectively solve the IVR problem, we develop an IVR framework, coined *ViewSeeker*, with three main functionalities:

- The first functionality is *utility function preference learning*, in which the system leverages active learning techniques to interact with the user to discover the user's interests across different utility functions. We pose that the discovered utility function would provide better recommendation quality than predefined utility functions.
- The second functionality is *user expectation learning*, in which the system interactively estimates user expectation of the values in the views. Our claim is that the utility function calculated based on the discovered user expectation would provide better recommendation quality than predefined user expectations (i.e., references).
- The third functionality is *external knowledge base utilization*, in which the system utilizes user interest information from external knowledge base to further reduce the current user's effort in the IVR process. The utilization of the external knowledge base would be effective if the user effort with its help is less than the user effort without.

The thesis focuses on single-view-based utility functions, namely the utility functions whose scores can be calculated based on a single view. Multi-view-based utility functions, such as *diversity* [18] whose score is calculated based on the differences among a set of views, are not included as candidate utility functions in the thesis.

1.3 Thesis Contributions

The main contributions of our thesis are the Interactive View Recommendation (IVR) and its functionalities. These contributions support our hypothesis and claims and address the shortcomings of the current state-of-the-art techniques. Specifically,

- We developed a general IVR framework and prototype, called *ViewSeeker*, that interacts with the user to learn user expectations and preferences to provide personalized view recommen-

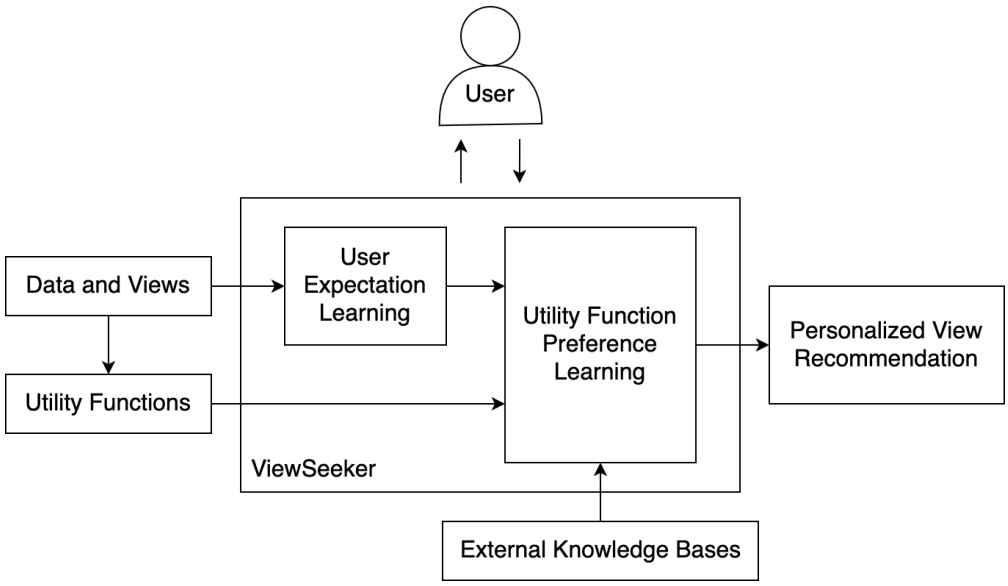


Figure 1: Overall workflow of ViewSeeker

dation. The overall workflow of ViewSeeker in Figure 1 illustrates the relations between its functionalities. First, user expectation can be estimated by the user expectation learning functionality, which in turn can be used to calculate individual utility functions such as deviation-based utility functions [32]. These utility functions together with other off-the-shelf utility functions will act as input features for the utility function preference learning functionality to learn the optimal combination of the features that fits the user's interest. External knowledge bases can also be used to learn generic user preference models to improve the recommendation quality in utility function preference learning, further reducing user effort in the IVR process.

- For utility function preference learning, we developed a variety of feedback types and their corresponding example forms, active learning-based example selection strategies and ranking algorithms under the IVR workflow [38, 36]. The simulated experimental results show that ViewSeeker can learn the ground truth utility functions well with fewer than 10 labels on average. We also showed that ViewSeeker outperforms non-IVR baselines by at least 24.6% in recommendation accuracy.
- For user expectation learning, we devised a novel algorithm, called *Expectation Acquisition &*

Propagation (EAP), which elicits user expectation on example views and based on them estimates user expectation on unlabeled views [39]. Within EAP, we designed expectation propagation algorithms across both dimensions and measures with corresponding runtime complexity analysis. The simulated experimental results indicate that ViewSeeker can reduce user expectation estimation error from state-of-the-art non-IVR recommenders by more than 90% with user labeling effort on only three example views. When combined with a downstream utility function preference learning process, ViewSeeker also outperforms the comparison models by 83.1% relatively in average recommendation accuracy.

- For external knowledge base utilization, we extended ViewSeeker to incorporate two stages [35]. In the offline stage, it builds a generic model library using data analysis session logs, and then in the online stage, it retrieves a suitable model from the library and refines it based on the current user's feedback during the IVR process. The simulated experimental results indicate that ViewSeeker could achieve an average recommendation accuracy improvement of 33.8% over an offline-only model with an average of 3 user labels. Besides, ViewSeeker could achieve the same accuracy as an online-only model with an average user effort reduction of 37.5%.
- We conducted extensive real user studies to evaluate the effectiveness of all three functionalities of ViewSeeker. For user expectation learning, the study result showed that ViewSeeker has an estimation error reduction of more than 50% against the state-of-the-art non-IVR model [6]. For utility function preference learning, ViewSeeker outperforms the baselines (i.e., individual utility functions) by a large margin, especially with a reasonable number of labels. Besides, the accurate user expectation estimates from the EAP algorithm also enables ViewSeeker to outperform [6] significantly in recommendation accuracy. For external knowledge base utilization, we refined the library model retrieval and online model refinement methods in order to extract more information from available user labels. We also developed a novel model convergence detection and swapping algorithm to further improve the model refinement process. The final ViewSeeker got the best of both worlds, achieving offline model's accuracy in the early stage and online model's accuracy in the late stage of the IVR process. Overall, the user study results are compatible with their simulated counterparts, although lower in general, yet significant.

1.4 Thesis Outline

The rest of the thesis is organized as follow: Section 2 discusses the background and related works of the thesis. Section 3 introduces ViewSeeker's utility function preference learning functionality. Section 4 introduces ViewSeeker's user expectation learning functionality. Section 5 introduces ViewSeeker's external knowledge base utilization functionality. Section 6 discusses the real-world user study setup and results and Section 7 concludes the thesis and discusses future work.

2.0 Background

In this chapter, we introduce the background concepts and related works to the thesis. We will first introduce the data visualization and its representation in the context of structural databases in Section 2.1. Then, we will talk about different utility function families proposed by previous works in Section 2.2. Finally, we will discuss related works for ViewSeeker's functionalities from Section 2.3 to Section 2.6.

2.1 Data Visualization and View Space

There is a variety of visualization types, such as bar chart, line chart, scatter plot, etc [17]. In this thesis, we focus on the most commonly used visualization type, namely the bar chart [22]. Bar charts are the preferred chart type for numerical and categorical attributes which are two common date types in databases [33].

In the context of structural databases, a bar chart view v can be seen as the result of an aggregate group-by SQL query Q. For example, the bar chart in Figure 2 illustrates the result of the below query:

```
SELECT SUM(Items_Sold) FROM Sales GROUP BY Brand;
```

It can be seen that there are four major components in this query:

- A table name (i.e., Sales), which defines the underneath dataset D.
- A measure attribute m (i.e., Items_Sold), which is a numerical attribute and can be aggregated.
- An aggregate function f (i.e., SUM) that is applied on the measure attribute. Other common aggregate functions are MAX, MIN, COUNT, AVG (Average), etc.
- A dimension attribute a (i.e., Brand), which is a categorical attribute and serves as the grouping basis of the aggregated values.

Therefore, we can represent a view v by a quadruple (D, a, m, f) that captures the components of the corresponding query. For cases when the dataset D is not changing, the representation can be

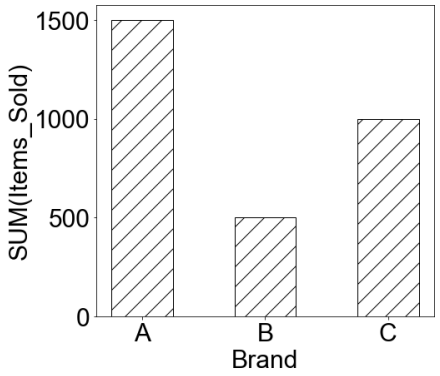


Figure 2: A view represented by the triple *(Brand, Items_Sold, SUM)*

simplified to a triple (a, m, f). Under the typical multi-dimensional data models, the data usually has a set of measure attributes $M = \{m_1, m_2, m_3, ...\}$ and a set of dimension attributes $A = \{a_1, a_2, a_3, ...\}$. Together with the choices of all possible aggregate functions $F = \{f_1, f_2, f_3, ...\}$, the View Space (VS), i.e., the total number of possible views will be:

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

where $|\cdot|$ is the cardinality operator. Clearly, the View Space can be very large, especially for high-dimensional data.

2.2 Utility Functions

Since the View Space is very large, manually selecting a view that is both usable and useful is not a trivial task for a user. In order to aid the user, a variety of utility functions have been proposed to estimate the utility of the views, so as to provide view recommendations.

Definition 1. Utility Function: A utility function $u()$ is a mapping from a view v to a real number $u(v)$ representing the utility estimate of the view.

Common representative utility function categories from the visualization & graphics community and the database & data mining community are discussed below.

2.2.1 Usability

This category usually focuses on the visual quality of the views. For example, in [16] and [33], the authors devices two ranking criteria for the views, *expressiveness* (whether a graphical language can express the desired information) and *effectiveness* (whether the graphical language exploits the capabilities of the output medium and the human visual system). For example, if there is a categorical attribute and a numerical attribute in the view, then the expressiveness criterion will tell that the chart types that support such data are bar chart, line chart and text table, while the effectiveness criterion will tell that among the supporting chart types, a bar chart is favored over a line chart over a text table.

2.2.2 Usefulness

This category usually focuses on the peculiarity of the patterns in the views, because views with peculiar patterns are usually worth further investigation.

One commonly used utility function in this category is called *deviation* [32]. This utility function is usually used when the visualization is based on a data subset D_Q of the entire dataset D. In order to calculate the deviation, we need two views. The original view to be scored (the *target* view v^T) and a helper view to facilitate the score calculation (the *reference* view v^R). If we represent the target view as (D_Q, a, m, f), then the corresponding reference view can be represented as (D, a, m, f). In other words, the reference view v^R is generated by applying the same (a, m, f) of the target view v^T on the whole dataset D. An example pair of target view and reference view is shown in Figure 3, where the target view represents the gender ratio of a data subset (e.g., kindergarten teachers) and the reference view represents the gender ratio of the whole dataset.

Then the deviation score can be calculated as the difference between v^T and v^R by some distance function such as L1 distance, L2 distance, etc. The deviation score would be low for a view v^T whose values are similar to those in v^R, which means that the data subset exhibits similar patterns as the whole dataset. On the other hand, the deviation score would be high for a view v^T whose values deviate largely from those in v^R (the case in the Figure), which means that the data subset exhibits very different patterns than the whole dataset, and thus is worth further investigation.

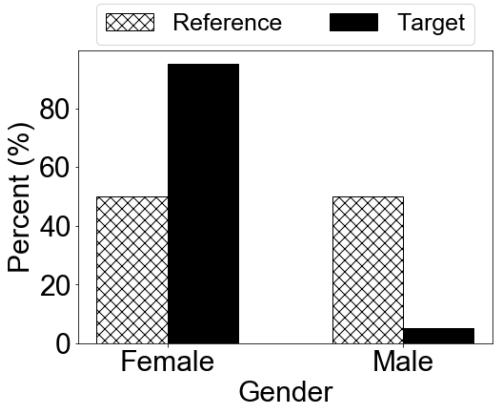


Figure 3: An example pair of target view and reference view

2.2.3 Diversity

Another category of utility functions that is commonly studied is based on *diversity* [18, 31]. Diversity measures the difference among the members of a set of views, and usually the higher the score the better. The intuition behind this measure is that it assumes that the more diverse a set of views is, the more information it can provide.

In order to calculate the diversity, a similarity (or distance) function needs to be defined between two views. For example in [18], this similarity is based on the *Jaccard similarity* between the two triples (a_1, m_1, f_1) and (a_2, m_2, f_2) of the two views v_1 and v_2, namely $\frac{|\{a_1,m_1,f_1\} \cap \{a_2,m_2,f_2\}|}{|\{a_1,m_1,f_1\} \cup \{a_2,m_2,f_2\}|}$, so that the more common elements they have, the higher the similarity.

2.2.4 Multi-Objective Utility Function

This category includes utility functions that assess multiple aspects of a view at the same time, and are usually in the form of a combination of multiple individual utility functions. The most common combination form is the linear form, such that different individual utility functions are linearly combined with a weight term for each. For example, the utility function in [7] is a linear combination of a usability utility function (i.e., conciseness S) and two usefulness utility functions

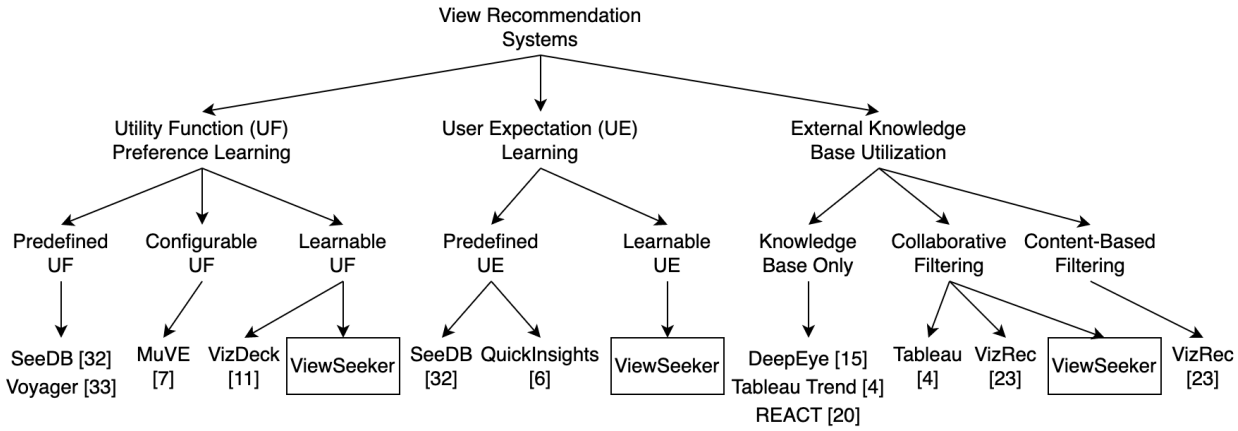


Figure 4: Related work taxonomy for ViewSeeker's functionalities

(i.e., deviation D and accuracy A) as shown in Equation 2 where the α's are the weights.

$$U(v) = \alpha_D \times D(v) + \alpha_A \times A(v) + \alpha_S \times S(v) \tag{2}$$

Similarly, the utility function in [18] is a linear combination of a usefulness utility function I and a diversity utility function f as shown in Equation 3 where S represents a set of views.

$$F(S) = (1 - \lambda) \times I(S) + \lambda \times f(S) \tag{3}$$

2.3 Utility Function Preference Learning Techniques

We will introduce the related works for ViewSeeker's functionalities in this and the following sections, which could be summarized in the taxonomy in Figure 4.

Utility function preference learning refers to the learning of the user's preference over all possible utility functions. The previous works could be divided into three categories based on the adaptability of the learning model towards the user's preference. The works in the lowest level usually have a predefined utility function and do not provide any adaptability to tune this fixed utility function based on the current user's interest. For example, SeeDB [32] has a predefined

deviation-based utility function (e.g., L1 distance) and does not allow the user to change to other utility functions. Another example is Voyager [33], in which preferences (i.e., partial orders) between view encodings are predefined according to perceptual principles.

The works in the second level usually allow the user to config the utility function manually so that it better fits the user's interest. For example, the multi-objective utility function in MuVE [7] assigns a weight for each composing individual utility function, and the weights can be tuned manually by the user to fit their preferences.

The works in the third level are most similar to ViewSeeker. These works such as VizDeck [11] also leverage user feedback to learn user utility function preference using machine learning models. The main difference between these works and ViewSeeker is that the former usually display all the views in a list and passively wait for the user to provide feedback while ViewSeeker actively asks the user to provide feedback on selected example views. ViewSeeker uses *active learning* techniques to select example views which will be introduced in the following section.

2.4 Active Learning Techniques

The key idea of active learning is that the learner actively selects unlabeled data instances for learning with the goal of achieving good prediction performance with as few user labels as possible.

The strategy that the learner uses to select the instances is called *query strategy* [26]. Generally, a query strategy strives to select the most informative instance. The informativeness of an instance measures the benefit from the label of that instance to the prediction performance improvement of the learner. However, since the actual informativeness can only be calculated after the acquisition of the label, various query strategies have been proposed to estimate the informativeness of the instances before label acquisition. The following are two commonly-used query strategy categories, which we explored in our work.

2.4.1 Uncertainty Sampling

The query strategies in the *uncertainty sampling* category are usually used for classification problems [26]. One typical uncertainty sampling strategy is called *least confident*. The least confident strategy selects the instance whose class label the learner is least confident about, as shown in Equation 4:

$$v^*_{LC} = \underset{v}{\operatorname{argmax}}(1 - P_\theta(\hat{y}|v)) \tag{4}$$

where v^*_{LC} is the selected instance, and $\hat{y} = argmax_y P_\theta(y|v)$ is the class label with the highest posterior probability by the learner θ. For example, for a binary classifier, the strategy will select the instance whose positive class probability is closest to 0.5, and for a multi-class classifier, the strategy will select the instance whose highest posterior probability in any class is the lowest.

2.4.2 Query-By-Committee

The query strategies in the *Query-By-Committee* (QBC) category can be used for both classification and regression problems [26]. These strategies usually build a committee (i.e., ensemble) of learners, and estimate the informativeness of the instances 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 instance.

For example, the learners could take the form of regression models in a regression problem. Each regression model in the committee c_i will predict a score $s_{c_i}(v)$ for an instance v. Then, one way to measure the disagreement among the committee members could be measuring the variance of their predictions, and the instance with the largest variance would be selected, as shown in Equation 5:

$$v^*_{QBC} = \underset{v}{\operatorname{argmax}}\, Var(\langle s_{c_1}(v), s_{c_2}(v), ..., s_{c_C}(v)\rangle) \tag{5}$$

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

2.5 User Expectation Learning Techniques

User expectation learning refers to the learning of the user's expectation over the values in the unlabeled views and state-of-the-art view recommendation works such as SeeDB [32] and QuickInsights [6] usually have a predefined method to estimate user expectation.

In SeeDB, the user's expectation of the values in a target view (D_Q, a, m, f) on a data subset D_Q is estimated by a reference view (D, a, m, f) on the whole dataset D. For example, the previously displayed Figure 3 shows a target view with a reference view generated using this method. This method is based on the assumption that, in common cases, the data in D_Q should exhibit similar patterns as the data in D. Therefore, when there is a large difference between the target view and the reference view, then the target view is likely to be interesting.

In QuickInsights, the user's expectation of the values in a target view is assumed to follow a power-law distribution and thus is calculated using power-law line fitting. To give a specific example, assume that the values in the target view are $\{x_1^T, x_2^T, x_3^T, x_4^T\}$. The method will first sort the values to form a sorted sequence, for instance $\{x_1^T, x_3^T, x_2^T, x_4^T\}$. Then it will fit a power-law line on the sorted sequence. After the fitting, assume that the fitted line has values $\{x_1^R, x_3^R, x_2^R, x_4^R\}$. Then the method will use x_i^R as the user expectation for the corresponding value in the target view x_i^T. For example, Figure 5 shows a target view with a reference view generated using this method in which the dashed line is the fitted power-law line. The assumption of this method is that in common cases, the sorted values in a view should follow a power-law line. Therefore, a view with sorted values deviating largely from the fitted power-law line could be interesting.

The main difference between ViewSeeker and above works is that instead of having a predefined reference view as user expectation for each target view, ViewSeeker learns personalized reference views tailored for the current user based on their user expectation feedback on the example views.

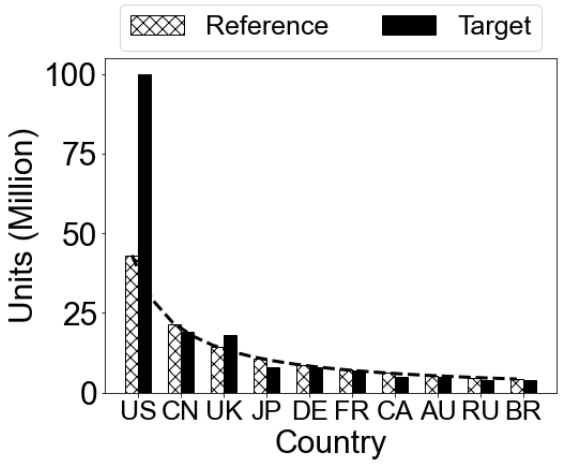


Figure 5: QuickInsights' reference view generation

2.6 External Knowledge Base Utilization Techniques

External knowledge bases such as labels or ratings from other users are commonly used in recommendation systems to provide cold-start recommendation and improve recommendation quality. Several related view recommendation systems that utilize external knowledge bases and their differences with ViewSeeker are discussed below.

Tableau [4] is a popular visual data analysis tool that introduced the view recommendation functionality recently. One of its recommendation methods is based on current trend in user data visualization, such that the system will recommend views on the relevant dataset that are commonly viewed or labeled interesting by other users recently. Similarly, Deepeye [15] leverages previous user ranking labels to train machine learning models for ranking recommendations. Another work REACT [20] uses previous user actions (e.g., group-by queries) to train machine learning models to predict the most suitable individual utility function at a certain data subset. Although above works use the external knowledge base (i.e., previous user preference information) in a variety of ways, the generated/learned ranking functions are still totally predefined and provide no adaptability to the current user's interest. Besides, compared to REACT which tries to learn the most suitable individual utility function, ViewSeeker takes a more general approach and aims to leverage the

knowledge base to learn the most suitable combination among different individual utility functions.

A commonly used technology that provides recommendation with adaptability to the current user's interest is called *collaborative filtering*. Collaborative filtering is an umbrella term covering many techniques that utilize user interest information from other users to predict the current user's interest. Tableau also provides a recommendation method based on collaborative filtering. The method first builds a user-item matrix based on user ratings of items and conducts matrix factorization to generate a user vector for each user and an item vector for each item. Then the user rating for an item could be estimated by matrix multiplication of the corresponding user vector and item vector. This method is able to adjust the recommendation based on the current user's item ratings. However, since the matrix factorization is an expensive operation, especially for large user-item matrices, the adjustment process cannot be *interactive*, namely in the sub-second scale. Actually, the Tableau system only updates the user vectors and item vectors once every day. On the country, ViewSeeker can interactively update view recommendation based on new item ratings from the current user.

Another view recommendation system, called VizRec [23], also offers a recommendation method based on collaborative filtering. In this method, the system first models the current user as a vector based on their ratings of the items, then searches for the nearest neighboring users based on the vector similarity scores, and finally estimates the current user's interest on unlabeled views using the average scores from the neighboring users. ViewSeeker differs from the above method by modeling each user as a vector based on the prediction of a machine learning model trained with the user's item ratings. By building the user vector using the model predictions instead of the raw item ratings, ViewSeeker would be able to discover neighboring users whose ratings do not share common items with the current user's ratings.

VizRec has another recommendation method called *content-based filtering*, which builds user vectors and item vectors based on their semantic meaning and discovers interesting items based on their similarities with the current user's vector. Specifically, the system asks the users to provide tags to the views that describe the content of the views. Example tags for a film dataset could be "movie", "genre", "timeline", "budget", "gross", "profit", etc. Then the system treats the tags of a view as a *document* and treats all the tags that a user has given as a *query*. Afterwards, the system could provide document recommendation (i.e., view recommendation) for the current query

(i.e., current user) using traditional information retrieval techniques, such as TF-IDF embedding, indexing and retrieval [19]. This method is less related to ViewSeeker because its recommendation is based on view tags while ViewSeeker's recommendation is mainly based on view ratings.

2.7 Summary

In this chapter, we provided the background knowledge of the thesis in multiple aspects. We first introduced the data visualization and its representation in the context of structural databases. Then, we talked about different utility function families proposed by previous works and explored in ViewSeeker including usability, usefulness, diversity, and multi-objective utility functions. After that, we discussed related works for the three functionalities of ViewSeeker. For utility function preference learning, we introduced relevant works with different levels of user preference adaptability and explained that ViewSeeker differs from these works by being the most adaptive and interactive model. Afterwards, we briefly talked about the active learning techniques adopted by ViewSeeker. For user expectation learning, we introduced several non-IVR works which use predefined methods to estimate user expectation (i.e., reference views), and explained that ViewSeeker differs from them by taking an interactive approach. Finally, for external knowledge base utilization, we introduced several non-IVR works as well as works with iterative learning traits. Then, we explained the advantages of ViewSeeker over these works such as faster user model update and wider range of usage scenarios.

3.0 Utility Function Preference Learning

The work covered in this chapter was published in 2019 EDBT International Workshop on Big Data Visual Exploration and Analytics [38] and 2020 International Conference on Information Visualisation [36]

In this chapter, we introduce the *utility function preference learning* functionality of ViewSeeker. As opposed to common view recommendation systems which have a predefined utility function, ViewSeeker interacts with the user to learn a personalized utility function that meets the user's need. We will provide the problem definition in Section 3.1. Then, we will introduce our approach in Section 3.2 and Section 3.3. Finally, we will present the experimental evaluation in Section 3.4.

3.1 Interactive Utility Function Preference Learning

We first provide the formal definition of the *interactive utility function preference learning* problem and then introduce our solution.

Definition 2. Interactive Utility Function Preference Learning: Given a database D, all possible views that can be generated from the dataset $V = \{v_1, v_2, ...\}$, and a set of n possible utility functions $U = \{u_1(), u_2(), ..., u_n()\}$, interactively learn a utility function $u()$ that can be any combination of the functions in U, such that the recommendation from V based on $u()$ meets the user's interest.

It can be seen that the difference between traditional view recommendation problem and the above problem is that the utility function $u()$ is predefined in the former, while interactively learned based on user feedback in the latter.

In utility function preference learning, the user interaction is conducted in an iterative fashion, and each iteration has four steps, as shown in Figure 6.

1. **Example Selection:** The system selects one or more examples for user feedback. The example

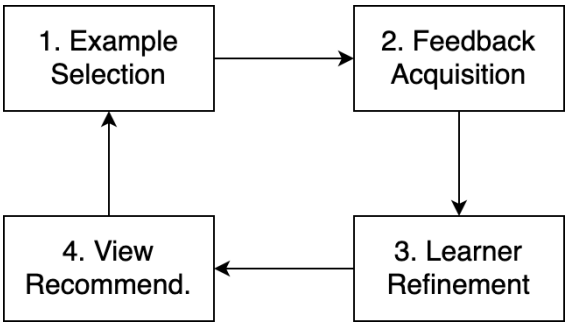


Figure 6: User interaction iteration

can be in a variety of forms based on the user feedback types which will be introduced in the following sections. The example selection strategy can be active learning-based, heuristic-based and so on.

2. **Feedback Acquisition:** The user feedback for the examples are acquired.
3. **Learner Refinement:** Newly acquired user feedback together with all previous feedback are used to refine the learner to improve its prediction performance.
4. **View Recommendation:** The system provides view recommendation among all possible views based on the prediction of the most recent learner.

If the user is satisfied with the view recommendation, then the IVR process stops, otherwise the system will start a new interaction iteration to further improve the recommendation quality.

3.2 User Interaction Methods

3.2.1 Learning Representation

In order to utilize machine learning techniques to learn user preference, we need a learning representation for each view. Since the goal of the preference learning problem is to determine the combination of the individual utility functions in a composite multi-objective utility function $u()$, it is natural to use the individual utility function scores for a view as the features for the view.

More precisely, for a view v, the learning representation will be $\langle u_1(v), u_2(v), ..., u_n(v)\rangle$ where $\{u_1(), u_2(), ..., u_n()\}$ are the n possible individual utility functions. In this thesis, we focus on the utility functions whose scores can be calculated based on a single view. Other utility functions whose scores are based on multiple views (e.g., *diversity* [18]) are not included as candidate utility functions.

Then the prediction of the learner will be a mapping from the features to a score estimating the utility of the view. This mapping serves the role of a utility function $u()$, which combines the individual functions $\{u_1(), u_2(), ..., u_n()\}$ and outputs a utility score $u(v)$ for a view v as shown in Equation 6.

$$u(v) = F(u_1(v), u_2(v), ..., u_n(v)) \tag{6}$$

3.2.2 User Feedback Types

ViewSeeker supports multiple user feedback types, which have their own corresponding example forms, learner types, example selection strategies, and ranking algorithms. We will discuss the first two here and the remaining in following sections.

There are four types of feedback: *binary*, *Likert-scale* [24], *real number*, and *pairwise comparison* [30].

Binary For binary feedback, the example is in the form of a single view. The user is given an example view and a binary option to indicate if the view is interesting or not. The corresponding learner is in the form of a binary classifier.

Likert-scale For Likert-scale feedback, the example is in the form of a single view. The user is given an example view and a Likert scale from 1 to 5 to indicate the interestingness of the view, with 5 being most interesting and 1 being not interesting at all. Since the five levels in the Likert scale feedback has ordinal relation between each other, they could be treated as 5 levels of real number feedback and thus the corresponding learner is in the form of a regression model.

Real number For real number feedback, the example is in the form of a single view. The user is given an example view and a real number option between 0.0 and 1.0 to indicate view interestingness, with 1.0 being most interesting and 0.0 being not interesting at all. The corresponding

learner is in the form of a regression model.

Pairwise comparison For pairwise comparison feedback, the example is in the form of a pair of views. The user is given a pair of example views and a binary option to indicate if the first view in the pair is more or less interesting than the second view. The corresponding learner is in the form of a learning-to-rank model [14]. A learning-to-rank model can be trained with partially ordered view lists, in this case labeled view pairs, and predict a ranking score for each view in a test list.

3.2.3 Query Strategies

ViewSeeker leverages *active learning* techniques for example selection. This selection strategy is called *query strategy* [26]. Recall that we introduced two commonly-used query strategies in the background chapter (Section 2.4), namely *uncertainty sampling* and *query-by-committee*, and these are the query strategies adopted by ViewSeeker.

ViewSeeker uses the uncertainty sampling strategy for the binary feedback type. Recall that this strategy selects the view whose highest posterior probability in any class is the lowest. For binary feedback, this means that the strategy will select the view whose positive probability is closest to 0.5.

On the other hand, ViewSeeker uses the query-by-committee strategy for Likert-scale, real number and pairwise comparison feedback types. For Likert-scale and real number feedback, the strategy selects the view that has the highest variance among committee member predictions. The committee members, in these cases, are regression models.

For pairwise comparison feedback, the committee members are in the form of *learning-to-rank models* and each committee members can predict a ranking score $s(v)$ for each view v in a pair of example views. Therefore, for a view pair $\langle v_1, v_2 \rangle$, a committee member can provide a prediction of the ranking score gap between the two views in the pair, namely $s(v_1) - s(v_2)$. Then, the strategy selects the view pair that has the highest variance among these gap predictions from the committee members as shown in Equation 7.

$$\langle v_1, v_2 \rangle^*_{QBC} = \underset{\langle v_1, v_2 \rangle}{\operatorname{argmax}} Var(\langle s_{c_1}(v_1) - s_{c_1}(v_2), s_{c_2}(v_1) - s_{c_2}(v_2), ..., s_{c_C}(v_1) - s_{c_C}(v_2) \rangle) \quad (7)$$

Table 1: Interaction methods for different feedback types

Feedback type	Example form	Learner type	Query strategy	Ranking score
Binary	Single view	Binary classifier	Least confident	$P_\theta(y = Y\|v)$
Likert-scale	Single view	Regression model	QBC	$s_\theta(v)$
Real number	Single view	Regression model	QBC	$s_\theta(v)$
Pairwise comparison	View pair	Learning-to-rank model	QBC	$s_\theta(v)$

where $Var(\cdot)$ is the variance operator, $s_{c_i}(v)$ is the ranking score of the i-th committee member for view v, and C is the number of committee members,.

3.2.4 Ranking Algorithms

Lastly, we introduce the ranking algorithms of ViewSeeker, which use the learners to generate a ranking score for each view for the purpose of view recommendation. Like the query strategies, ViewSeeker uses different ranking algorithms for different feedback types.

For binary feedback, ViewSeeker uses the positive class probability $P_\theta(y = 1|v)$ of the binary classifier θ as the ranking score for v. For Likert-scale, real number and pairwise comparison feedback types, the learner's prediction $s_\theta(v)$ can be directly used as the ranking score for v. A summary of different interaction methods for different feedback types are listed in Table 1.

3.3 Overall Workflow

The overall workflow for ViewSeeker's utility function preference learning is shown in Algorithm 1. The algorithm takes in a dataset D and outputs a view utility estimator *UE*.

The $GenerateExamples$ procedure (line 1) generates the learning representations for the views and populates the unlabeled example set U with appropriate example forms based on user feedback types. The $AcquireInitLabels$ procedure (line 2) randomly selects several examples for user labeling to populate the labeled example set L. Then L is used to initialize the view utility estimator *UE* and the example informativeness estimator *IE* (lines 3-4). The *UE* is in the form

Algorithm 1 Interactive Utility Function Preference Learning

Require: The dataset D

Ensure: The view utility estimator *UE*

1: Unlabeled example set $U \leftarrow GenerateExamples()$
2: Labeled example set $L \leftarrow AcquireInitLabels()$
3: $UE \leftarrow$ initialize view utility estimator *UE* using L
4: $IE \leftarrow$ initialize example informativeness estimator *IE* using L
5: **loop**
6: Choose an example x from U using *IE*
7: Solicit user label on x
8: $L \leftarrow L \cup \{x\}$
9: $U \leftarrow U - \{x\}$
10: $UE \leftarrow$ refine *UE* using L
11: $IE \leftarrow$ refine *IE* using L
12: $T \leftarrow$ recommend top views using *UE*
13: **if** the user is satisfied with T or the user wants to stop **then**
14: Break
15: **end if**
16: **end loop**
17: Return the most recent *UE*

of a machine learning model, which can be used to predict view interestingness and provide view recommendation. The *IE* is in the form of one or an ensemble of machine learning models, which can be used to conduct query strategy and select example views.

Lines 5 to 16 represents the user interaction loop which contains the aforementioned four steps in each iteration. Line 6 represents the example selection step, in which ViewSeeker leverages the query strategies through *IE* to select examples from U. Line 7 represents the feedback acquisition step, in which ViewSeeker acquires user feedback on the example. Lines 8-11 is the learner refinement step, in which ViewSeeker refines *UE* and *IE* using the latest L. Lines 12-15 is the view recommendation step, in which ViewSeeker uses the most recent *UE* to recommend views, and

Table 2: Testbed Parameters

Total number of records ($\|D\|$)	$100K$ (DIAB), $100K$ (CENSUS)
Average query set size ($\|D_Q\|$)	$43K$ (DIAB), $47K$ (CENSUS)
Number of dimension attributes (A)	7 (DIAB), 5 (CENSUS)
Number of measure attributes (M)	8 (DIAB), 5 (CENSUS)
Number of aggregate functions	5
Total view count	280 (DIAB), 125 (CENSUS)
Number of individual utility functions	8
Feedback type	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
The number of views to recommend (k)	5,10,15,20
Runs for each configuration	10 (with different D_Q)

stops the interaction if the user is satisfied with the recommendation. Finally, the latest trained *UE* will be returned as the algorithm output.

3.4 Experimental Evaluation

In this section, we present the experimental evaluation of ViewSeeker's utility function preference learning functionality. We will introduce the experimental settings in Section 3.4.1 and the experimental results in Section 3.4.2.

3.4.1 Experimental Settings

The experimental testbed was built in Python. This testbed will also be used and extended in future experimental sections (i.e., Section 4.4 and Section 5.3) to evaluate ViewSeeker's other functionalities. The experiments were conducted on a Core i5 server with 8GB of RAM. All the experimental settings are listed in Table 2 and detailed below.

Datasets Two datasets are used in the experiments: the DIAB dataset and the CENSUS dataset.

The DIAB dataset is a real-world dataset of diabetic patients [3]. We removed the attributes that have a large amount of missing data. After preprocessing, the data set has 7 dimension attributes, 8 measure attributes, and approximately 100 thousand records. The CENSUS dataset contains microdata from the U.S. labor force survey [8]. We removed the "not in universe" records and records with zero income. After preprocessing, the dataset has 5 dimension attributes, 5 measure attributes, and approximately 100 thousand records. All measure attributes in the two datasets are normalized to a real number range between 1 and 100.

Query Simulation Since the individual utility functions we used to generate the view features include deviation-based utility functions, we need to simulate the data subset D_Q from the whole dataset D so that we can calculate the deviation scores for each view (D_Q, a, m, f) based on the difference between (D_Q, a, m, f) and (D, a, m, f). We used 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 utility functions. They are 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 are *usability* [7], *accuracy* [7], and *p-value* [29].

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) between the aggregated values and the raw data point values. 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" [12]. 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 view.

All utility function 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 utility function scores.

It should be noted that, in general, users may customize the utility functions, including adding new utility functions, for specific analysis purposes. The current set of utility functions mentioned above are selected as an example to illustrate the effectiveness of ViewSeeker.

User Simulation We simulated different user interests using different *ideal utility functions* (IUFs). Each IUF represents the ground truth utility function according to the user's interest. We designed 11 diverse IUFs that included 3 single-component utility functions and 8 multi-component (i.e., multi-objective) composite utility functions (Table 3). We chose the components in multi-component IUFs carefully such that they represent different characteristics of the candidate views. For example, for IUF 10 in Table 3, EMD measures the absolute differences across the bins, KL-divergence measures the relative entropy between the two distributions, while Usability represents the visual quality of a view, etc. By choosing diverse utility functions, we could also reduce the probability of selecting highly correlated utility functions into one IUF. The weights in each IUF were evenly distributed and were set in a way such that the value range of each IUF is between 0.0 and 1.0.

Evaluation Metrics We evaluated the performance of ViewSeeker in the aspect of *recommendation accuracy*. Specifically, we measured the number of labeled examples needed for ViewSeeker 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 ViewSeeker and the IUF, respectively, the accuracy is calculated as: $\frac{|V^p \cap V^*|}{k}$.

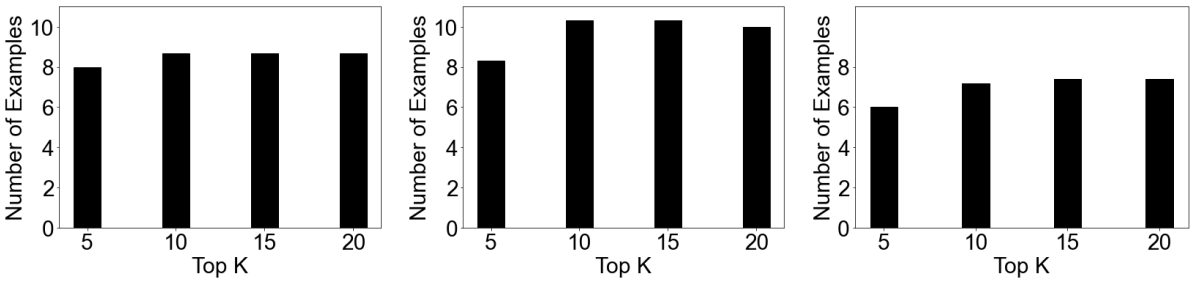
3.4.2 Experimental Results

Figures 7 and 8 show the number of example views that need to be labeled in order for ViewSeeker to reach an 100% accuracy in the top-k recommended views. Here, the x-axis is for k in top-k (i.e., the number of views to be recommended), and the y-axis is the number of required examples for the accuracy to reach 100%.

Specifically, Figures 7(a) and 8(a) are for one-component IUFs (i.e., average result over IUF

Table 3: 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



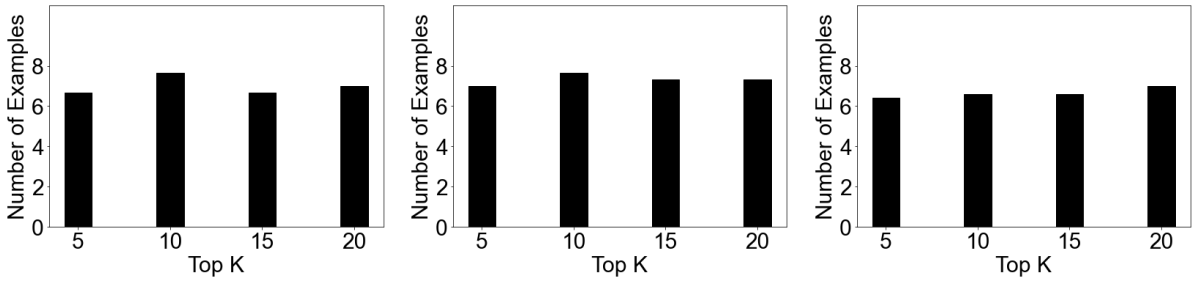
(a) Single-component IUFs (b) Two-component IUFs (c) Three-component IUFs

Figure 7: Recommendation accuracy for DIAB dataset with different IUFs

1-3 in Table 3). Figures 7(b) and 8(b) are for two-component composite IUFs (i.e., average result over IUF 4-6). Finally, Figures 7(c) and 8(c) 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 utility function preference learning functionality.

To the best of our knowledge, currently there are no existing solutions that provide the same



(a) Single-component IUFs (b) Two-component IUFs (c) Three-component IUFs

Figure 8: Recommendation accuracy for the CENSUS dataset with different IUFs.

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 utility functions, as shown in Figure 9. Particularly, while ViewSeeker tries to accurately estimate the ideal utility function for the current user, each of those baselines uses one of the eight predefined individual utility functions as discussed in the experimental settings section (e.g., KL, EMD, L1, etc) with no user interaction. Figure 9 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 utility functions with a small amount of additional user effort.

3.5 Summary

In this chapter, we introduced ViewSeeker's utility function preference learning functionality. We first gave the definition of the interactive utility function preference learning problem. Then, we introduced in detail ViewSeeker's user interaction methods including learning representation, user feedback types, query strategies, and ranking algorithms. After that, we talked about the

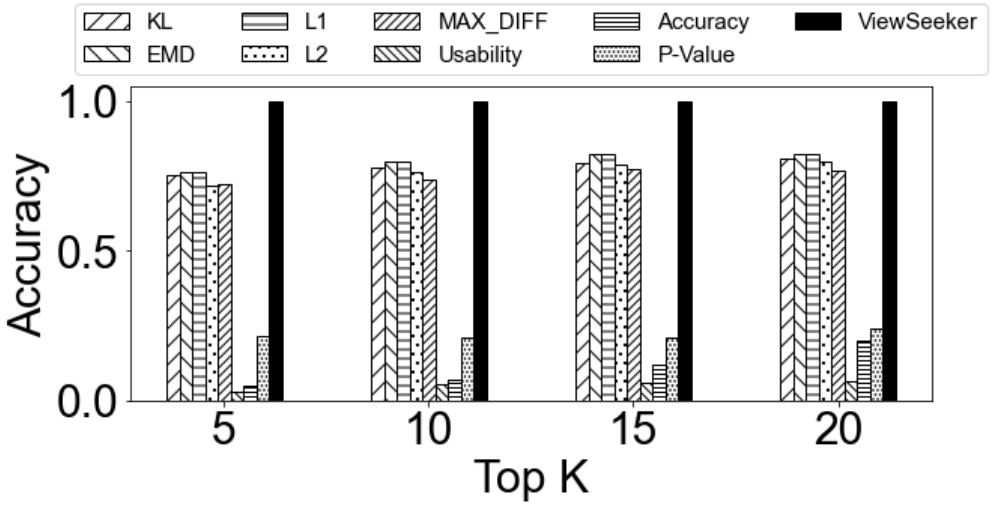


Figure 9: Maximum achievable accuracy by baselines and ViewSeeker

overall iterative learning algorithm utilizing the above interaction methods. For experiments, we tested ViewSeeker on different datasets and different simulated ideal utility functions (IUFs). The experimental results show that ViewSeeker can accurately learn the IUFs with fewer than 10 labels on average. We also showed that ViewSeeker outperforms non-IVR baselines in the form of individual utility functions by at least 24.6% in recommendation accuracy.

4.0 Interactive User Expectation Learning

The work covered in this chapter was published in the Big Data Research Journal - Special Issue on Interactive Big Data Visualization and Analytics, 2021 [39]

In this chapter, we present the second functionality of ViewSeeker, *user expectation learning*, in which the system estimates user expectation of the values in the views to provide view recommendation based on the difference between the estimates and the actual values. We will provide the problem definition in Section 4.1. Then, we will introduce our approach in Section 4.2 and the corresponding time complexity analysis in Section 4.3. Finally, we will present the experimental evaluation in Section 4.4.

4.1 Interactive User Expectation Learning

We will first provide the formal definition of the *interactive user expectation learning* problem and then introduce our solutions.

Definition 3. Interactive User Expectation Learning: Given a database D, all possible views that can be generated from the dataset $V^T = \{v_1^T, v_2^T, ...\}$, interactively learn user expectation $V^R = \{v_1^R, v_2^R, ...\}$ for V^T, such that the recommendation for V^T based on the distances between individual views in V^T and V^R, namely $\{dist(v_1^T, v_1^R), dist(v_2^T, v_2^R), ...\}$ for some distance function $dist$, meets the user's need.

As mentioned in Chapter 2, [32] and [6] use predefined methods to generate V^R when user prior knowledge and expectation about the dataset D is not available. In contrast, interactive user expectation learning brings in user prior knowledge and expectation through user interaction in order to generate a more accurate V^R.

Our solution to the interactive user expectation learning problem is a novel algorithm, called *Expectation Acquisition and Propagation* (EAP). As can be inferred from its name, the algorithm has two stages. The first stage is called *Expectation Acquisition* (EA), in which ViewSeeker elicits

user expectation from example views. The second stage is called *Expectation Propagation* (EP), in which ViewSeeker leverages the user labels to estimate user expectation on unlabeled views. The EP process can be further divided into two parts: *Expectation Propagation across Dimensions* (EPaD) and *Expectation Propagation across Measures* (EPaM). We will introduce them one by one in the following sections.

4.2 The EAP Algorithm

We formally present the EAP algorithm in this section. To facilitate the understanding of the algorithm, we will use an example dataset (Table 4) alongside the algorithm walkthrough to show how the algorithm works on actual data. The dataset 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.

Recall from Section 2.1 that a view v can be represented by a triple (a, m, f) indicating the *dimension*, *measure* and *aggregate function* used in the view. For presentation purposes, in this section, we will denotae a view v with a triple (a, m, f) as $v_{a,m,f}$. Further, in order to distinguish between the original view $v_{a,m,f}$ and the user expectation for that view, we will name the original view as $v^T_{a,m,f}$ (i.e., the target view) and the labeled user expectation for that view as $v^R_{a,m,f}$ (i.e., the reference view). Besides, the superscript T will also be used for any view that is generated directly from the data, while the superscript R will also be used for any view with estimated user expectation.

4.2.1 Expectation Acquisition

During the EA stage, the user expectation on example views are acquired. The example selection strategies will be introduced in later sections. For each example view, the user is asked to indicate if the values in the view match their expectation. If so, the values in the view are recorded as the user's expectation for this view. If not, then the user is asked to indicate their expected values on the view, which will be recorded as is.

Table 4: Example Dataset D

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 5: $v^T_{Edu,WI,AVG}$

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

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

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

For the example dataset, assume that the user is given a target view $v^T_{Edu,WI,AVG}$ (Table 5) and expects that people with PhD education should have a higher wage income. Then they can specify their expectation $v^R_{Edu,WI,AVG}$ as shown in Table 6.

4.2.2 Expectation Propagation across Dimensions

Due to the prohibitively large view space, it is unrealistic to ask the user to label all possible views in the dataset. Therefore, we devised a novel algorithm, called Expectation Propagation (EP), to estimate user expectation on unlabeled views based on existing user labels. ViewSeeker currently supports EP across dimensions and EP across measures, and covers aggregate functions of `COUNT`, `SUM`, and `AVG`.

For EP across dimensions (EPaD), ViewSeeker needs to estimate user expectation on a view with a different dimension as the labeled view, for example, from $v^R_{a_1,m_1,AVG}$ to $v^R_{a_2,m_1,AVG}$ (i.e., from $v^R_{Edu,WI,AVG}$ to $v^R_{Occ,WI,AVG}$ in the example dataset), where $v^R_{a_1,m_1,AVG}$ is the labeled expectation from the user.

The EPaD procedure can be described as a path with three steps:

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

It can be seen that the propagation path will go through views with aggregate function `SUM` (i.e., Steps 1 and 2). The reason for going through Steps 1 and 2 instead of going through Step 3 only is that 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 can not be reflected by a view with an `AVG` aggregate function, namely $v_{a_1,m_1,AVG}$. Therefore, EP needs to multiply the average view by the corresponding counts of the groups to form $v_{a_1,m_1,SUM}$ to capture the impact of the groups.

The details of each step in the propagation path are described below:

Steps 1: $v^R_{a_1,m_1,AVG} \rightarrow v^R_{a_1,m_1,SUM}$

$v^R_{a_1,m_1,AVG}$ can be represented as a vector $[g_1, g_2, ..., g_p]$, where p is the number of groups in dimension a_1. Then, the first step can be described by Equation 8:

$$v^R_{a_1,m_1,SUM} = v^R_{a_1,m_1,AVG} \odot v^T_{a_1,m_1,COUNT} = [g_1c_1, g_2c_2, ..., g_pc_p] \tag{8}$$

where $v^T_{a_1,m_1,COUNT} = [c_1, c_2, ..., c_p]$ is a helper view which contains tuple 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:

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

Steps 2: $v^R_{a_1,m_1,SUM} \rightarrow v^R_{a_2,m_1,SUM}$

First, EP needs to create a helper view with two grouping attributes a_1 and a_2, which can be represented as a matrix as shown in Equation 9.

$$V^T_{a_1,a_2,m_1,SUM} = \begin{bmatrix} s_{11} & \dots & s_{1q} \\ \vdots & \ddots & \vdots \\ s_{p1} & \dots & s_{pq} \end{bmatrix} \tag{9}$$

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 normalizes each row in $V^T_{a_1,a_2,m_1,SUM}$ to get $V^T_{a_1,a_2,m_1,SUM,norm}$ as shown in Equation 10.

$$V^T_{a_1,a_2,m_1,SUM,norm} = \begin{bmatrix} \frac{s_{11}}{G_1} & \dots & \frac{s_{1q}}{G_1} \\ \vdots & \ddots & \vdots \\ \frac{s_{p1}}{G_p} & \dots & \frac{s_{pq}}{G_p} \end{bmatrix} \tag{10}$$

where $G_i = \sum_{j=1}^{q} s_{iq}$ for $i = 1...p$. In other words, G_i is the row sum of row i in the matrix. The corresponding matrix for the example dataset is:

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

Then, EP can generate $v^R_{a_2,m_1,SUM}$ using Equation 11.

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

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

Occupation	AVG(Wage Income)
Engineer	233
Scientist	233

Table 8: $v^T_{Occ,WI,AVG}$

Occupation	AVG(Wage Income)
Engineer	233
Scientist	133

In other words, EP splits each group sum in a_1 by the proportion of each a_2 group therein, then sums up these splits by the groups in a_2. The corresponding calculation for the example dataset is:

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

Steps 3: $v^R_{a_2,m_1,SUM} \rightarrow v^R_{a_2,m_1,AVG}$

Finally, EP generates $v^R_{a_2,m_1,AVG}$ using Equation 12.

$$v^R_{a_2,m_1,AVG} = v^R_{a_2,m_1,SUM} \oslash v^T_{a_2,m_1,COUNT} = [t_1, t_2, ..., t_q] \oslash [c_1, c_2, ..., c_q] = [\frac{t_1}{c_1}, \frac{t_2}{c_2}, ..., \frac{t_q}{c_q}] \quad (12)$$

where $v^T_{a_2,m_1,COUNT} = [c_1, c_2, ..., c_q]$ is another helper view, which contains tuple counts for each group 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 estimated user expectation $v^R_{Occ,WI,AVG}$ is shown in Table 7, and the corresponding target view $v^T_{Occ,WI,AVG}$ generated directly from the dataset is shown in Table 8. 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 because the user could expect the scientist to have a higher wage income.

Now we briefly introduce EPaD between views with other aggregate functions. EPaD between views with the SUM function can be performed using Equation 11, with the difference that $v^R_{a_1,m_1,SUM}$ is directly specified by the user in this case. EPaD between views with the

COUNT function 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., Equation 9 - 11) to COUNT, such that EP proceeds as shown in Equation 13.

$$v^R_{a_2,m_1,COUNT} = v^R_{a_1,m_1,COUNT} V^T_{a_1,a_2,m_1,COUNT,norm} \tag{13}$$

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

To summarize, EPaD leverages the correlation between *dimensions* in the dataset to propagate labeled expectation on example views to other views, such that views that deviate largely from the corresponding expectation estimates would be recommended.

4.2.3 Expectation Propagation across Measures

ViewSeeker also supports EP across measures (EPaM). Again, we will first introduce a propagation example from two views with different measures, such as from $v^R_{a_1,m_1,AVG}$ to $v^R_{a_1,m_2,AVG}$ (i.e., from $v^R_{Edu,WI,AVG}$ to $v^R_{Edu,TI,AVG}$ in the example dataset), where $v^R_{a_1,m_1,AVG}$ is specified by the user. Then, we will briefly introduce EPaM for other aggregate functions.

As we will see, the EPaM process proceeds in the same way for each group g in dimension a_1, so for simplicity reasons, we will focus on an example group g_1 (e.g., PhD) of a_1 in our example. The EPaM process will be from $v^R_{g_1,m_1,AVG}$ to $v^R_{g_1,m_2,AVG}$, and is also divided into three steps:

$$v^R_{g_1,m_1,AVG} \xrightarrow{\text{Step 1}} \Delta_{g_1,m_1,AVG} \xrightarrow{\text{Step 2}} \Delta_{g_1,m_2,AVG} \xrightarrow{\text{Step 3}} v^R_{g_1,m_2,AVG}$$

The details of each step in the propagation path are described below:

Steps 1: $v^R_{g_1,m_1,AVG} \rightarrow \Delta_{g_1,m_1,AVG}$

EP calculates the difference between user expectation $v^R_{g_1,m_1,AVG}$ and the original view $v^T_{g_1,m_1,AVG}$ as shown in Equation 14:

$$\Delta_{g_1,m_1,AVG} = v^R_{g_1,m_1,AVG} - v^T_{g_1,m_1,AVG} \tag{14}$$

For the example dataset, we can read the third row from Table 5 and Table 6 and conduct the

corresponding calculation as follow:

$$\Delta_{PhD,WI,AVG} = v^R_{PhD,WI,AVG} - v^T_{PhD,WI,AVG} = 300 - 150 = 150$$

Steps 2: $\Delta_{g_1,m_1,AVG} \rightarrow \Delta_{g_1,m_2,AVG}$

EP propagates $\Delta_{g_1,m_1,AVG}$ to $\Delta_{g_1,m_2,AVG}$ based on the correlation between measure m_1 and m_2 in g_1 as shown in Equation 15:

$$\Delta_{g_1,m_2,AVG} = \Delta_{g_1,m_1,AVG} \div \sigma_{g_1,m_1} \times r_{g_1,m_1,m_2} \times \sigma_{g_1,m_2} \tag{15}$$

where σ_{g_1,m_1} is the standard deviations of m_1 in g_1, r_{g_1,m_1,m_2} is the Pearson correlation coefficient between m_1 and m_2 in g_1, and σ_{g_1,m_2} is the standard deviations of m_2 in g_1.

In other words, EP first normalizes $\Delta_{g_1,m_1,AVG}$ by σ_{g_1,m_1} to make it dimensionless, then EP converts this dimensionless value to another dimensionless value for m_2 based on the correlation between m_1 and m_2, and finally EP multiplies the second dimensionless value by σ_{g_1,m_2} to form $\Delta_{g_1,m_2,AVG}$. The corresponding calculation for the example dataset is:

$$\Delta_{PhD,TI,AVG} = \Delta_{PhD,WI,AVG} \div \sigma^T_{PhD,WI} \times r^T_{PhD,WI,TI} \times \sigma^T_{PhD,TI} = 150 \div 10 \times 1.0 \times 10 = 150$$

Steps 3: $\Delta_{g_1,m_2,AVG} \rightarrow v^R_{g_1,m_2,AVG}$

In this steps, EP adds $\Delta_{g_1,m_2,AVG}$ to $v^T_{g_1,m_2,AVG}$ to form the user expectation estimate $v^R_{g_1,m_2,AVG}$ as shown in Equation 16:

$$v^R_{g_1,m_2,AVG} = v^T_{g_1,m_2,AVG} + \Delta_{g_1,m_2,AVG} \tag{16}$$

The corresponding calculation for the example dataset is:

$$v^R_{PhD,TI,AVG} = v^T_{PhD,TI,AVG} + \Delta_{PhD,TI,AVG} = 165 + 150 = 315$$

After estimating user expectation for other education groups, EP would generate the reference view $v^R_{Edu,TI,AVG}$ as shown in Table 9, and the corresponding target view $v^T_{Edu,TI,AVG}$ is shown in Table 10. We see that the average total income for PhD in $v^T_{Edu,TI,AVG}$ (i.e., 165) is much lower

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

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

Table 10: $v^T_{Edu,TI,AVG}$

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

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 aggregate function SUM can be done in a similar fashion as shown in Equation 17:

$$v^R_{g_1,m_2,SUM} = v^T_{g_1,m_2,SUM} + (v^R_{g_1,m_1,SUM} - v^T_{g_1,m_1,SUM}) \div \sigma_{g_1,m_1} \times r_{g_1,m_1,m_2} \times \sigma_{g_1,m_2} \tag{17}$$

where $v^R_{g_1,m_1,SUM}$ is specified by the user.

EP between views with aggregate function COUNT directly copies the user expectation, because changing the measure shouldn't affect user expectation on the tuple count in the group:

$$v^R_{g_1,m_2,COUNT} = v^R_{g_1,m_1,COUNT} \tag{18}$$

where $v^R_{g_1,m_1,COUNT}$ is specified by the user.

To summarize, EPaM leverages the correlation between *measures* in the dataset to propagate labeled expectation on example views to other views, such that views that deviate largely from the corresponding expectation estimates would be recommended.

4.2.4 Example Selection Strategy

As seen from previous sections, Expectation Propagation (EP) can be done across both dimensions and measures, which means that ideally, EP would be able to estimate user expectation for all target views by asking the user to label one target view for each aggregate function. Therefore, the strategy to select the example view for labeling in the Expectation Acquisition (EA) stage becomes very important to the effectiveness of the EAP algorithm.

The example selection strategy involves two parts: *dimension selection* and *measure selection*. For dimension selection, EA selects the dimension that has the largest group count among all dimensions (Equation 19) under the assumption that a target view with more groups has the potential to elicit more useful information from user expectation labeling:

$$a^* = \underset{a}{\operatorname{argmax}}(|a|) \tag{19}$$

where a^* is the selected dimension and $|\cdot|$ is the group count operator. If multiple dimensions have the same largest group count, EA selects the first dimension based on the attribute order in the table. For measure selection, EA selects the first measure based on the attribute order in the table. Based on the selected dimension and measure, EA will be able to select an example view for each aggregate function.

4.2.5 Overall Workflow

Algorithm 2 describes the overall workflow of ViewSeeker's interactive user expectation learning using the EAP algorithm. Given a dataset D, the algorithm interacts with the user to estimate user expectation V^R on the views V^T that can be generated by D.

First, the $GenerateViews$ procedure generates all possible views of D. Then, for each aggregate function in `COUNT`, `SUM` and `AVG`, EAP uses the example selection strategy to select an example, and uses EPaD and EPaM to estimate user expectation for all views with that aggregate function.

The estimated user expectation V^R can then be used to create a ranking function based on deviation between individual views in V^T and V^R to recommend views in V^T. The learned ranking

Algorithm 2 Interactive User Expectation Learning

Require: A dataset D with set of dimensions A and set of measures M

Ensure: User expectation estimates (i.e., reference view set) $V^R = \{v_1^R, v_2^R, ...\}$

Target view set $V^T = \{v_1^T, v_2^T, ...\} \leftarrow GenerateViews()$
Reference view set $V^R \leftarrow \{\}$
for all $f \in \{COUNT, SUM, AVG\}$ **do**
  Select an example $v_{a,m,f}^T$ from V^T for expectation labeling to get $v_{a,m,f}^R$
  $V^R \leftarrow V^R \cup \{v_{a,m,f}^R\}$
  for all $a' \in A$ **do**
    if $a' \neq a$ **then**
      Expectation propagation from $v_{a,m,f}^R$ to $v_{a',m,f}^R$
      $V^R \leftarrow V^R \cup \{v_{a',m,f}^R\}$
    end if
    for all $m' \in M$ **do**
      if $m' \neq m$ **then**
        Expectation propagation from $v_{a',m,f}^R$ to $v_{a',m',f}^R$
        $V^R \leftarrow V^R \cup \{v_{a',m',f}^R\}$
      end if
    end for
  end for
end for
Return V^R

function can also be used as an individual utility function in the utility function preference learning process.

Table 11: EAP Algorithm Complexity

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

4.3 Runtime Complexity Analysis

To analyze the runtime complexity of the EAP algorithm as described in Algorithm 2, we divide the EAP algorithm into four parts and evaluate the time complexity in terms of number of operations for each part. The cost for each part is summarized in Table 11. The meanings of the symbols in the table are: A is the number of dimension attributes, G is the average number of groups in each dimension attribute, M is the number of measure attributes, F is the number of aggregate functions, and N is the number of records in the dataset.

For the *view generation* part, ViewSeeker needs to scan the dataset to generate a view for each (a, m, f) triple, which results in a complexity of $AMFN$. 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 $AAMFN$. ***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 $AMFG$.

4.4 Experimental Evaluation

We conducted two sets of experiments in our evaluation. The first set evaluates the effectiveness of ViewSeeker's user expectation learning functionality. The second set evaluates the

Table 12: Testbed Parameters

Total number of records ($\|D\|$)	$100K$ (DIAB), $100K$ (CENSUS)
Average query set size ($\|D_Q\|$)	$36K$ (DIAB), $39K$ (CENSUS)
Average reference set size ($\|D_R\|$)	$36K$ (DIAB), $39K$ (CENSUS)
Number of dimension attributes (A)	7 (DIAB), 5 (CENSUS)
Number of measure attributes (M)	8 (DIAB), 5 (CENSUS)
Number of aggregate functions	3 (COUNT, SUM, AVG)
Total view count	168 (DIAB), 75 (CENSUS)
Number of individual utility functions	8
Feedback type	Real number
View interestingness estimator	Linear regressor
Example informativeness estimator	Query-by-committee
Number of views presented per iteration	1
Evaluation metrics	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 and D_R)

combined effectiveness of ViewSeeker's user expectation learning and utility function preference learning functionalities.

4.4.1 Experimental Settings

The experimental testbed was extended from the previous testbed in Section 3.4. It was created in Python and the experiments were conducted on a Core i5 server with 8GB of RAM. All experimental parameters are listed in Table 12 and detailed below.

Datasets Two datasets are used in the experiments: the DIAB dataset [3] and the CENSUS dataset [8]. For the DIAB dataset, we removed the attributes that have a large amount of missing data. After preprocessing, the data set has 7 dimension attributes, 8 measure attributes, and approximately 100 thousand records. For the CENSUS dataset, we removed the "not in universe" records and records with zero income. After preprocessing, the dataset has 5 dimension attributes, 5 measure attributes, and approximately 100 thousand records. All measure attributes in the two datasets are normalized to a real number range between 1 and 100.

User Simulation for User Expectation Learning 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 $|D_Q| = |D_R| < |S|$ 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 target view v_i^T described as (D_Q, a, m, f), an *ideal reference view* v_i^{IR} described as (D_R, a, m, f) could be generated that represents 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 Utility Function Preference Learning In the second set of experiments, we used the same individual utility functions and ideal utility functions (IUFs) as those in Section 3.4.1 (reiterated in Table 13) to simulate the user. The ground truth scores for deviation-based utility functions are calculated based on the target views (v_i^T) and the corresponding ideal reference views (v_i^{IR}).

State-of-the-art Models We compare ViewSeeker with two state-of-the-art view recommendation works SeeDB [32] and QuickInsights [6]. As mentioned in Section 2.5, the two models have predefined methods to estimate user expectations (i.e., reference views). SeeDB estimates user expectation of (D_Q, a, m, f) with a reference view (D, a, m, f). Since this approach utilizes the information in the whole dataset to generate the reference view, we call it the *Global* approach.

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

QuickInsights uses the fitted power-law line on target view (D_Q, a, m, f) as the reference view. Since this approach only uses the information in the target view to generate the reference view, we call it the *Local* approach.

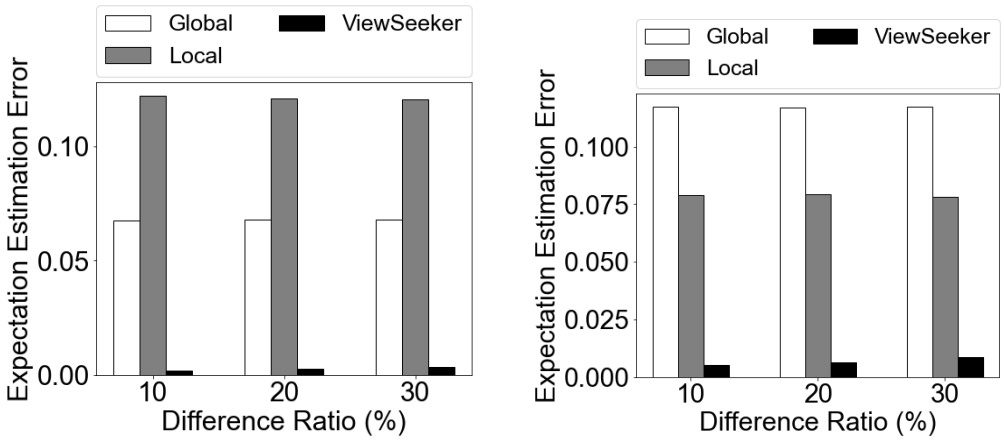
It can be seen that, for a target view v_i^T, each model could use its own approach to generate the corresponding reference view v_i^R, and then calculate its own score for the deviation-based utility functions based on Equation 20:

$$s_i = Dist(P(v_i^T), P(v_i^R)) \tag{20}$$

where $Dist(\cdot, \cdot)$ is certain distance function and $P(\cdot)$ is the view normalization operator. What $P(\cdot)$ does is that it normalizes the values in a view such that the values after normalization add up to one. This operation is in place to offset the effect of the absolute magnitude of the values in the view, such that the distance function focuses on the shape (i.e., distribution) of the values.

Evaluation Metrics We evaluated ViewSeeker's performance in multiple aspects:

- We used *expectation estimate error (EEE)* to measure system effectiveness in user expectation learning.
- We used *system runtime* to measure system efficiency in user expectation learning.



(a) DIBA dataset

(b) CENSUS dataset

Figure 10: EEE comparison for different reference generation methods.

- We used *recommendation accuracy* to measure the system effectiveness in the combined process of user expectation learning and utility function preference learning.

The definition of EEE is 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 21.

$$EEE_i = Dist(P(v_i^R), P(v_i^{IR})) \tag{21}$$

The $L1$-norm was used as the distance function 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 user expectation learning process as mentioned in Section 4.3. 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. We measured the accuracy after 10 labeled examples when the model learning stabilized for most of the configurations.

4.4.2 Experimental Results

Expectation Estimate Error Figure 10 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. Meanwhile, 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 user expectation learning process is shown in Table 14. We used the CENSUS dataset in the experiments. We have tested different data dimensionality by using different numbers of dimension attributes A and different numbers of measure attributes M. We simulated different data dimensionality by duplicating dimension and measure attributes of the dataset. The result was averaged over all possible example view configurations.

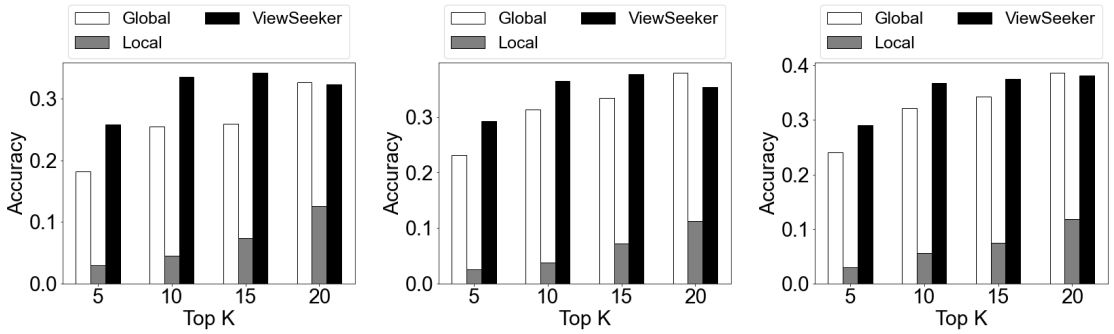
The first and second parts are view generation and helper view/matrix generation, which could be offline calculation. The third and four parts combined is the interactive EP process, which requires interactive system response time.

The runtime result verified our complexity analysis in Table 11 in Section 4.3, 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 datasets with hundreds of dimension and measure attributes on the testbed machine.

Recommendation Accuracy Figures 11 and 12 show the view recommendation accuracy comparison results of the combined process. For the DIAB dataset, the three models have achieved an average 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

Table 14: Runtime in User Expectation Learning (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%

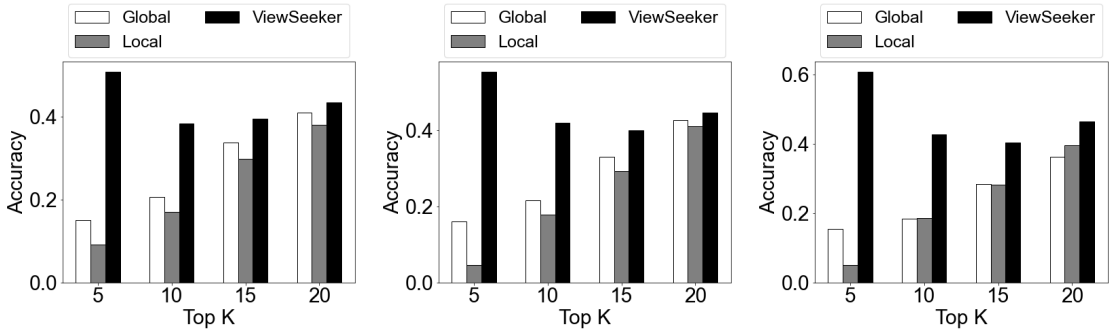
Figure 11: Recommendation Accuracy comparison for DIAB dataset with various difference ratios between D_Q and D_R.

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. Meanwhile, ViewSeeker achieved a relative accuracy improvement of 69.6% against Global and 95.5% against Local for the CENSUS datasets.

The user effort in the user expectation learning process is one example view for each aggregate function and the number of groups for user 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 user expectation learning process 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 ef-



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

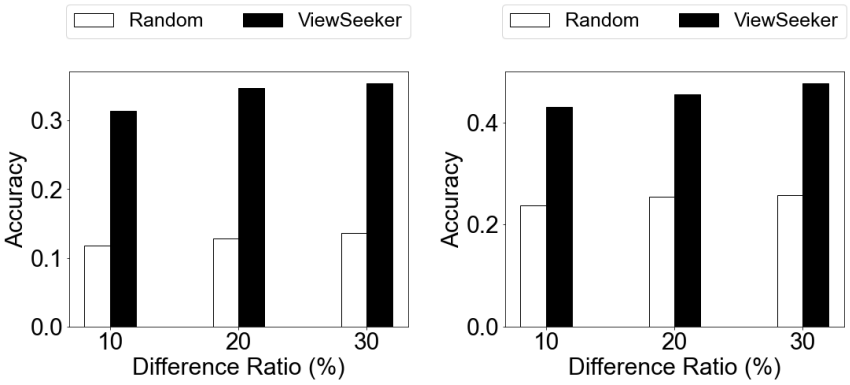
Figure 12: Recommendation Accuracy comparison for CENSUS dataset with various difference ratios between D_Q and D_R.

fectiveness of ViewSeeker's example selection strategy in the user expectation learning process. As discussed in Section 4.2.4, for each aggregate function, ViewSeeker selects the view with the dimension attribute that has the highest group count as the example view. We compare our designed example selection strategy with a baseline strategy, which selects a random example view for each aggregate function.

Figure 13 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 example selection strategy in selecting informative examples in the user expectation learning process.

4.5 Summary

In this chapter, we introduced ViewSeeker's user expectation learning functionality. We first gave the definition of the interactive user expectation learning problem. Then, we discussed in detail the Expectation Acquisition and Propagation (EAP) algorithm including the expectation ac-



(a) DIAB Dataset (b) CENSUS Dataset

Figure 13: Recommendation accuracy comparison for different query strategies in user expectation learning process.

quisition stage, expectation propagation across dimensions and measures, as well as the example selection strategies. After that, we talked about the overall EAP algorithm utilizing the above components. Besides, we also conducted runtime complexity analysis of the EAP algorithm. We compared ViewSeeker with two state-of-the-art non-IVR models on different datasets. The experimental results indicate that, on average, ViewSeeker can reduce the expectation estimation error from the comparison models by more than 90% with user labeling effort on only three example views. When combined with a downstream utility function preference learning process, ViewSeeker also outperforms the comparison models by 83.1% relatively in average recommendation accuracy. Finally, we showed that our example selection strategy has an average relative accuracy improvement of at least 81.5% over a baseline random strategy.

5.0 External Knowledge Base Utilization

The work covered in this chapter was published in 2019 IEEE International Conference on Collaboration and Internet Computing [35]

In this chapter, we will discuss the third functionality of ViewSeeker, *utilizing external knowledge bases* to further reduce user effort in utility function preference learning. We will introduce the concept of external knowledge base in Section 5.1. Then, we will introduce our approach in Section 5.2. Finally, we will present the experimental evaluation in Section 5.3.

5.1 External Knowledge Bases

In the context of our work, we refer to an external knowledge base as any data source that can provide information about user preference over views in some dataset. An example of such knowledge bases is called *data analysis session logs* [20], which record analyst actions during data analysis sessions.

Each *session* contains the actions of a user for a dataset in one sitting. There are two commonly-used *actions*: filter and group-by. A *filter* action changes the underneath dataset that will be used to generate the views, whereas a *group-by* action generates a view containing grouped-by aggregated values (as described in Section 2).

Assume that the dataset has three attributes a, b, and c. An example session on the dataset can be represented as a directed acyclic graph (DAG) as illustrated in Figure 14. In the graph, each edge represents a filter action and each node represent a result data subset (or *result* for short). For example, at the start, the *initial result* A represents the whole dataset. After a filter action with condition $a = 1$, the session lands on result B. The user has the option to backtrack to a previous result and take an alternative filter action. For example, after landing on result C, the user could backtrack to A and take another filter action with condition $b = 1$ to land on D.

Each result can be represented by the filter conditions on the path between that result and the initial result. For example, result C can be represented by a set of two conditions $\{a = 1, b = 1\}$.

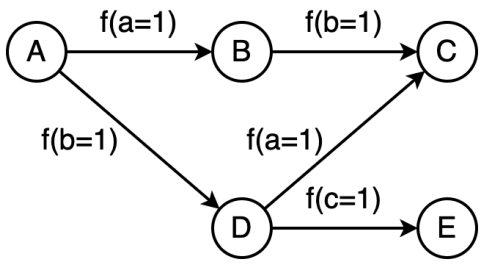


Figure 14: An example session with filter actions

Table 15: Example data analysis session log records

Action ID	Action type	Action parameters	Parent result ID	Child result ID
29	filter	tcp_stream = 0	27	32
30	filter	tcp_stream > 0	27	33
31	group	MIN(ip_src) by tcp_stream	33	34

A result can be reached by different paths as long as they share the same filter conditions. For example, result C can be reached by path $A \rightarrow B \rightarrow C$ and path $A \rightarrow D \rightarrow C$, because they share the same filter condition set $\{a = 1, b = 1\}$. At each result, the user can take group-by actions to generate views (not depicted in Figure 14 for brevity reasons).

Table 15 shows some example data analysis session log records from the REACT-IDA data analysis dataset [21], which will also be used in our experiments. The records represent a session with three actions and four results. First, the user issued a filter action 29 from the initial result 27 to reach a new result 32. Then, the user backtracked to the initial result and issued another filter action 30 to land on result 33. Finally, the user issued a group-by action 31 at result 33 to reach result 34.

5.2 External Knowledge Base Utilization

In this section, we present an extension to ViewSeeker, which incorporates the original utility function preference learning process (Chapter 3), and an additional generic model library building and retrieval process, in order to further reduce user effort during the IVR process. The extended system operates in two stages. In the *offline processing stage*, it leverages the data analysis session logs to build a generic model library. In the *online interaction stage*, it retrieves an offline model from the library to initialize the learners and refines the learners during the IVR process.

5.2.1 Offline Processing Stage

In this stage, our goal is to build a generic model library based on the information in the data analysis session logs. There are two main tasks in this stage. First, we need to create the generic model library structure. Second, we need to fill the model library with generic models.

5.2.1.1 Generic Model Library

Our designed model library has two dimensions: *result* and *feedback*. A value in the *result* dimension represents a specific result in a specific dataset. A value in the *feedback* dimension represents a specific feedback type that will be used in the IVR process. For example, if the IVR process will use real number feedback on the whole dataset, then the corresponding result value is the whole dataset (i.e., an empty filter set) and the feedback value is real number feedback. The unique combination of a result value and a feedback value defines a unique library *cell*, which could hold the generic model for that result and feedback.

5.2.1.2 Model Training Method

In the second task, we need to train a generic model for each library cell. In order to do so, we need to first convert the users' group-by actions at a specific result r into the corresponding feedback type. We will use the *real number* feedback as an example to explain this conversion process.

Ideally, if the information about previous user actions for a specific result r is abundant, we can use for example the counts of the following group-by actions from all users as the real number labels for the corresponding views generated by the group-by actions. However, when such information is scarce, for example when there is only one or two following group-by actions from all users for a specific result r, then the above method would not be able to train a very useful model, because the model would very likely predict almost all views as uninteresting. Therefore, we need to design a conversion process that targets such scarce training data situations.

Without loss of generality, we first look at the scenario when only one subsequent group-by action exists for a specific result r. Since the user chose the view v_1 generated by the group-by action among all possible views, we assume that v_1 should have a higher utility score than other views (e.g., v_2) based on the user's interest. If we assume that the utility function based on the user's interest (i.e., ideal utility function) is a linear combination of the individual utility functions as shown in Equation 22:

$$u^*() = \beta_1 u_1() + \beta_2 u_2() + \cdots + \beta_n u_n() \tag{22}$$

where $\{u_1(), u_2(), ..., u_n()\}$ are the individual utility functions and $\{\beta_1, \beta_2, ..., \beta_n\}$ are the corresponding weights, then the above assumption essentially means that:

$$\beta_1 u_1(v_1) + \beta_2 u_2(v_1) + \cdots + \beta_n u_n(v_1) > \beta_1 u_1(v_2) + \beta_2 u_2(v_2) + \cdots + \beta_n u_n(v_2) \tag{23}$$

Our next step would be to determine the weights of u^* that would satisfy the above condition and afterwards use them to estimate the utility of all views as a way to estimate the real number labels for the views. We use the following example to illustrate what properties the weights of u^* should have in order to satisfy the above condition.

Assume that the utility measures (i.e., utility function scores) for the chosen view v_1 and the unchosen view v_2 satisfy the below conditions:

$$u_1(v_1) = u_2(v_2) > u_2(v_1) = u_1(v_2)$$
$$u_i(v_1) = u_i(v_2) \quad \text{for} \quad i = 3, \ldots, n$$

In other words, v_1 and v_2 have same utility measures except for u_1 and u_2, where v_1 has a high u_1 value and v_2 has an equally high u_2 value, while v_1 has a low u_2 value and v_2 has an equally low

u_1 value. Then, if $\beta_1 = \beta_2$, v_1 and v_2 would have the same utility score. Therefore, in order for v_1 to have a higher utility score than v_2, β_1 needs to be larger than β_2.

The above example gives us an important hint about how to discover the weights $\{\beta_1, \beta_2, ..., \beta_n\}$ of u^*. That is, the weights should put more emphasis on the utility measures that have high values for the chosen view v and put less emphasis on the utility measures that have low values for v.

A simple way to estimate such weights for u^*, that complies with the above rules, is to set the weights $\{\beta_1, \beta_2, ..., \beta_n\}$ to be equal to the values of the individual utility measures $\{u_1, u_2, ..., u_n\}$ for the chosen view v, and this is the method we used to estimate the weights of u^*. In the cases when multiple subsequent group-by actions exist for a result r, we use the average utility measures of the corresponding views as the weights.

One thing to note is that different individual utility measures $\{u_1, u_2, ..., u_n\}$ might have different distributions and scales, which could negatively affects the effectiveness of the weight-discovery approach above. So, we adopt a two-stage normalization approach to alleviate the issue. We first use the Box-Cox transformation [5] to transform the distributions of the individual utility measures into normal shapes. Then we use 0-1 scaling to ensure that all distributions have a similar scale.

After determining the weights of u^*, we could use them to estimate the real number labels for the views that could be generated from result r and train a machine learning model based on the labels. The model, in turn, will be stored in the model library cell corresponding to the result r and real number feedback.

5.2.2 Online Interaction Stage

This stage serves the online IVR process and has two main tasks. The first task is to retrieve a generic model from the library based on the required result and feedback. The second task is the continuous refinement of the learners during the IVR process.

5.2.2.1 Result Representation

Under scarce training data situations, there could be no previous user actions available for a specific result of interest. We will use nearest neighbor search in this situation to find results that

are similar to the target result and retrieve generic models from the neighbors' cells. In order to do so, we need to design vector representations for the results.

A result r is essentially a record table RT_r containing raw data records that are produced by the collection of all filtering actions on the path between the current result r and the root result r_0. If there are no such filtering actions, then the record table contains essentially the whole dataset.

In order to produce the feature vector that represents the record table, the simplest way is to use the raw record values in the record table as the representation. However, the drawback of such an approach is apparent. First, the record table might contain a large number of data records, so the size of the feature would become very large. Using such a large representation will most likely slow down the model training and inference, which is not suited for interactive data analysis scenarios where fast machine response is desired. Second, using such a large representation might not even be feasible under various resource restrictions.

In light of the above observation, we opted for using meta-data of the records to form the representation of the record table and used three features that capture different characteristics of the records in the record table. The first feature is the *entropy of the data*, which measures the amount of information contained in the data. The second feature is the *number of unique values in the data*, which measures the value diversity in the data. The third feature is the *number of null values in the data*, which captures the data availability information. We calculate the three features for each attribute (i.e., column) of the current record table RT_r and concatenate them to form the feature representation for RT_r.

5.2.2.2 Library Model Retrieval

As mentioned in the previous section, each result has a vector representation. Therefore, we could use the vector representation of the current result r and certain similarity metric to search for similar results and retrieve the generic models from the corresponding library cells.

However, since the IVR process only needs one learner instead of multiple learners retrieved above, we need to find a way to merge the retrieved models. We will use the *linear regressor* as an example model type to describe our model merging process.

The linear regressor model's prediction is a weighted sum of the features as shown below:

$$y = w_0 + w_1 x_1 + \cdots + w_n x_n \tag{24}$$

One simple way to combine multiple linear regression models is to take the average weight $\overline{w}_i$ among all models for each wight w_i and assign them to a new linear regressor, and this is the merging method we used. Merging processes for other model types could be conducted in a similar way when applicable.

It should be noted that since the above search method is dataset agnostic (i.e., no explicit dataset information is encoded in the result representation), it can also be used to search for candidate results that are similar to the target result from data analysis session logs across multiple datasets at the same time.

5.2.2.3 Online Model Refinement

The overall workflow for online model refinement is described in Algorithm 3. Firstly, the retrieved generic model M would be used to initialize the weights of the view utility estimator *UE* and the example informativeness estimator *IE* (Line 1-2). Then we need to generate learning representations for all views at result r using $GenerateExamples()$. For each view, the learning representation is the the vector representation containing the individual utility measures. Each utility measure will go through the same normalization process as in the offline stage. The generated learning representations of the views are then stored in the unlabeled example set U as unlabeled views (Line 3). Besides, the labeled example set is initialized as an empty set (Line 4).

What follows is the user interaction iterations, which has been introduced in Chapter 3 and will be briefly reiterated here. Each user interaction iteration has four steps in each iteration. Step 1 selects the example view from U using *IE* (Line 6). Step 2 elicits user feedback (i.e., label) on the example view (Line 7). Step 3 refines *UE* and *IE* using the latest labeled view set L (Line 10-11). The refinement uses iterative optimization methods such as stochastic gradient descent. Step 4 uses the latest *UE* predict view interestingness and provide view recommendation (Line 12). If the user is satisfied with the recommendation, then the IVR process stops and the latest *UE* is returned as the algorithm output. Otherwise, the system will start a new user interaction iteration.

Algorithm 3 ViewSeeker Online Model Refinement

Require: The current result r, the retrieved generic model M

Ensure: The view utility estimator *UE*

1: $UE \leftarrow$ initialize view utility estimator *UE* using M
2: $IE \leftarrow$ initialize example informativeness estimator *IE* using M
3: Unlabeled example set $U \leftarrow GenerateExamples(r)$
4: Labeled example set $L \leftarrow \emptyset$
5: **loop**
6: Choose an example x from U using *IE*
7: Solicit user label on x
8: $L \leftarrow L \cup \{x\}$
9: $U \leftarrow U - \{x\}$
10: $UE \leftarrow$ refine *UE* using L
11: $IE \leftarrow$ refine *IE* using L
12: $T \leftarrow$ recommend top views using *UE*
13: **if** the user is satisfied with T or the user wants to stop **then**
14: Break
15: **end if**
16: **end loop**
17: Return the most recent *UE*

5.3 Experimental Evaluation

In this section, we present the experimental evaluation of ViewSeeker's external knowledge base utilization functionality. We will introduce the experimental settings in Section 5.3.1 and the experimental results in Section 5.3.2.

Table 16: Testbed Parameters

Total number of records	$23,369$
Dimension attribute count	11
Measure attribute count	1
Aggregation function count	5
Possible view count for each result r	55
Individual utility measure count	4
Learner model type	Linear regressor
Generic model retrieval method	Nearest neighbor
Query strategy	Active search
Performance measurement	Top-k accuracy
Baseline Models	Offline, Online
Recommend view count (k)	5,6,7,8,9,10
Offline ratio	0.1,0.2,0.3,0.4
Runs for each configuration	5

5.3.1 Experiment Settings

The experimental testbed was extended from the testbed described in Section 3.4. It was created in Python and the experiments were conducted on a Linux server with 32 Intel® Xeon® CPU E5-2650 v2 cores and 96GB RAM. All experimental parameters are listed in Table 16 and detailed below.

Dataset The dataset we used in our study is the REACT-IDA dataset [21], a benchmark dataset that contains real user analysis sessions for cybersecurity logs. The dataset contains 454 sessions with 2459 actions and 2913 corresponding results. There are 11 dimension attributes and one measure attribute. We used five aggregation functions: count, sum, average, max, and min. Therefore, there are totally 55 possible views for each result r.

We split the dataset into two sets: an offline set and an online set. The offline set is used to build the generic model library. The online set is used to simulate the IVR sessions which will retrieve generic models from the model library. We define the size ratio between the offline set and the whole dataset as the *Offline Ratio* (OR) that will serve as a configuration parameter in the experiments. Generally speaking, the larger the OR, the more beneficial the generic models should be, because they have captured more user interest information

Generic Models We created a library cell (i.e., generic model) for each result r that is reached in the dataset. The feedback type for the library cells is real number feedback and the model type is linear regressor. We skipped the initial result r_0, because it has too many subsequent group-by actions, namely the user interests are very diverse at r_0. Therefore, the user interest captured by a generic model at r_0 would be too general and not very beneficial to the IVR process in the online stage.

Model Retrieval We used the L2 distance to calculate the similarity between the result representations and select the top 10 results (i.e., library cells) that are most similar to the current result r to retrieve the generic models.

Individual Utility Functions We used four different individual utility functions (i.e., utility measures) to generate view representation that capture different characteristics of a view:

- *Diversity (D)* measures the degree of differences in the view values.
- *Concentration (C)* measures the degree of uniformity in the view values.
- *Outlier Score (O)* would be high if a small group of data has a value that is different from the majority of the data.
- *Usability (U)* describes how easily a view can be perceived and understood by the user.

The definition of the utility measures are described in Table 17, in which m is the number of group-by bins, p_j is the normalized frequency for each bin, $\overline{q}$ is the average value among all p_j's, and $H(\cdot)$ is the entropy function.

Query Strategy In our study, we adopt the *active search* approach [9] as the query strategy to select example views. Active search selects the view that has the highest predicted utility by the *UE* as the example view.

Table 17: Individual Utility Measures

Measure	Formula	Reference
Diversity (D)	$\sum_{j=1}^{m} p_j^2$	[28]
Concentration (C)	$1 + H(\frac{p_j+\bar{q}}{2}) - \frac{\log_2 m - H(p_j)}{2}$	[10]
Outlier Score (O)	See [13]	[13]
Usability (U)	$1 - \frac{\min(\log m, c)}{c}$	[25]

Table 18: Average number of IUF in the online stage (for each result r)

OR = 0.1	OR = 0.2	OR = 0.3	OR = 0.4
1.83	1.80	1.75	1.78

Ideal Utility Functions (IUFs) We assume that the IUF $u^*()$ is a linear combination of the individual utility measures as shown in Equation 25.

$$u^*() = \beta_1 \cdot D + \beta_2 \cdot C + \beta_3 \cdot O + \beta_4 \cdot U \tag{25}$$

We used the same approach as described in the offline stage (Section 5.2.1.2) to discover the weights of the IUF u^*_r for certain result r as the ground truth user feedback function. Table 18 shows the average number of distinct ground truth IUFs u^*_r for each result r in the online stage with respect to four different offline ratios. For each r, there are more than one IUF u^*_r, which means that the cybersecurity dataset is appropriate to test ViewSeeker's effectiveness, because with more than one IUF in the online stage, the retrieved generic model will need the online model refinement process in order to accurately predict the interest of the online user.

Baseline Methods We implemented two baseline methods with reference to [20] and [38]. The former is one of the inspirations of our work and essentially represents the offline stage of ViewSeeker, in which the system uses the retrieved generic model for prediction without refining it in the online stage. We call the model using this method the *Offline* model. The latter is essentially

the online stage of ViewSeeker, which means that the system trains *UE* and *IE* from scratch in the online stage. We call the model using this method the *Online* model. It should be noted that the Offline model here is more general than [20] in that the generic model in the Offline model learns the most suitable combination of individual utility functions, while the generic model in [20] only learns to select one individual utility function.

Evaluation Metric Our evaluation metric is recommendation accuracy and user effort in terms of number of labels. For each result r, we define the accuracy as the size of the intersection between the top-k views recommended by *UE* and the top-k views recommended by the IUF ${u^*}_r$. For the two sets of top-k views V^p and V^* produced by *UE* and ${u^*}_r$, respectively, the accuracy is calculated as:

$$\frac{|V^p \cap V^*|}{k} \tag{26}$$

where $\cap$ is the intersection operator and $|\cdot|$ is the cardinality operator. Each result r is treated independently and the final score is the average score over all results.

Other Parameters We compared all models involved in the experiments for different top-k view recommendations with k equals to 5, 6, 7, 8, 9, and 10. For each configuration, we conducted five runs and reported the average performance in the experiment results section. In order to evaluate the size effect of the data set used to build the generic library, we have set the offline ratio to be 10%, 20%, 30%, and 40% in our experiments.

5.3.2 Experimental Results

Offline vs. ViewSeeker The comparison between the Offline model and ViewSeeker is based on recommendation accuracy. The accuracy for ViewSeeker is set to be 100% because it reached such accuracy in all configurations. The actual user labels required for it to reach a 100% accuracy are listed in Table 19.

The comparison results are shown in Figures 15-18. The Offline model achieves an average accuracy of 63%, 66%, 67% and 69% across all top-k's for offline ratio of 0.1 - 0.4 respectively, while ViewSeeker achieves 100% accuracy for all offline ratios. This observation indicates that the Offline model cannot fully capture the interest of the current user, and an online interactive model refinement stage with an average of 3 user labels could improve the recommendation accuracy

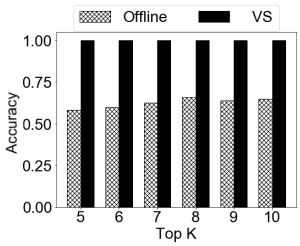


Figure 15: Accuracy of Offline and ViewSeeker (10% offline ratio).

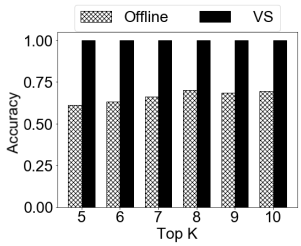


Figure 16: Accuracy of Offline and ViewSeeker (20% offline ratio).

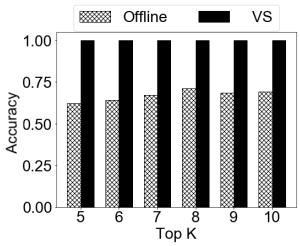


Figure 17: Accuracy of Offline and ViewSeeker (30% offline ratio).

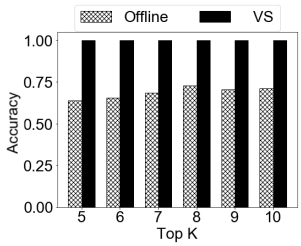


Figure 18: Accuracy of Offline and ViewSeeker (40% offline ratio).

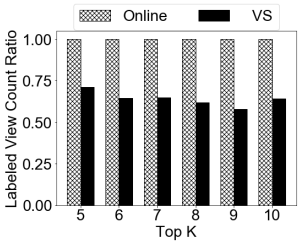


Figure 19: Accuracy of Online and ViewSeeker (10% offline ratio).

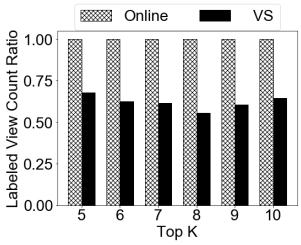


Figure 20: Accuracy of Online and ViewSeeker (20% offline ratio).

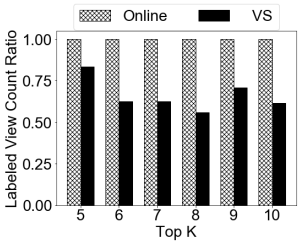


Figure 21: Accuracy of Online and ViewSeeker (30% offline ratio).

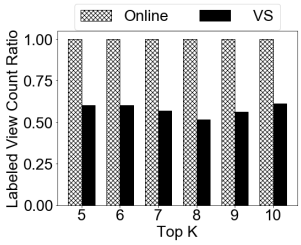


Figure 22: Accuracy of Online and ViewSeeker (40% offline ratio).

Table 19: Labeled View Count (Different offline ratios)

	OR = 0.1	OR = 0.2	OR = 0.3	OR = 0.4
Online	4.8	4.8	4.8	4.8
ViewSeeker	3.1	3.0	3.2	2.8

significantly.

Online vs. ViewSeeker Since both the Online model and ViewSeeker achieved a 100% accuracy in the online stage in the experiments, the comparison between the two is based on the ratio of label count required by the two models to reach a 100% accuracy.

Table 20: Labeled View Count (Different top-k)

	k = 5	k = 6	k = 7	k = 8
Online	4.5	4.4	5.2	4.0
ViewSeeker	3.1	2.7	3.1	2.2

The comparison results are shown in Figures 19-22. Across all top-k's, on average, ViewSeeker requires only 64%, 62%, 66%, 58% labeled views compared to the Online model to achieve a 100% accuracy for offline ratio of 0.1 - 0.4 respectively. This observation indicates that using the generic model for model initialization in the online stage can significantly reduce the user labeling effort. The actual number of labeled views for the two models are shown in Table 19 and 20.

Effect of Offline Ratio As can be seen from Figures 15-18 that with increasing offline ratios, the recommendation accuracy of the Offline model also increased slightly. This is quite understandable. With more data used to build the generic model library, the generic models will see more interest patterns from the users, thus be more likely to capture the interest of the current user.

Similarly, as can be seen from Table 19, the accuracy of ViewSeeker also increases slightly with increasing offline ratios (i.e., the model requires fewer user labels to reach a 100% accuracy). The same reason as above goes here. With increased offline ratio, the generic model will be more likely to capture the interest of the current user, so the user effort required to refine the model in the subsequent online stage is reduced.

5.4 Summary

In this chapter, we introduced ViewSeeker's external knowledge base utilization functionality. We first talked about the concept of external knowledge base and gave an example of data analysis session logs. Then, we presented an extension to ViewSeeker, which works in two stages. For the offline processing stage, we talked about the generic model library structure and training

method for the library models. For the online interaction stage, we discussed result representation, model retrieval method, and online model refinement method. We compared ViewSeeker with two baseline models which essentially represent ViewSeeker's offline and online stages respectively. The experimental results indicate that ViewSeeker could achieve an average recommendation accuracy improvement of 33.8% over the offline model with an average of 3 user labels. Besides, ViewSeeker could achieve the same accuracy as an online model with an average user effort reduction of 37.5%. Finally, we showed that with increasing offline ratios, the user effort required in the online stage could be reduced.

6.0 User Study

"Numbers tell a story but humans make it real."

– Panos K. Chrysanthis

In the previous chapters, we have shown through simulated user studies that: (1) ViewSeeker's utility preference learning and user expectation learning can improve view recommendation accuracy and (2) ViewSeeker's external knowledge base utilization can reduce user effort during the online IVR process. In this chapter, we will discuss in detail the settings and results of our real user studies to further verify the effectiveness of different functionalities of ViewSeeker. We will introduce the user study setup and protocol in Section 6.1 and Section 6.2 respectively. Then, we will discuss the data collection results in Section 6.3. Finally, we evaluate ViewSeeker's functionalities using real user data from Section 6.4 to Section 6.6.

6.1 User Study Setup

We have designed a user study that is capable of collecting all required data for the three functionalities of ViewSeeker. The user study was conducted online using the Prolific platform [2]. The recruitment process took place on Prolific and the participants need to be adult U.S. residents and have at least high school diplomas. In addition, in order to improve the data quality, they need to have at least 20 previous submissions and have an approval rate of at least 95 percent. The actual user study was implemented as a web application which was served on the cloud serving platform Heroku [1].

The dataset used in the study to generate the views is the 2019 Annual Social and Economic Supplement (ASEC) survey [8]. ASEC is a supplementary survey to the Current Population Survey (CPS) and contains data about respondents' social status and economic situation. We have selected 5 variables as dimension attributes in our study: *sex, race, marital status, highest education, work*

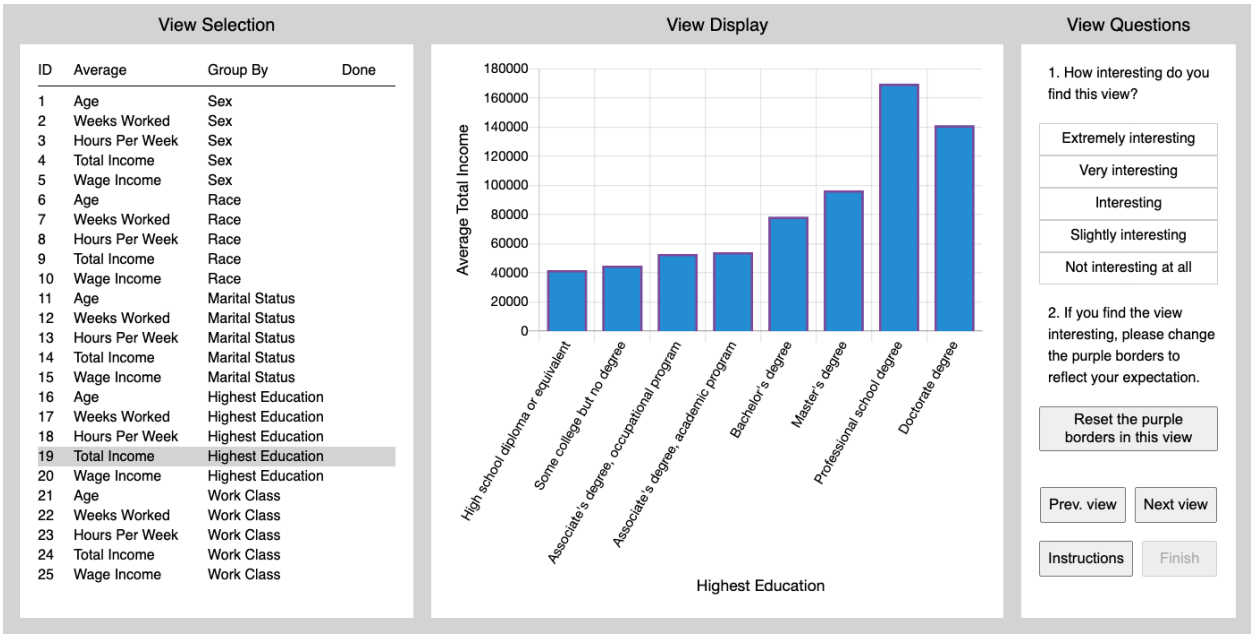


Figure 23: Initial study question page

class. We have selected 5 variables as measure attributes in our study: *age, weeks worked, average hours per week, total income, wage income*. We have used the aggregate function `AVERAGE` to generate the views. In total, each participant has 25 views to label.

6.2 Study Protocol

We discuss step by step the study protocol of our user study in this section. The first page of the study app is an introduction page. It briefly introduces the study setup and provides an example case of how a user could label a view.

After hitting the "Continue" button, the user will come to the second page of the app, which contains the actual view labeling questions. As can be seen from Figure 23, the page has three panels. The left panel is for view selection, in which the user can select a view to label. Each entry in this panel displays the dimension and measure attributes used to generate the corresponding view. The middle panel displays the selected view. For example, the view in Figure 23 illustrates

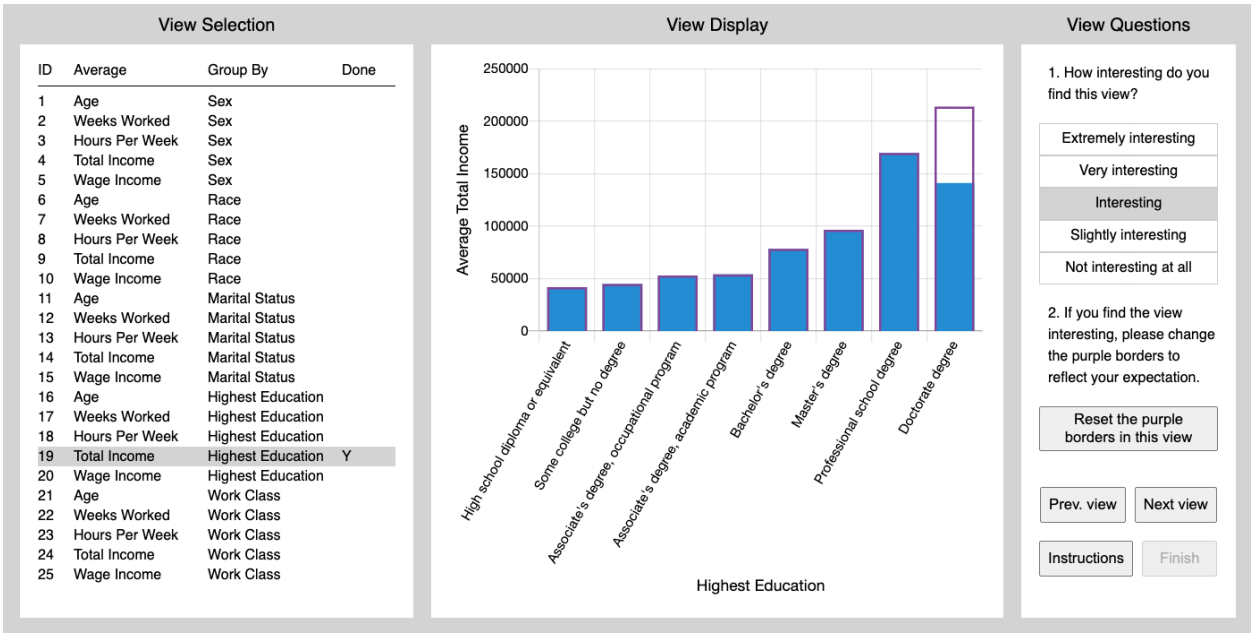


Figure 24: Study question page with a labeled view

the average total income based on highest education levels. As can be seen that each bar in the view has a purple border, whose function will be introduced shortly.

The right panel contains the questions for the selected view. Each view has two questions. First, the user needs to indicate the interestingness of the view on a five-level Likert-scale. The users have been instructed during introduction that in the context of our study, a view is interesting if *its values differ from their expectation.* In other words, the interestingness of the view should be based on the deviation between the values in the displayed view and their expectation. The second question asks the user to drag the purple borders in the displayed view up or down to reflect their expected values if they found the view interesting. In other words, the purple borders after the user change should capture the user's expectation of the values in the view. Note that, the user does not need to make changes to the purple borders if they found the view uninteresting (i.e., selected the option "not interesting at all") in the first question.

Figure 24 shows an example labeling of the aforementioned view. It can be seen that the user finds the view interesting and have increased the average total income of people with doctor

degrees to reflect their expectation. After the user finishes labeling all 25 views, the "Finish" button at the bottom right of the page will be enabled to allow the user to go to the final page of the app, which contains an optional text field for the user to leave comments and an exit button to complete the study.

Each user study survey was estimated to take around 15 minutes. The payment for completing each survey was 8 dollars and was handled by the Prolific platform. Each Prolific user can only participate in the survey once.

6.3 Data Collection Results

We have collected labeling data from 120 users, among which 107 responses are with good quality and are used in the evaluation. Low quality responses involve one or more of the following issues.

- The user has bypassed the integrity constraints. For example, the user managed to skip the value changing step even if they found the view interesting. This could be due to their JavaScript version being too old thus causing malfunction in some JavaScript components.
- The user has provided unrealistic responses, for instance, working 100 weeks a year.
- The user has made very tiny changes in the view. For example, they may have dragged the average income up or down by less than one cent. This is actually an impossible input from the GUI, so this also could be caused by some JavaScript version issues, which caused incorrect rendering and reading of bar values.

Furthermore, we also excluded responses with fewer than 5 labeled-interesting views. The reason is that during the evaluation of utility function preference learning, we need to calculate the top-k accuracy of our prediction, and a ground truth ranking with 5 or more ranked views will make the evaluation more meaningful.

Figure 25 shows participant distribution in some basic demographic attributes. As the figure shows, the majority of the participant is under 40 years old. Female participant count is about twice that of male participant. Around one-third of the participants are students and more than half

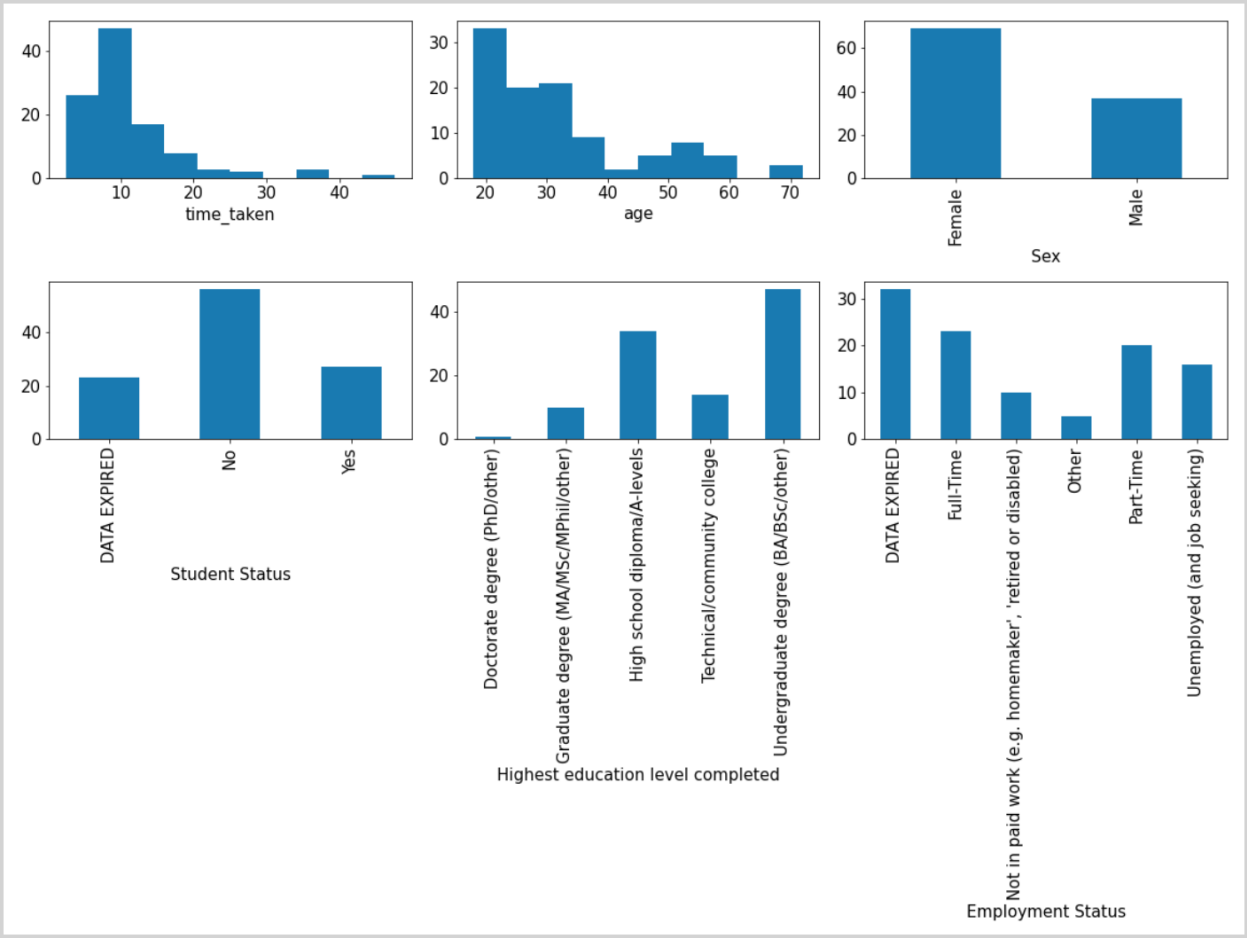


Figure 25: Study participant demographic distribution

of the participants have undergraduate or higher degrees. As for completion time, a majority of the participants finishes the study within 20 minutes.

6.4 Study Evaluation - User Expectation Learning

As mentioned in the study protocol, the changed purple borders in the views reflect user expectation of values in the views. Since the user needs to label all views, so we have essentially collected their expectation on all views. Therefore, we can conduct the EAP algorithm (Chap-

ter 4) based on some selected example views, and evaluate the estimated user expectation on the unlabeled views using the corresponding ground truth labels.

In the user study, we select one example view and estimate user expectation on the remaining 24 unlabeled views. We enhanced the example selection strategy mentioned in Chapter 4 to take into account both the dimension and measure attributes of the views. Recall that the original strategy would select a view whose dimension has the largest group count among all dimensions as shown in Equation 27:

$$a^* = \underset{a}{\operatorname{argmax}}(|a|) \tag{27}$$

where a^* is the selected dimension and $|\cdot|$ is the group count operator. In the user study, we added another criterion for measure selection as shown in Equation 28:

$$m^* = \underset{m}{\operatorname{argmax}}\, Var(P(v_{a^*,m,AVG})) \tag{28}$$

where $P(\cdot)$ is the view normalization operator as introduced in Chapter 4 and $Var(\cdot)$ is the variance operator. In other words, among all views with dimension a^*, the strategy will select the view whose normalized values have the highest variance as the example. In our user study, the example view selected by the above strategy is the view shown in Figure 23.

We used the same evaluation metric from the simulated user study (Section 4.4), namely the average Expectation Estimation Error (EEE) over the unlabeled views, with EEE for each view being calculated according to Equation 29:

$$EEE_i = Dist(P(v_i^R), P(v_i^{IR})) \tag{29}$$

where v_i^R is the estimated user expectation and v_i^{IR} is the ground truth user label. The distance function $Dist(\cdot)$ is the L1 distance. The state-of-the-art model for comparison is the QuickInsights model [6]. Recall from Section 2.5 that QuickInsights uses a power-law line fitted on the sorted values in the view as the estimated user expectation for the view.

The evaluation results are as follow. The average EEE among all unlabeled views for QuickInsights and EAP are **0.097** and **0.045** respectively, which means that EAP can reduce the expectation estimation error of QuickInsights by more than 50%, and thus can estimate user expectation more accurate than QuickInsights with user label on one example view.

6.5 Study Evaluation - Utility Function Preference Learning

Recall from the study setup that we have asked the users to evaluate the overall view interestingness based on the deviation between the view values and their expectation, so we could assume that the users' evaluation could be captured by deviation-based utility functions that are calculated based on the the difference between the view values and the corresponding user expectation labels. Therefore, in our evaluation, we have used deviation-based utility function scores as view features to predict the overall view interestingness labels. The calculation equation for each utility function is shown below:

$$u(P(v^T), P(v^{IR})) \tag{30}$$

where $u(\cdot)$ represents the utility function, v^T is the original view and v^{IR} contains the ground truth expectation labels.

In our evaluation, we used five different deviation-based utility functions to generate view features. Some of the utility functions were proposed by previous related works, others are originally designed based on our observation of the relations between user expectation labels and the corresponding view interestingness labels:

- The first deviation-based utility function is the *earth mover's distance (EMD)* [32], which is same as the L1 distance if there is no ordinal relations between the dimension groups in the view.
- The second utility function is called *bar percentage (Bar%)*, and it is calculated as the count ratio of bars with changed values (i.e., purple borders). For example, if four bars were changed in a view with eight bars, then the Bar% score for this view will be 0.5.
- The third and fourth utility functions are calculated based on the visual offset between the original bar (i.e., the blue bar) and the purple border for each bar, and we call this offset *absolute change*. The third utility function is the average absolute change across all bars in a view (*Avg-Abs*). The fourth utility function is the maximum absolute change across all bars in a view (*Lead-Abs*).
- The fifth utility function is calculated based on the ratio between the change and the original value in a bar, and we cal this ratio *relative change*. The fifth utility function is the maximum

relative change across all bars in a view (*Lead-Rel*).

As mentioned in Chapter 3, for Likert-scale feedback, we would formalize the learning problem as a regression problem and convert the Likert-scale labels into real number labels. We used an equal-spacing conversion method in our evaluation and mapped the five levels 1-5 into real number labels of values 0, 0.25, 0.5, 0.75 and 1.0 respectively. The learner used in the evaluation is the linear regression model.

The accuracy evaluation metric is the *utility distance* [32]. For the ground truth top-k views $V^* = \{v_1^*, v_2^*, ..., v_k^*\}$ and the top-k prediction $V^p = \{v_1^p, v_2^p, ..., v_k^p\}$, the utility distance is calculated as:

$$\frac{1}{k}(\sum_{i=1}^{k} U(v_i^*) - \sum_{i=1}^{k} U(v_i^p)) \tag{31}$$

where $U(\cdot)$ represents the real number label for the view.

The baselines used in the evaluation are the individual deviation-based utility functions (i.e., individual view features), whose accuracy results are shown in Figure 26.

The x-axis in the figure represents different top-k's and the y-axis represents the utility distance. It can be seen that the baseline EMD and Avg-Abs have the best accuracy among the baselines and will be used in the comparison with ViewSeeker. The corresponding comparison result is shown in Figure 27.

The labels starting with the letters "VS" in the legend represent ViewSeeker's accuracy, and the numbers after "VS" indicate the number of labels used to train the model. It can be seen that the accuracy of ViewSeeker improves with increasing training labels. When trained with 3 labels, the model performs worse than the baselines. When trained with 6 labels, the model performs slightly better than the baselines when top-k is smaller than 7. When trained with 9 labels, the model performs overall better than the baselines.

The above results indicate that ViewSeeker can eventually outperform the baselines by a large margin but has a cold start issue, namely its accuracy is not satisfactory with few training labels. This cold start issue could be mitigated by the utilization of external knowledge bases (Chapter 5), which will be evaluated in the next section.

In addition, we have also conducted evaluations for the combined process of user expectation learning and utility function preference learning. In order to do so, we changed the reference

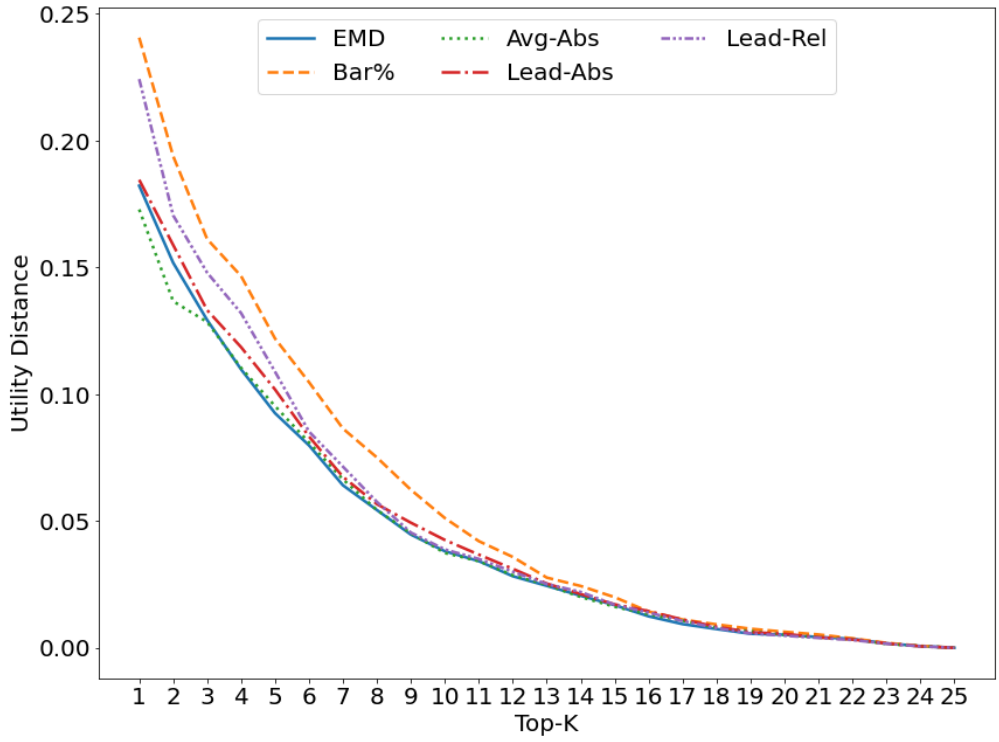


Figure 26: Utility distances for baseline models

view in the utility function calculation to be the estimated user expectation instead of ground truth expectation labels as shown in Equation 32:

$$u(P(v^T), P(v^R)) \tag{32}$$

where v^R represents the estimated user expectation.

We used two user expectation estimation models as those in Section 6.4, namely QuickInsights and our EAP algorithm. The comparison results are shown in Figure 28.

It can be seen that when trained with 3 labels, both models perform similarly. However, with increasing training labels, the combined process with the EAP algorithm gradually outperforms the QuickInsights counterpart. We can also see that at 9 training labels, the "EAP-VS-9" model performs almost as well as the "VS-9" model in Figure 27. This indicates that the EAP algorithm has estimated user expectation quite accurately, thus generating quite accurate view features, enable a recommendation accuracy almost as good as models using ground truth view features.

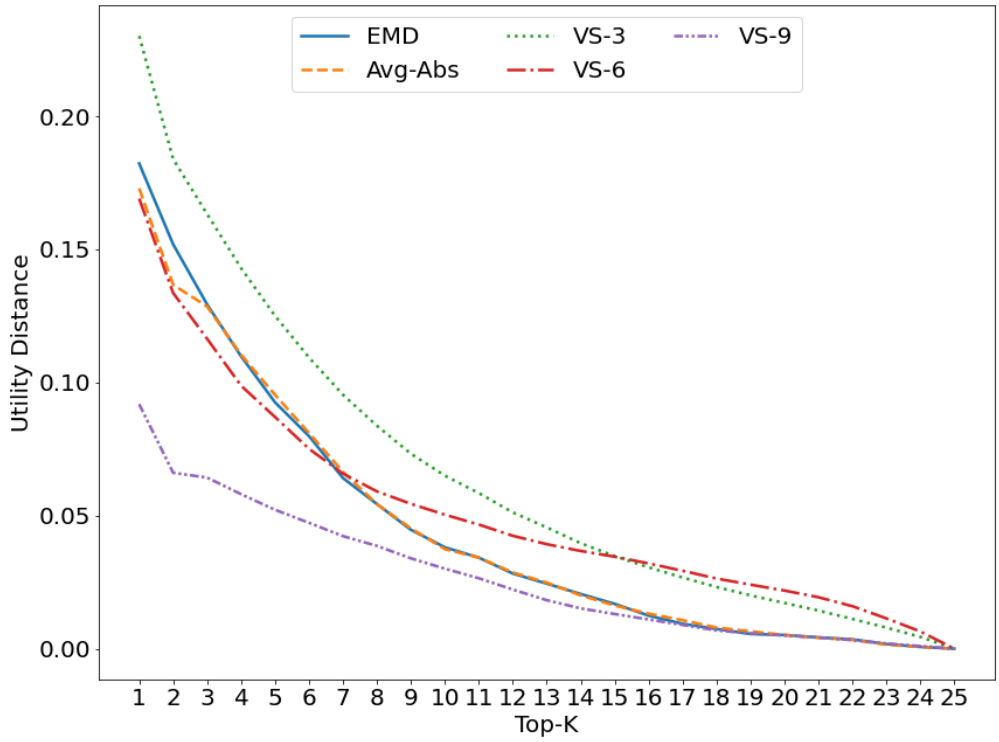


Figure 27: Utility distances for baselines and ViewSeeker

6.6 Study Evaluation - External Knowledge Base Utilization

In our study evaluation, we treat the labels from the other users as the external knowledge base and build a generic model library thereon, then we retrieve a generic model from the library to initialize the learners and continue the IVR process as in the previous section.

Recall from Chapter 5 that the generic model library has a cell for each unique combination of *data result* and *feedback type*. However, since in the user study all views are generated from the initial result (i.e., whole dataset) and there is only one feedback type (i.e., Likert-scale), there is only one library cell. This means that for any session in the user study, the generic model to be retrieved will be the same, which might not be very helpful in providing personalized view recommendation. Therefore, in our study evaluation, we have enhanced our generic model building and retrieval methods to be more fine-grained within one library cell. We introduce our new methods

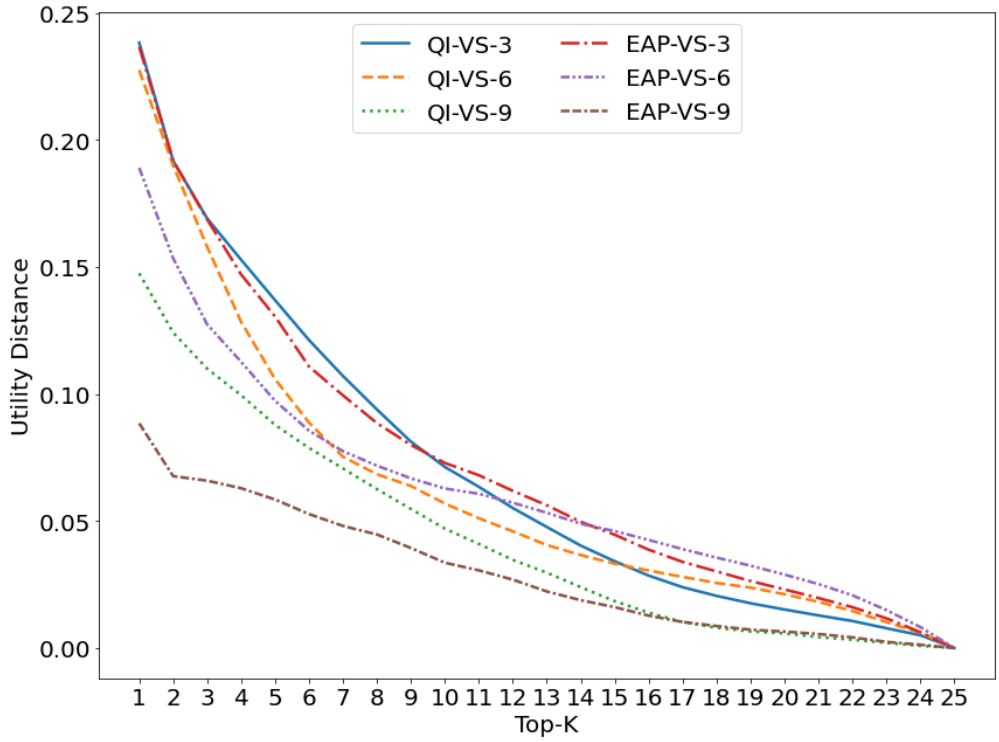


Figure 28: Accuracy results for QuickInsights and EAP in the combined process

below.

6.6.1 Generic Model Building & Retrieval

First of all, we train a model for each user. As with the previous section, the learning features are the utility function scores based on the ground truth expectation labels and the learning targets are the overall view interestingness scores.

Then, based on the current user's view interestingness labels, we search the model pool to find user models whose predictions for the labeled views are most similar to the current user's labels. In practice, we used the $L2$ distance function as shown in Equation 33 for similarity calculation between the target user a and a candidate model b:

$$sim(a,b) = -L2(\langle s_a(v_1), s_a(v_2), ..., s_a(v_n)\rangle, \langle s_b(v_1), s_b(v_2), ..., s_b(v_n)\rangle) \quad (33)$$

where $s_a(v)$ represents the user's label for view v and $s_b(v)$ represents the model's prediction for v.

Finally, we retrieve the top-3 neighboring models that will be used as the example informativeness estimator *IE* and the view utility estimator *UE* in the online IVR process.

6.6.2 Online Model Refinement

Since we have 3 neighboring models retrieved from the previous step, which are similar but not necessarily identical, they would serve very well the role of a learner committee to conduct the query-by-committee query strategy for example selection. Therefore, we use the neighboring model committee as the example informativeness estimator.

Besides, we build a bagging ensemble from the 3 neighboring models as the view utility estimator. To put it in a simpler way, we use the average prediction from the neighboring models as the view utility prediction.

We also enhanced the model refinement method. Recall that in Chapter 5, we used the current user's labels on the generic model to refine it. We chose this approach because we had only one model in each library cell. However, in our user study, we have multiple user models in each library cell. Therefore, we could refine the model through refining the neighbor retrieval process, in order to acquire as much useful external knowledge (i.e., similar users' interest patterns) as possible. Specifically, after each new user label, we would use the latest user label set to retrieve a new set of neighbor models, that would serve the roles of a refined *IE* and *UE*.

The accuracy result from the above refinement method is shown in Figure 29. The "GT" in the legend means that we used ground truth expectation labels to generate the view features. The number indicates the number of online labels.

It can be seen that the model's accuracy improves in the early stage of the IVR process, but stops improving in the late stage. This is understandable, because our view interestingness prediction is only based on the neighbor models, which are not supposed to fully capture the interest of the current user.

On the contrary, if we do not use the neighbor models and train a model from scratch, and predict view interestingness based on that model, which we call the *current* model, then the ac-

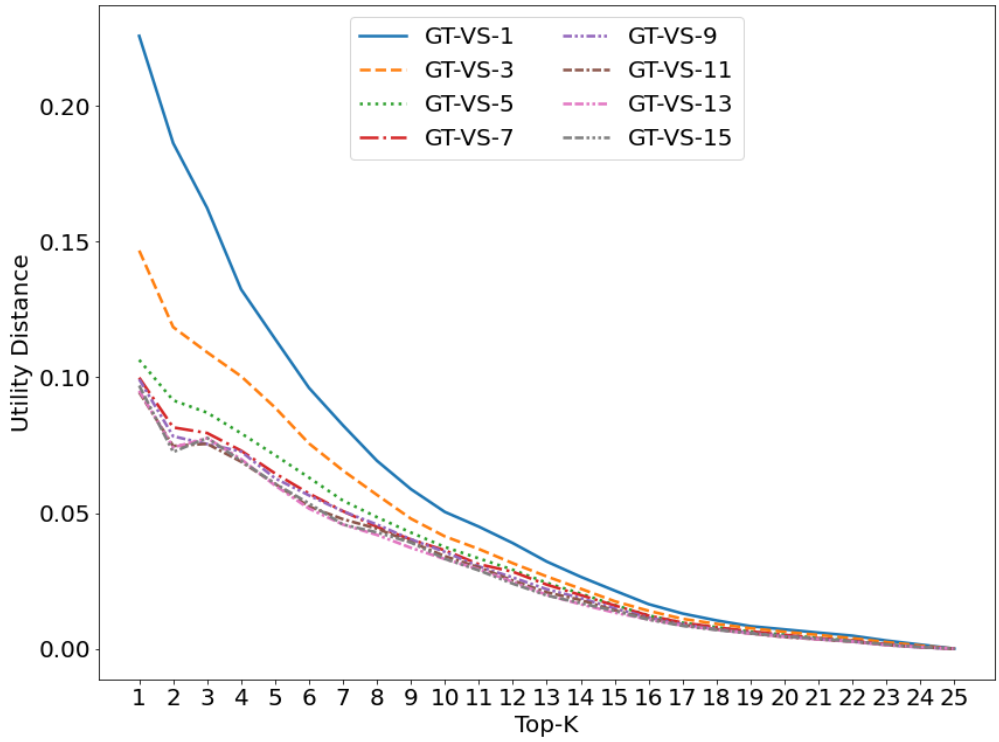


Figure 29: Accuracy result using only the neighbor models

curacy result would look like Figure 30. Since this configuration is essentially the same as the ViewSeeker model in Figure 27, it suffers the same cold start issue, such that the model's accuracy is not satisfactory in the early stage of the IVR process, but gradually improves in the late stage.

One natural question to ask after reviewing the above results is that if there is a way to combine the strengths of the two models above such that the combined model would have a good accuracy in the early stage of the IVR process, while also being able to gradually improve its accuracy in the late stage. In the following part of the study evaluation, we take a first step towards the above objective.

6.6.3 Model Convergence Detection & Swapping

The approach we attempted is to add a model swapping mechanism to the model refinement process, such that we use the neighbor models in the early stage of the IVR process, and swap

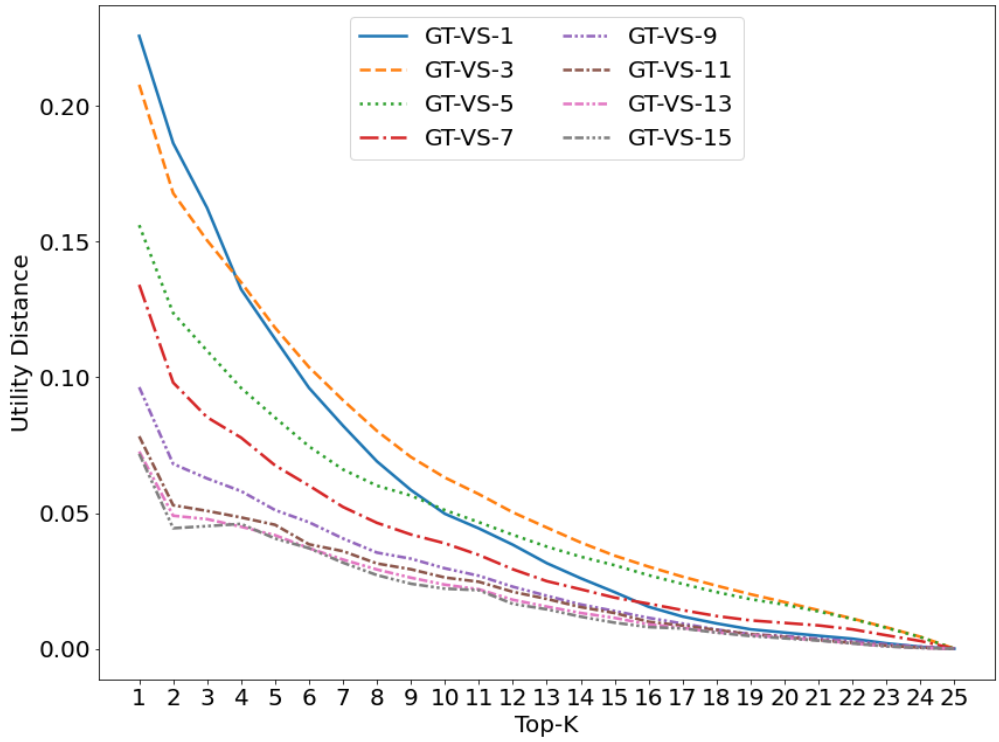


Figure 30: Accuracy result using only the current model

to the current model when its accuracy is satisfactory enough. However, since model accuracy is unknown in real use cases, we have introduced a *model convergence detection* algorithm for the current model, such that after the algorithm decides that the current model has converged, it will swap it in the place of the neighbor models.

The convergence detection algorithm is based on a new metric, called *Inter-iteration Prediction Error (IPE)*, which measures the mean squared error between two versions of model prediction on all views, one from the previous iteration and the other from the current iteration. The calculation formula for IPE is shown in Equation 34:

$$IPE = MSE(\langle s_{k-1}(v_1), s_{k-1}(v_2), ..., s_{k-1}(v_n)\rangle, \langle s_k(v_1), s_k(v_2), ..., s_k(v_n)\rangle) \tag{34}$$

where $s(v)$ represents the model's prediction on view v and the subscript k represents the iteration number.

Algorithm 4 ViewSeeker Model Convergence Detection & Swapping

Require: Generic model library G, current result r
Ensure: The view utility estimator *UE*

Unlabeled example set $U \leftarrow GenerateExamples(r)$
Labeled example set $L \leftarrow AcquireInitLabels()$
$M \leftarrow$ retrieve neighbor models from G based on L
$N \leftarrow$ initialize current model using L
$P_{new} \leftarrow$ generate predictions on all views using N
$UE \leftarrow$ initialize view utility estimator *UE* using M
$IE \leftarrow$ initialize example informativeness estimator *IE* using M
loop
  Choose an example x from U using *IE*
  Solicit user label on x
  $L \leftarrow L \cup \{x\}$
  $U \leftarrow U - \{x\}$
  $M \leftarrow$ update neighbor models from G based on L
  $N \leftarrow$ update current model using L
  if M is used for *UE* **then**
    $P_{old} \leftarrow P_{new}$
    $P_{new} \leftarrow$ generate predictions on all views using N
    $IPE \leftarrow$ calculate IPE from P_{old} and P_{new}
    if IPE drops for 3 consecutive iterations **then**
      $UE, IE \leftarrow$ refine *UE* and *IE* using N
    else
      $UE, IE \leftarrow$ refine *UE* and *IE* using M
    end if
  else
    $UE, IE \leftarrow$ refine *UE* and *IE* using N
  end if
  $T \leftarrow$ recommend top views using *UE*
  if the user is satisfied with T or the user wants to stop **then**
    Break
  end if
end loop
Return the most recent *UE*

The criterion we used to detect convergence is that if the IPE keeps dropping for three consecutive iterations, the algorithm would initialize the swapping operation. The intuition beneath the algorithm is that the change of model prediction on the views reflects the change in the model parameters. So if the change gets smaller, it means that the model parameters becomes more and more stable, which could indicate model convergence. The overall workflow of ViewSeeker based on the new algorithm is shown in Algorithm 4.

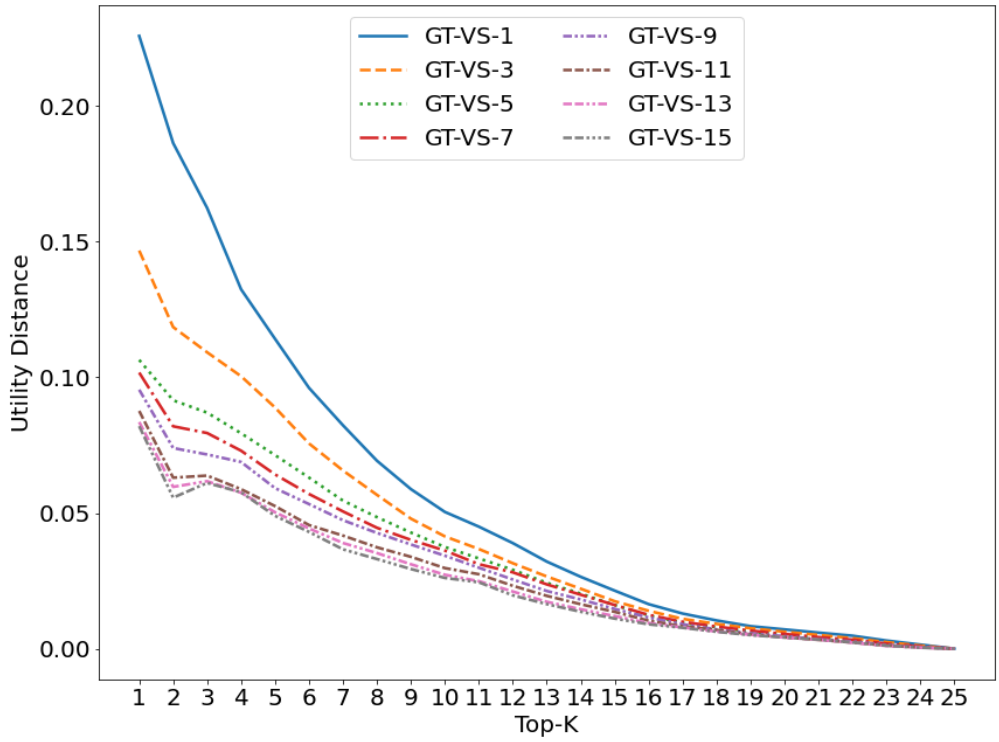


Figure 31: Accuracy result swapping model at IPE = 3

The accuracy result for the above algorithm is shown in Figure 31. It can be seen that the model's accuracy is as good as the neighbor models in the early stage of the IVR process, but is not as good as the current model in the late stage. Since we suspect that the convergence detection algorithm failed to detect some convergence cases, such that it did not trigger the swapping operations for some cases, leading to worse accuracy in the late stage, we added a force swapping operation at 10 examples. The corresponding result is shown in Figure 32.

Now we can see that the model's accuracy is as good as the neighbor models in the early stage of the IVR process, and as good as the current model in the late stage, which means that we have got the best of both worlds.

Apparently the convergence detection algorithm introduced above is still not sophisticated enough to handle all use cases, but from a qualitative analysis point of view, we have verified our claim in the previous section that the use of the external knowledge bases could improve the model

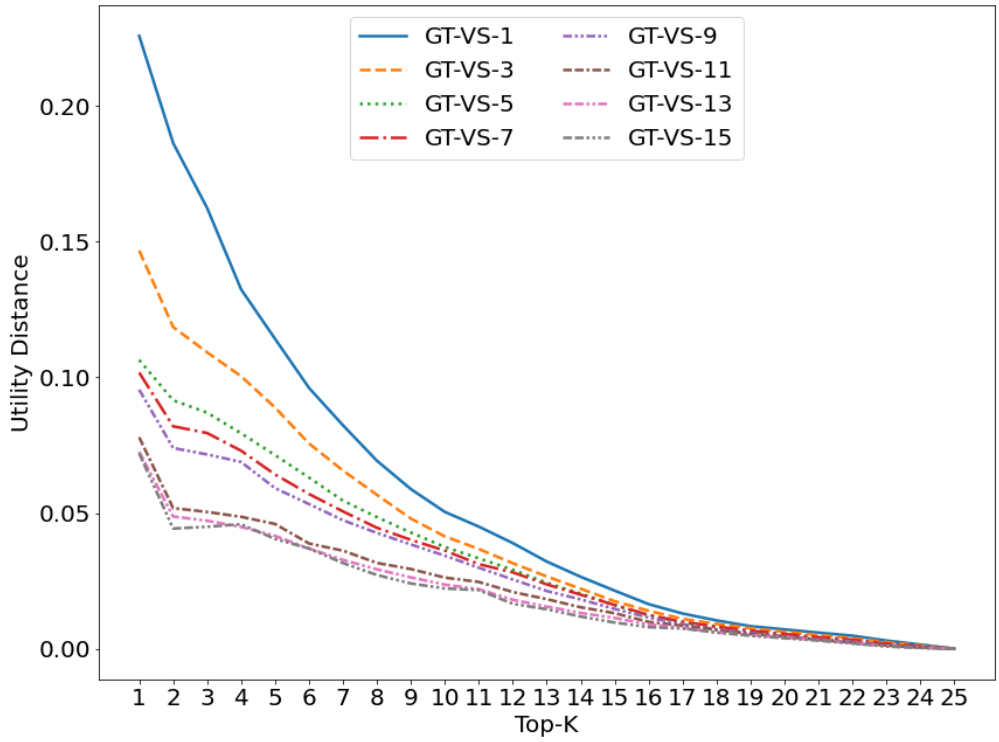


Figure 32: Accuracy result swapping model at IPE = 3, force swapping at 10 examples

accuracy, especially in the early stage of the IVR process (i.e., mitigate the cold start issue). As discussed in Section 7.2, we are planning to further develop the convergence detection algorithm.

6.7 Summary

In this chapter, we discussed in detail our real user studies for ViewSeeker. We first introduced the study setup and protocol. Then, we talked about the data collection results including study statistics and demographic distributions. After that, we conducted study evaluation for each of ViewSeeker's functionalities. For user expectation learning, the study result showed that ViewSeeker has an estimation error reduction of more than 50% against the state-of-the-art non-IVR model QuickInsights. For utility function preference learning, ViewSeeker outperforms

the baselines (i.e., individual utility functions) by a large margin, especially with a reasonable number of labels. The accurate user expectation estimates from the EAP algorithm also enables ViewSeeker to outperform QuickInsights significantly in recommendation accuracy. For external knowledge base utilization, we refined the library model retrieval and online model refinement methods in order to extract more information from available user labels. We also developed a convergence detection method to improve model refinement. The final ViewSeeker could achieve neighboring models' accuracy in the early stage and current model's accuracy in the late stage of the IVR process. Overall, the real user study results are compatible with the corresponding results in the simulated user studies.

7.0 Conclusion

7.1 Contributions

Motivated by the need to provide personalized view recommendations in this thesis we brought the user-in-the-loop paradigm into view recommenders. Our hypothesis is that *a user-in-the-loop approach for view recommendations can improve the view recommendation quality by tailoring them to the user's interest.*

To test our hypothesis, we proposed a new paradigm, called *Interactive View Recommendation (IVR)*, in which the system interacts with the user to elicit user interests and preferences so as to formulate a utility function that is most suitable for the current user. Our contributions are summarized below:

As a proof-of-concept, we developed *ViewSeeker*, an IVR framework and prototype system. We further designed an overall workflow of ViewSeeker that contains three main functionalities (Figure 1). The first functionality is *utility function preference learning*, in which the system discovers the user's interests across different utility functions. The second functionality is *user expectation learning*, in which the system estimates user expectation of the values in the views. The third functionality is *external knowledge base utilization*, in which the system utilizes user interest information from external knowledge base to further reduce user effort in the IVR process. The functionalities are considered effective if the recommendation quality could be improved by the IVR process and the user effort could be reduced with the help of external knowledge bases.

To validate our hypothesis and claims, we have evaluated the functionalities using both experimental simulations and real user studies in terms of view recommendation quality and user effort (i.e., number of feedback/labels). The summarized comparison between the simulated and the real user studies is shown in Table 21. All measures are relative measures against corresponding baselines. The results for real user study are based on Top-k with $k = 5$, which represents a recommendation ratio of 20% of all views and is larger than most of the counterparts in the simulated studies. For table space reasons, we have used "Acc $\uparrow$" for accuracy improvement, "UD $\downarrow$" for utility distance reduction, $|L|$ for number of labels, and Func 1 to Func 3 for the three functionalities

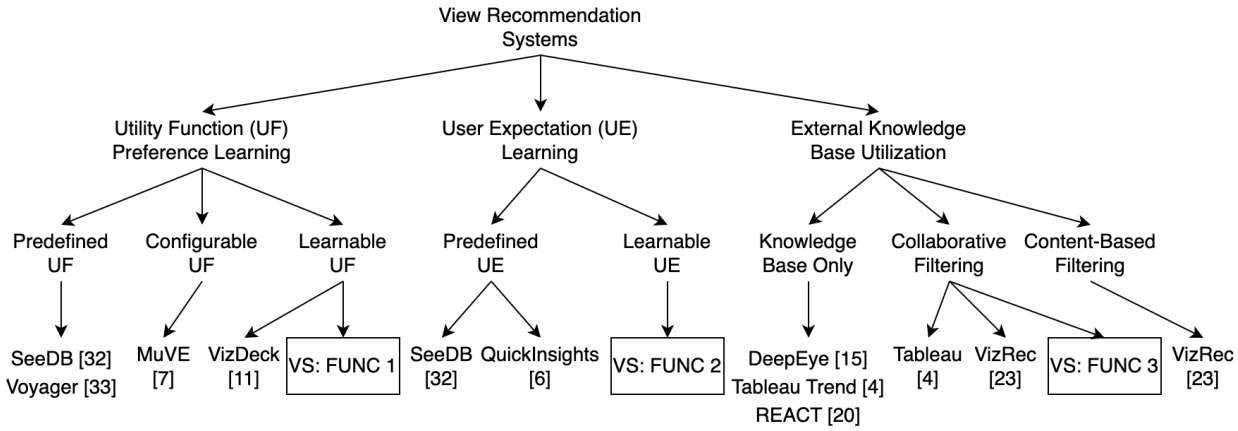


Figure 33: Related work taxonomy for ViewSeeker's functionalities. Our contributions are marked with squares.

of ViewSeeker respectively.

The contributions for the three functionalities are marked with squares in Figure 33 and detailed below:

- For utility function preference learning (VS: Func 1), we developed a variety of feedback types and their corresponding example forms, active learning-based query strategies and ranking algorithms under the IVR workflow. The simulated experimental results show that ViewSeeker can learn the IUFs well with fewer than 10 labels on average. We also showed that ViewSeeker outperforms non-IVR baselines by at least 32.6% relatively in recommendation accuracy. In real user study, ViewSeeker outperforms the baselines (i.e., individual utility functions) by a large margin, especially with a reasonable number of labels (i.e., 9 on average) as shown in Table 21, row Func 1.
- For user expectation learning (VS: Func 2), we devised a novel algorithm, called *Expectation Acquisition & Propagation (EAP)*, which elicits user expectation on example views and based on them estimates user expectation on unlabeled views. Within EAP, we designed expectation propagation algorithms across both dimensions and measures with corresponding runtime complexity analysis. The simulated experimental results indicate that ViewSeeker

Table 21: Result comparison between simulated and real user studies
(Acc ↑: accuracy improvement; UD ↓: utility distance reduction; $|L|$: number of labels)

Functionality	Simulated user study	Real user study
Func 1	32.6% Acc ↑ ($|L|$=6-11)	42.9% UD ↓ ($|L|$=9)
Func 2	94.8% EEE ↓	53.6% EEE ↓
Func 2 + 1	83.1% Acc ↑ ($|L|$=10)	34.1% UD ↓ ($|L|$=9)
Func 3(a)	vs Offline: 51.1% Acc ↑ ($|L|$=3)	vs Neighbor: 24.4% UD ↓ ($|L|$=15)
Func 3(b)	vs Online: 37.5% Effort ↓ ($|L|$=3 vs 5)	vs Current: 26.6% UD ↓ ($|L|$=3)

can reduce user expectation estimation error from state-of-the-art non-IVR recommenders by more than 90% with user labeling effort on only three example views. When combined with a downstream utility function preference learning process, ViewSeeker also outperforms the comparison models by 83.1% relatively in average recommendation accuracy. In real user study, ViewSeeker has an estimation error reduction of more than 50% against the state-of-the-art non-IVR model QuickInsights (see Table 21, row Func 2). The accurate user expectation estimates from the EAP algorithm also enables ViewSeeker to outperform QuickInsights significantly in recommendation accuracy (see Table 21, row Func 2 + 1).

- For external knowledge base utilization (VS: Func 3), we extended ViewSeeker to incorporate two stages. In the offline stage, it builds a generic model library using data analysis session logs, and then in the online stage, it retrieves a suitable model from the library and refines it based on the current user's feedback. The simulated experimental results indicate that ViewSeeker could achieve an average recommendation accuracy improvement of 51.1% relatively over the offline model with an average of 3 user labels (see Table 21, row Func 3(a)). Besides, ViewSeeker could achieve the same accuracy as an online model with an average user effort reduction of 37.5% (see Table 21, row Func 3(b)). In real user study, we refined the library model retrieval and online model refinement methods in order to extract more information from available user labels. We also developed a novel model convergence detection and

swapping algorithm to further improve the model refinement process. The final ViewSeeker significantly outperforms the current model in the early stage and the neighbor model in the late stage of the IVR process (see Table 21, row Func 3(b) and Func 3(a)).

It can be seen that, the improvements of ViewSeeker against the baselines in real user study are compatible with their simulated counterparts, although lower in general, yet still significant for all functionalities.

7.2 Future Work

Clearly, there are many potential directions for future work derived from this thesis.

- An obvious extension to the ViewSeeker framework and its functionalities is to add support for visualization types other than the bar chart. Take line chart for example, user utility function preference learning could be naturally applied to it because there are also a variety of utility functions proposed by previous works such as [6] and [15] for it. For user expectation learning, since the dimension attribute in all line charts is time, we could apply the *Expectation Propagation across Measures* algorithm to estimate user expectation on unlabeled views.
- Besides, currently ViewSeeker assumes that the user's utility function preference is in the form of a linear model. We could extend this to non-linear forms and experiment with different non-linear learners. Some preliminary work in this direction could be found in [37].
- In user expectation learning, the generation of the helper views and helper matrices has a high time complexity. We could adopt uniform or stratified sampling to generate sketches for them and evaluate the corresponding recommendation accuracy loss.
- User expectation labeling effort for an example view might not be small because a view could have multiple dozens of bars. We could develop selection strategies for bars in a view, so that the user only needs to label several bars in an example view.
- For external knowledge base utilization, we could incorporate previous results into the result representation and correspondingly improve the search method to be session-history-aware.

- Another direction is to improve the convergence detection algorithm. For example, we could use time series models on different indicators such as the inter-iteration prediction error to predict convergence during an IVR session.
- Finally, more real user studies could be conducted on different datasets as well as with different user groups (e.g., general public, domain experts, etc.) to further validate the effectiveness of ViewSeeker's functionalities.

7.3 Broad Impact

Data visualizations (i.e., views) are very helpful in initial data exploration and analysis to help the users quickly understand large volumes of data, which is crucial in sense-making data-intensive domains such as finance, science, healthcare, etc. View recommendation, as an enabling technique, can greatly speed up the exploration process for discovering interesting and insightful views. The output of this thesis, namely the ViewSeeker framework, aims to improve the view recommendation quality through light user interaction.

Naturally, ViewSeeker could be adopted by domain experts to conduct visual data analyses that are tailored to their interests and goals. For example, they could use the user expectation learning functionality to quickly discover interesting views and filter out uninteresting ones. More broadly, ViewSeeker could also be used by general public due to its intuitive interface and easy-to-use characteristics. This is especially the case during this big data era in which data visualizations become a trend in data analysis reports in public media like newspapers and news feeds. The utility function preference learning functionality could serve the general public well in this case to enable interactive data visualizations and even personalized data visualizations for registered users. Certainly, the external knowledge base utilization functionality could also help in the above process.

Our prototype and user study could serve the first step in this effort to bring more interactive and personalized data visualizations to the domain experts and the general public. Since the study results are good, we are quite confident that ViewSeeker, as a novel interactive tool in the view recommendation arsenal, would bring values to the visual data analysis community.

Finally, our experience developing ViewSeeker have also contributed to the development of other visual data analysis tools, specifically CovidLens [27] and HR-VEAW [34]. The former supports visual analysis of the COVID-19 pandemic's effect on people's mobility patterns whereas the latter helps to understand the relation between human rights violations and large-scale societal instability events such as civil wars.

Bibliography

[1] Heroku. https://heroku.com. Accessed: 2021-11-02.

[2] Prolific. https://prolific.co. Accessed: 2021-11-02.

[3] UCI ML diabetes data set. https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008/, 2019.

[4] Tableau public. http://public.tableau.com, 2020. Accessed: 2020-07-14.

[5] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[6] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *ACM SIGMOD*, pages 317–332, 2019.

[7] Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis. Efficient recommendation of aggregate data visualizations. *IEEE Trans. Knowl. Data Eng.*, 30(2):263–277, 2018.

[8] Sarah Flood, Miriam King, Renae Rodgers, Steven Ruggles, and Robert Warren. Integrated public use microdata series, current population survey: Version 7.0 [dataset]. In *Minneapolis, MN: IPUMS*, 2020.

[9] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff G. Schneider, and Richard P. Mann. Bayesian optimal active search and surveying. In *ICML*. icml.cc / Omnipress, 2012.

[10] Robert J. Hilderman and Howard J. Hamilton. Knowledge discovery and measures of interest. 638, 2013.

[11] Alicia Key, Bill Howe, Daniel Perry, and Cecilia R. Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *ACM SIGMOD*, pages 681–684, 2012.

[12] Martin Krzywinski and Naomi Altman. Significance, p values and t-tests. In *Nature methods*, 2013.

[13] Song Lin and Donald E Brown. An outlier-based data association method for linking criminal incidents. *Decision Support Systems*, 41(3):604–615, 2006.

[14] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[15] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. Deepeye: Towards automatic data visualization. In *IEEE ICDE*, 2018.

[16] Jock D. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[17] Jock D. Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1137–1144, 2007.

[18] Rischan Mafrur, Mohamed A. Sharaf, and Hina A. Khan. Dive: Diversifying view recommendation for visual data exploration. In *ACM CIKM*, pages 1123–1132, 2018.

[19] Christopher D Manning. *Introduction to information retrieval*. Syngress Publishing,, 2008.

[20] Tova Milo, Chai Ozeri, and Amit Somech. Predicting "what is interesting" by mining interactive-data-analysis session logs. In *EDBT*, 2019.

[21] Tova Milo and Amit Somech. Next-step suggestions for modern interactive data analysis platforms. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 576–585. ACM, 2018.

[22] Kristi Morton, Magdalena Balazinska, Dan Grossman, and Jock D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *Proc. VLDB Endow.*, 7(6):453–456, 2014.

[23] Belgin Mutlu, Eduardo E. Veas, and Christoph Trattner. Vizrec: Recommending personalized visualizations. *ACM Trans. Interact. Intell. Syst.*, 6(4):31:1–31:39, 2016.

[24] Melanie A Revilla, Willem E Saris, and Jon A Krosnick. Choosing the number of categories in agree–disagree scales. *Sociological Methods & Research*, 43(1):73–97, 2014.

[25] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[26] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[27] M. A. Sharaf, X. Zhang, P. K. Chrysanthis, W. Alsaedi, M. Alkalbani, H. Helal, and A. Aldhaheri. Covidlens: Visually understanding the covid-19 indicators through the lens of mobility data. In *23rd IEEE International Conference on Mobile Data Management*, pages 302–305. IEEE, 2022.

[28] Edward H Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.

[29] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. Extracting top-k insights from multi-dimensional data. In *ACM SIGMOD*, 2017.

[30] L. L. Thurstone. A law of comparative judgment. *Psychological Review*, 34(4):273–286, 1927.

[31] Manasi Vartak, Silu Huang, Tarique Siddiqui, Samuel Madden, and Aditya G. Parameswaran. Towards visualization recommendation systems. *SIGMOD Rec.*, 45(4):34–39, 2016.

[32] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[33] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock D. Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Trans. Vis. Comput. Graph.*, 22(1):649–658, 2016.

[34] X. Zhang, J. Xu, M. Keskin, M. Colaresi, and V. Zadorozhny. HR-VEAW: A human rights violation exploration analytics, and warning system. In *Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference*, volume 3135 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[35] Xiaozhong Zhang, Xiaoyu Ge, and Panos K. Chrysanthis. Leveraging data-analysis session logs for efficient, personalized, interactive view recommendation. In *IEEE CIC*, 2019.

[36] Xiaozhong Zhang, Xiaoyu Ge, and Panos K. Chrysanthis. Evaluating query strategies for different feedback types in interactive view recommendation. In *IV*, 2020.

[37] Xiaozhong Zhang, Xiaoyu Ge, and Panos K. Chrysanthis. Interactive view recommendation with a utility function of a general form. In *HILDA*, 2020.

[38] Xiaozhong Zhang, Xiaoyu Ge, Panos K. Chrysanthis, and Mohamed A. Sharaf. Viewseeker: An interactive view recommendation tool. In *BigVis Workshop*, 2019.

[39] Xiaozhong Zhang, Xiaoyu Ge, Panos K. Chrysanthis, and Mohamed A. Sharaf. Viewseeker: An interactive view recommendation framework. *Big Data Research Special Issue on Interactive Big Data Visualization and Analytics*, 2021.