ViewSeeker: An Interactive View Recommendation Tool

Xiaozhong Zhang
University of Pittsburgh
xiaozhong@pitt.edu

Xiaoyu Ge
University of Pittsburgh
xiaoyu@cs.pitt.edu

Panos K. Chrysanthis
University of Pittsburgh
panos@cs.pitt.edu

Mohamed A. Sharaf
University of Queensland
m.sharaf@uq.edu.au

ABSTRACT

View recommendation has emerged as a powerful tool to assist data analysts in exploring and understanding big data. Existing view recommendation approaches proposed a variety of utility functions in selecting useful views. Even though each utility function might be suitable for specific scenarios, identifying the most appropriate ones along with their tunable parameters, which represent the user's intention during an exploration, is a challenge for both expert and non-expert users. This paper presents the first attempt towards *interactive view recommendation* by automatically discovering the most *appropriate view utility functions* in an exploration based on the user's preferences. In particular, our proposed *ViewSeeker* uses a novel active learning method to discover the view utility function by interactively refining the set of *k* view recommendations. The effectiveness and efficiency of ViewSeeker was experimentally evaluated using both synthetic and real data sets.

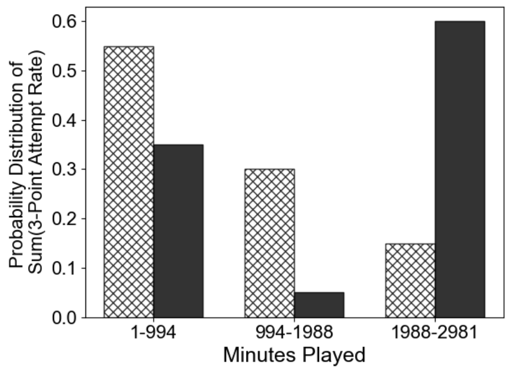


Figure 1: Example of a view providing an insight about the performance of an NBA team.

1 INTRODUCTION

The ubiquitously available information sources and the advancements in data storage and acquisition techniques have led to an aggressive increase in the data volumes available for data analysis tasks. One major challenge in utilizing these abundantly available data is discovering insights from them effectively and efficiently. Examples of an "insight" include the structure, patterns, and causal relationships. To explore these massive and structurally complicated datasets, data analysts often utilize visual data analysis tools such as Tableau [2] and Voyager [28]. However, the effectiveness of these tools depends on the user's expertise and experience. Coming up with a visualization that shows interesting trends/patterns is a non-trivial issue. Consider, for example, a view comparing the player 3-point attempt rate of a selected NBA team with that of all teams in the league (Figure 1), which could explain why the selected team on the right (black) outperformed the league average on the left (gray) and won a championship. The analyst needs to examine the relationships among various attributes and consider various aggregate functions before any useful visualizations can be discovered. This approach is typically ad-hoc, labor-intensive, and not scalable, especially for high-dimensional databases.

To address the shortcoming of the current visual analysis tools, several methods for recommending visualizations have recently been proposed (e.g., [5, 6, 11, 12, 16, 18, 27]). These methods automatically generate all possible views of data, and *recommend* the *top-k interesting* views, according to some utility function (e.g., deviations, data variance, usability) that measures the interestingness of data. Even though each utility function might be suitable for specific scenarios, identifying the most appropriate ones and their tunable parameters, which represent the user's intention during an exploration, is still a challenge for both expert and non-expert users.

Motivated by the need for supporting visualization recommendation, which is tailored to the user's exploration task, we propose an interactive view recommendation tool, called *ViewSeeker*, that efficiently assists the user to explore visually large, high-dimensional datasets. The main idea of the ViewSeeker is to automatically identify the *most appropriate utility function* based on the user's feedback on selected views. In particular, ViewSeeker iteratively presents a set of views to the user, and the user is asked to provide simple feedback to each of the presented views. Utilizing such feedback, ViewSeeker learns the most appropriate utility function by employing a novel active learning method. In each iteration, ViewSeeker effectively predicts and refines the view utility function and identify those views, from all possible views, which have high values in their utility functions. ViewSeeker ensures a sub-second runtime response time for each subsequent iteration, by employing multiple optimization techniques such as pruning, sampling, and ranking.

To verify the effectiveness of our proposed interactive view recommendation tool, *ViewSeeker*, we implemented a prototype system and experimentally evaluated it using both a synthetic dataset and a real-world dataset of diabetic patients [1]. Our evaluation results have confirmed the effectiveness of ViewSeeker, such that on average only *7-16* user answers (feedback) are required to obtain a set of highly accurate top-k view recommendations.

Specifically, the contributions of this paper are the following:

(1) Designing an interactive view recommendation tool called *ViewSeeker* that iteratively discovers the most appropriate view utility function and refines the recommended views (i.e., histograms or bar charts) with the feedback that the user provided.

(2) Proposing a suite of optimization techniques that improve the efficiency of the system, while minimizes the impact on the quality of the recommended views.
(3) Implementing a prototype system of the proposed ViewSeeker, and then verified the effectiveness of our proposed approach on both synthetic and real-world datasets.

Outline The rest of the paper is structured as follows. Section 2 introduces our problem definitions. Section 3 presents our proposed approach. Section 4 and 5 describe our experimental testbed and results. Section 6 discusses related work and Section 7 concludes the paper.

2 PROBLEM FORMULATION

In this section, we present the necessary background details of our work. Specifically, we discuss how views can be constructed through SQL queries, and then explain how the interestingness of a view may be captured through a pre-defined utility function. Afterward, we formally present our proposed problem.

2.1 Views & Data Visualization

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

$$VS = 2 \times |A| \times |M| \times |F| \quad (1)$$

Clearly, VS can be large, especially with high-dimensional data. In order to recommend the set of k most interesting views from a large number of views, utility scores are required to rank all the views. To compute such utility scores, existing literatures have proposed a large number of utility functions, some commonly used ones includes deviation [27], accuracy [5], usability [5] and p-value [26]. Furthermore, these utility functions also contain their own parameters, and any of these functions can further be combined linearly with others to form composited utility functions, thus leading to an enormous search space for the utility functions.

We use the family of deviation-based utility functions as an example to further explain the potential search space of the utility function and illustrate how the interestingness of a view may be measured. For clarity, we call each original view a target view v_i^T, which is represented as a triple (a, m, f) applied to a subset of the data D_Q that is produced by a given user query Q. In order to define the deviation, we create a helper view called the reference view v_i^R for each target view. The reference view visualizes the results of grouping the data in the whole database D with the same set of triple

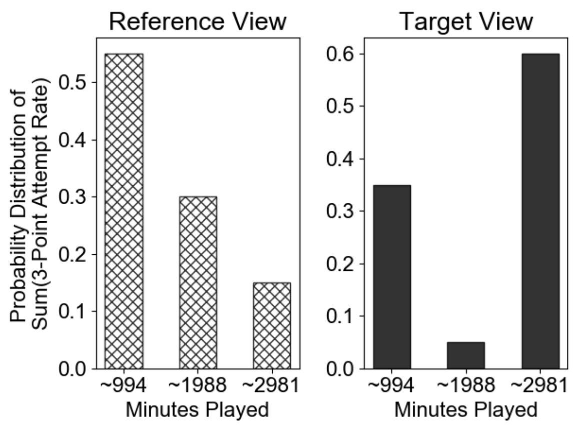


Figure 2: A target view and its corresponding reference view.

(a, m, f) used by the target view. An example of a target view with its reference view is illustrated in Figure 2.

Deviation measures the difference between the target view and the reference view with an underlying assumption that the greater the distance between the target view and the reference view, the higher the utility is. In the case of histograms or bar charts, measuring the distance between two views v_i and v_j essentially equals measuring the distance between two normalized probability distributions $P(v_i)$ and $P(v_j)$. Formally, the utility score $u(v_i)$ of a view v_i computed from deviation can be defined as:

$$u(v_i) = DT(P(v_i^T), P(v_i^R)) \quad (2)$$

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

Consequently, the typical view recommendation problem can be defined as follows:

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

2.2 Problem Settings

The above definition of a typical view recommendation problem assumes that the utility function is given [19, 21, 28, 29]. In this section, we will formalize our problem of *interactive view recommendation* in which the composition of the utility function is not given but is discovered.

Definition 2. (*View Utility Function Selection Problem*) Consider a d-dimensional database D. Further, consider a user-specified query Q, a set of results R produced by Q, a set of n possible utility function $U = \{u_1, u_2, ..., u_n\}$, and the size of the preferred view recommendations K. Find the utility function $u^p()$, which produces top-K views $V^p = \{v_1^p, v_2^p, ..., v_k^p\}$, constructed from R based on user's feedback, such that $u^p()$ can be any arbitrary combination of the utility functions in U and most accurately captures the user's ideal utility function $u^*()$.

Clearly, view recommendation with dynamic view utility function selection, is a more challenging problem compared to traditional view recommendation, given that it has a much higher search space complexity, because it combines the traditional View Space (Eq. 1) with the search space of the components of the utility functions.

$$UtilityVS = VS \times |u_1()| \times \cdots \times |u_n()| \tag{3}$$

where $u_i()$ is a utility function, $i = 1, .., n$.

That is, in the context of our problem, we are expanding the representation of a view v_i from a triple (a, m, f) to be a tuple $(a, m, f, u_1(), ..., u_n())$, and central to our solution is discovering the ideal utility function $u^*()$, interactively through user feedback:

$$u^*() = \beta_1 u_1() + \cdots + \beta_n u_n() \tag{4}$$

where β_i is the weights assigned to the corresponding possible utility function $u_i, i = 1, ..., n$. It can be seen from this equation that $u^*()$ can be any linear combination of the individual utility functions. For instance, $u^*()$ can be mapped to a single utility function u_i, in this case $\beta_i = 1$ and all other $\beta = 0$; or $u^*()$ can be a combination of multiple utility functions, where a set of corresponding β of the utility functions in $u^*()$ sum to 1 and the remaining β are set to 0.

Since our objective is to predict the user's most preferred utility function $u^*()$ with high precision, we can measure precision by the distance between the top-K views V^p, recommended by our solution using the predicted utility function $u^p()$, and those top-K views V^* produced by the ideal utility function $u^*()$. Particularly, a utility function $u^p()$, which selects top-k views closer to V^*, is considered to be more preferred than a utility function $u^{p'}()$, which selects top-k views farther away from V^*. More details on the evaluation will be provided later, in Section 4.

Definition 3. (*Interactive View Recommendation Problem*) Find the solution of the View Utility Function Selection problem such as the computational delay between each subsequent interactions (feedback) with the user is within a time constraint tl.

In each interaction between the system and the user, the user is presented with a set of M views and expected to provide feedback on the interestingness of each view. The feedback should reflect the user's belief with respect to the interestingness of each view, which would be a number ranging from 0-1, with 0 being the less interesting and 1 being the most interesting. Utilizing the feedback the system would provide better recommendations of views in the subsequent iterations. Furthermore, to enable fluent user interaction, the response time between each subsequent iteration must be within the time constraint tl, which is typically below one second.

3 THE VIEWSEEKER

In this section, we discuss the details of our proposed ViewSeeker, a novel view recommendation tool, which interactively finds the set of views that align best with the user's interest.

As shown in Algorithm 1, the ViewSeeker takes a dataset D_R to use as reference and a subset of the reference data D_Q as inputs. D_Q can be specified by any data specification method such as an SQL/NoSQL query over D_R. The output of ViewSeeker is a view utility estimator trained with user's feedback that predicts the utility of any given view generated from D_R based on user preference.

Algorithm 1 The ViewSeeker

```
Require: The raw data set $D_R$ and a subset $D_Q$ specified by a query
Ensure: The view utility estimator $VE$
 1: Unlabeled view set $U \leftarrow generateViews(D_Q, D_R)$
 2: Labeled view set $L \leftarrow$ obtain initial set of view labels
 3: $VE \leftarrow$ initialize view utility estimator $VE$ using $L$
 4: $UE \leftarrow$ initialize uncertainty estimator $UE$ using $L$
 5: loop
 6:     Choose one $x$ from $U$ using $UE$
 7:     Solicit user's label on $x$
 8:     $L \leftarrow L \cup \{x\}$
 9:     $U \leftarrow U - \{x\}$
10:     $VE \leftarrow$ refine $VE$ using $L$
11:     $UE \leftarrow$ refine $UE$ using $L$
12:     $T \leftarrow$ recommend top views using $VE$
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 $VE$
```

ViewSeeker operates in two phases: an *Off-line Initialization*, and *Interactive View Recommendation*.

3.1 Off-line Initialization

In the first phase, ViewSeeker generates all possible views in an offline pre-processing fashion to facilitate the subsequent execution. This phase proceeds in two stages. During the first, for each view v_i, ViewSeeker generates two corresponding views: the reference view v_i^R and the target view v_i^T. In particular, v_i^R shows the aggregate values from D_R and v_i^T shows the corresponding aggregate values from D_Q. Each view is generated by aggregating the data from a specific dataset along a dimension attribute A, using an aggregation function F on a measure attribute M, as discussed above.

To facilitate the computation, ViewSeeker represents each target and reference view as probability distributions, which are stored internally as vectors. Consequently, for each view v_i, two probability distributions, $P(v_i^T)$ and $P(v_i^R)$, are obtained for its target view v_i^T and reference view v_i^R, respectively. The conversion from views to probability distributions can be achieved simply through normalizations. As illustrated below, in Equation 5, we normalize each view v_i by individually dividing the aggregate value of each bin in v_i by the sum of the aggregated values of all bins in v_i, such that the sum of aggregated values of all bins in v_i would become 1.

$$P(v_i) = \langle \frac{g_1}{G}, \frac{g_2}{G}, ..., \frac{g_b}{G} \rangle \tag{5}$$

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

In the second stage, ViewSeeker would produce an internal representation for each view, which is needed for the training of the view utility estimator. As mentioned above, a variety of different utility functions have been previously proposed. To find the most appropriate one that matches the user's intention, ViewSeeker combines each possible utility function into distinct features of views. Specifically, we noticed that each previously proposed utility function is essentially a combination of one or more "utility components" (e.g., deviations, usability, accuracy). Thus, we incorporate these components as additional features of the views denoted as *utility features*. The value of a utility feature of a view v_i is the result of the corresponding "utility component" (e.g., deviation) computed on v_i.

For illustration purposes, in our current tool, we have implemented eight utility features for each view. The first five utility features are the deviation between target view and reference view with different distance measures: Kullback-Leibler divergence (KL-divergence), Earth Mover Distance (EMD), L1 distance, L2 distance and the maximum deviation in any individual bin. The last three utility features represent the usability [5], accuracy [5], and p-value [26]. Usability refers to the quality of the visualization in terms of providing the analyst with an understandable, uncluttered representation, which is quantified via the relative bin width metric. Accuracy refers to the ability of the view to accurately capture the characteristics (i.e., distribution) of the analyzed data, which is measured in terms of Sum Squared Error (SSE). The p-value is a statistical term defined as "the probability of obtaining a result equal to or more extreme than what is actually observed, with the given null hypothesis being true" [13]. In the problem of view recommendation, the null hypothesis refers to the reference view, and the extremeness of the results refers to the interestingness of the target views.

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

3.2 Interactive View Recommendation

In the second phase, ViewSeeker interactively presents views to the user and then requests user feedback on each presented view—we denote such feedback as a *label*. In each iteration, ViewSeeker would present M example views (default $M = 1$) selected from all possible views to the user, and the user is expected to express their interest with respect to each example view with a numeric label that ranges between 0.0 - 1.0 with 1.0 being the most interesting—Example values would be 0.0 (not interesting), 0.7, 0.9, 1.0.

After each iteration, ViewSeeker would use all collected feedback to train a *Linear Regression* model as the *view utility estimator* that predicts for each view the corresponding score produced by the ideal utility function $u^*()$. We choose Linear Regression as the view utility estimator because the task for predicting the utility score of a view can naturally be seen as a regression task.

It can be easily noticed that the effectiveness of our approach depends heavily on the example views being presented to the user for labeling. However, to measure exactly the benefit of each user label before obtaining it is difficult, especially given the fact that the delay between each subsequent iteration cannot exceed the time constraint tl. Fortunately, our problem of finding the most beneficial views to be labeled aligned with the objective of *Active Learning* techniques [22].

Active Learning is an interactive machine learning framework that achieves accurate classification with minimum human supervision. A typical active learning approach would employ a *query strategy* to sequentially select which unlabeled example (i.e., object) in the database should be presented to the user next for labeling. A query strategy attempts to minimize the labeling costs by selecting the most informative examples for building the classification model. Several query strategies that define the "informativeness" of examples have been proposed in the literatures (e.g., [14, 22–24]). Due to the interactive nature of ViewSeeker, we choose to use the most efficient query strategies, named *uncertainty sampling* [14], as the way to measure the benefit of labeling one view.

The intuition underlying uncertainty sampling is that patterns with high uncertainty are hard to classify, and thus if the labels of those patterns are obtained, they can boost the accuracy of the classification models. Particularly, in binary classification models (with class labels 0 and 1), the most uncertain example $\mathbf{x}$ is the one which can be assigned to either class label $z(\mathbf{x})$ with probability 0.5.

Inspired by such idea of uncertainty, also known as *least confidence*, we adopted the measurement of uncertainty proposed in [14] for binary classification models:

$$u^{(lc)}(\mathbf{x}) = 1 - p(\hat{y}|\mathbf{x}) \tag{6}$$

where $u^{(lc)}(\mathbf{x})$ is the uncertainty score with a least confidence measurement of $\mathbf{x}$; $\hat{y}$ means the predicted class label of the unlabeled $\mathbf{x}$. Accordingly, after measuring the uncertainty of each unlabeled sample, the unlabeled sample with highest uncertainty is selected:

$$\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} u(\mathbf{x}) \tag{7}$$

where $u(\mathbf{x})$ can be any other measurement of informativeness over the unlabeled sample $\mathbf{x}$.

Since estimating the uncertainty of a given view requires a probabilistic based machine learning model, the view utility estimator (i.e., non-probabilistic linear regression model) cannot be used to obtain the uncertainty score. To overcome this challenge, we employed a separate *Logistic Regression* model trained on the same set of labeled views as the view utility estimator to serve as the *uncertainty estimator*, which estimates the uncertainty of each view.

Machine learning models such as the uncertainty estimator must be trained with both positive (i.e., interesting) and negative (i.e., not interesting) views. For this reason, ViewSeeker splits its second phase into two stages as well, such that the first stage addresses the "cold start" issue of the system and the second stage quickly refines the view utility estimator to discover the set of k most interesting views.

The "cold start" issue is basically the acquisition of the first positive view. To facilitate this process, ViewSeeker would first select views ranked highest according to each utility feature (e.g., deviation, accuracy, usability, p-value). Each utility feature would then be considered in a sequential manner, such that in each iteration the set of M views ranked highest according to the current utility feature under consideration would be chosen from all possible views and then presented to the user for their feedback (i.e., labels). In the case where no positive or negative feedback has been received after visiting all dimensions, ViewSeeker will then switch to random sampling for the subsequent interactions with the user.

In the second stage of the second phase, ViewSeeker uses the uncertainty estimator to choose the most informative views to be presented to the user for feedback in each iteration. Specifically, when the output likelihood $p(\hat{y}|\mathbf{x})$ of the uncertainty estimator of a view is closest to 50%, this view is considered to be the most uncertain, and thus should be preferred to views that are less uncertain. During each iteration, ViewSeeker would sample as many distinct views as possible within the given time constraint tl and

Table 1: Testbed Parameters

Total number of records	10×10^5 (DIAB) 10×10^6 (SYN)
Cardinality ratio of records in D_Q	0.5 %
Number of dimension attributes (A)	7 (DIAB), 5 (SYN)
Number of distinct values in A	variable (DIAB), 3 and 4 (SYN)
Number of measure attributes (M)	8 (DIAB), 5 (SYN)
Number of aggregation functions	5
Number of view utility feature	8
Utility estimator	Linear regressor
Number of views presented per iteration	1
Performance measurement	Top-k precision
The number of views to recommend (k)	5, 10, 15, 20, 25, 30
Optimization partial data ratio α	10%
Optimization time limit per iteration	1 second
Optimization performance measurement	Top-k precision, Running time

then present M views with the highest uncertainty scores to the user for feedback. After each iteration, the newly acquired label(s) (i.e., feedback) would be merged with all previously obtained labels for use as the training data for the refinement of both the view utility estimator and uncertainty estimator.

After each refinement, the system will show the user the set of top-k views ranked highest according to the utility score produced by the most recent view utility estimator. Once the user is satisfied with the set of recommended views, the entire exploration process terminates, and the most recently trained view utility estimator $u^p()$ is selected as the estimation of $u^*()$.

3.3 Optimizations

In this section, we introduce the key optimizations built in ViewSeeker to speed up the interactive recommendation process while minimizing the reduction in the effectiveness of the recommendation.

Recall that in the first phase, ViewSeeker converts each basic utility component into utility features and embeds them into the vector representation of the views. During this conversion, each utility feature needs to be calculated for all possible views, which is time-consuming. As the majority of the views are typically not of the user's interest, it is unnecessary to fully compute these view utilities with the entire database. Instead, we propose an optimization that aims to compute only the set of utilities for views that are more likely to be of interest to the user.

Specifically, our optimization works as follows. During the first phase, we compute the set of utility features for each possible view only with α percent of the data, where α is a pre-defined ratio, which can be tuned based on data size, available system resources, and time constraint tl. In particular, ViewSeeker will uniformly sample α percent of data from the underlying database to produce a "rough" utility score for each utility feature of each view.

Later on, during the second phase, ViewSeeker will incrementally refine the utility score of each view with the entire set of data whenever there is spare computing power available between user labeling prompts while ensuring the time constraint tl is obeyed. In other words, the main idea is to quickly compute a set of initial "rough" values for the utility features of each view to enable subsequent interactions and then incrementally compute the accurate utility scores for each view while leveraging the user interaction time. This allows ViewSeeker to hide the necessary computation, and in turn, makes the delays transparent to the user.

Table 2: Simulated Ideal Utility Functions

#	Involved utility features and weights
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

To utilize spare computing power efficiently, it is important to determine the order of the incremental view computation. To do so, ViewSeeker uses the current view utility estimator to rank the views, and the views ranked highly would have higher priority in computing the accurate utility features. Effectively, these optimizations allow ViewSeeker to reduce the unnecessary computation by pruning out the calculations for views that are less promising.

4 EXPERIMENTAL TESTBED

We evaluated the benefit of using ViewSeeker through a simulated user study, and measured both efficiency and effectiveness. Here we present the details of our testbed (Table 1).

Setup We built a ViewSeeker platform in Python. Our experiments were performed on a Core i5 server with 8GB of RAM.

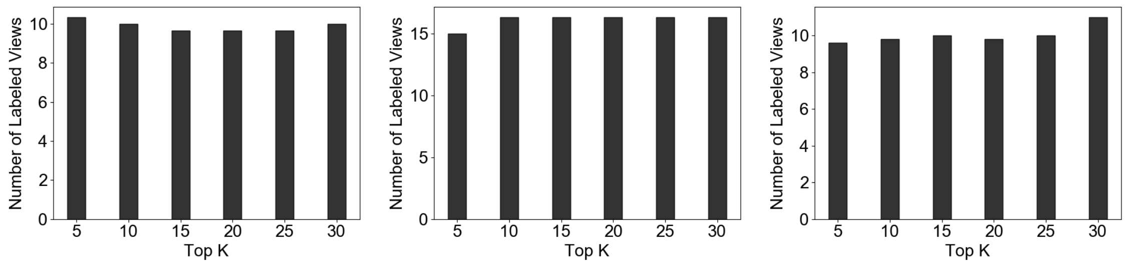
Data Sets Two datasets were used in the experiments: `DIAB` & `SYN`.

`DIAB` is a categorical dataset of diabetic patients [1]. It contains 100 thousand records. We removed the attributes that have a large amount of missing data or are very sparse. After preprocessing, the data set had 7 dimension attributes A, and 8 measure attributes M, and a total of 280 distinct views were generated.

`SYN` is a synthetic dataset with 1 million numerical records that contains 5 dimension attributes, 5 measure attributes, and 2 bin configurations (i.e., we create views with 3 bins or 4 bins). The values of the attributes of each record are uniformly distributed. The total view space for the `SYN` dataset is 250 distinct views.

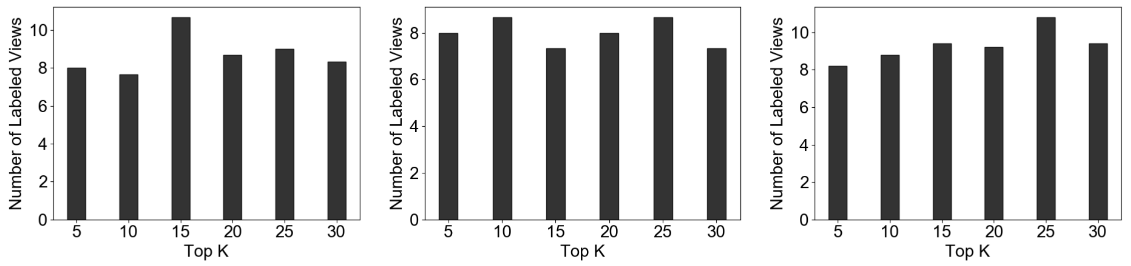
Simulated User Study Initially, we generated the view utility features in two-step: 1) we created a hypercube in the recording space to represent D_Q, which is a subset of data specified by a query; and 2) we generated view utility features based on the different utility functions mentioned in Section 3. Afterward, we simulated different user utility functions $u^*()$, each of which is a weighted sum of one or more individual utility functions. For each presented view v_i, we simulated the user's belief with respect to the interestingness of a view through the normalized utility score produced by the $u^*(v_i)$, such that $u^*(v_i) = 0.7$ indicates the interestingness of view v_i is about 70% of the maximum.

Performance We evaluated the performance of our system in the aspect of recommendation precision. In particular, we showed the effectiveness of our proposed approach by measuring the number of example views needed to reach 100% precision. Here we define



(a) Single Component $\mathbf{u}^*()$ **(b) Two Components $\mathbf{u}^*()$** **(c) Three Components $\mathbf{u}^*()$**

Figure 3: Recommendation precision for DIAB dataset with different ideal utility functions $\mathbf{u}^*()$.



(a) Single Component $\mathbf{u}^*()$ **(b) Two Components $\mathbf{u}^*()$** **(c) Three Components $\mathbf{u}^*()$**

Figure 4: Recommendation precision for SYN dataset with different ideal utility functions $\mathbf{u}^*()$.

the precision as the size of the intersection between the top-k views recommended by the ViewSeeker and the top-k views recommended by the $u^*()$. For two sets of top-k views V^p and V^* produced by ViewSeeker and $u^*()$, respectively, the precision is calculated as: $\frac{|V^p \cap V^*|}{k}$.

Simulated Ideal Utility Functions We evaluated the effectiveness and efficiency using 11 diverse ideal utility functions $u^*()$ that included 3 single component utility functions and 8 multi-component, composite utility functions (Table 2). We chose the components in multi-component $u^*()$ carefully such that they represent different characteristics of the view. For example, EMD focuses on deviations across bins, KL-divergence indicates the sum of deviation in individual bins, Usability represents the visual quality of a view, etc.

5 EXPERIMENTAL RESULTS

We evaluated both the effectiveness of ViewSeeker and its optimizations by contacting three experiments.

5.1 Evaluation of Effectiveness

We contacted two experiments to evaluate the effectiveness of the ViewSeeker. The first measured the user effort, whereas the second measured the precision of the predicted utility function.

Experiment 1: Figures 3 and 4 illustrate the effectiveness of the ViewSeeker by showing the number of example views that need to be labeled in order for the view utility estimator to reach 100% precision in the top-k recommended views. Here, the x-axis is the k in top-k (i.e., the number of views on which the precision calculation is based), and the y-axis is the number of example views presented, capturing the user effort required for the precision to reach 100%.

Specifically, in Figures 3a and 4a, we evaluated the effectiveness of ViewSeeker with respect to the ideal utility functions that contain only a single utility component (i.e., results are averaged over ideal Utility Functions 1-3 in Table 2). In Figures 3b and 4b, we repeated the evaluation for composited ideal utility functions that consists of two utility components (i.e., results are averaged over ideal Utility Functions 4-6). In Figures 3c and 4c, we repeated the evaluation for composite ideal utility functions with three utility components (i.e., results are averaged over ideal Utility Functions 7-11).

From these results, we can observe that our proposed ViewSeeker is extremely effective in discovering the set of ideal top-k views: for k ranging from *5-30*, on average only *7-16* labels were required before the ViewSeeker reached a precision of 100% for both `DIAB` and `SYN` datasets. Clearly, this indicates that only a small amount of user effort is needed before a satisfactory set of results can be obtained by the ViewSeeker.

Experiment 2: We compared the top-k recommended views by ViewSeeker with the top-k recommended views by the baselines in terms of the maximum achievable recommendation precision. We use the 8 individual utility features (e.g., KL, EMD, L1, L2, etc.) as the baselines. Figure 5 shows the result for ideal Utility Function 11 (i.e., $u^*() = 0.3 * EMD + 0.3 * KL + 0.4 * Accuracy$) in the `DIAB` dataset. We observe that the ViewSeeker achieved a 3X improvement against the best-performing baseline (i.e., EMD) in recommendation precision. This indicates that using single fixed utility features will not be able to capture a complex ideal utility function, which best captures the user's interest, such as the one above.

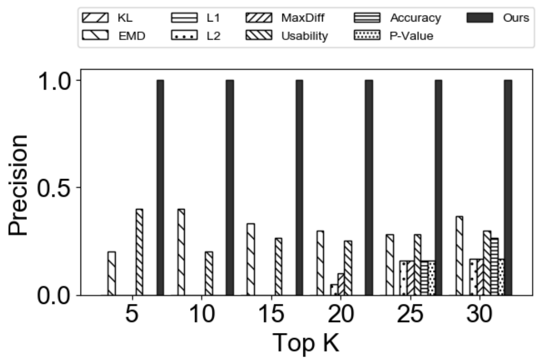


Figure 5: Precision comparison with individual utility features

5.2 Evaluation of Optimizations

We evaluated the effectiveness of our optimization techniques by comparing the recommendation precision and runtime between the optimizations-enabled ViewSeeker and the optimizations-disabled ViewSeeker (i.e., baseline model).

When comparing the recommendation precision, in order to eliminate the non-determinism in selecting the k^{th} view in top-k we introduced the concept of *utility distance (UD)*, which is defined as:

$$UD = \Big(\sum_{v_i \in V^*} u^*(v_i) - \sum_{v_i \in V^p} u^*(v_i) \Big) / k \tag{8}$$

where V^* are top-k views recommended by the ideal utility function $u^*()$, and V^p are top-k views recommended by ViewSeeker.

To explain the top-k non-determinism that motivated UD, note that views directly after the k^{th} view may have very close, or even identical, utility as the k^{th} view. In such cases, changing the order among these close views should not affect the precision too much, if any. UD has this desired property because its evaluation focuses on utility distance instead of the exact inclusion of the top-k views.

Figures 6 and 7 show the number of feedback and runtime, respectively, needed for both models to reach $UD = 0$ for the `DIAB` dataset. On average, the model with optimization achieved 43% reduction in running time while requiring only 19% more user labeling effort.

The experiment result confirmed that our optimization methods are effective in reducing computing time by pruning out calculating for views that are less promising. The increase in the required labeling effort is because the view utility features computed on partial data is only an estimation and may be discrepant from the true features, which hinders the learning of the ideal utility function.

6 RELATED WORKS

In this section, we review works that are strongly relevant to ours.

View Recommendation techniques automatically generate all possible views of data, and recommend the top-k interesting views, according to some utility function (e.g., [5, 10, 16, 18, 19, 21, 27–29]). A key difference among these works is the proposed utility functions. Recent work placed addition constraints, e.g., upper bound on the number of views to be explored and execution time limit when computing the recommended views [10]. The most close work to ours is the use of generic priors (i.e., general knowledge of how people associate views with different datasets and exploration tasks) to train a general machine learning model that predicts the utility score of any given view [16]. The generic priors were obtained by hiring a large number (i.e., 100) of human annotators to annotate multiple real-world datasets. The key difference between our work and all prior work is that all previous works use predefined view utility functions and do not discover the utility function that best matches an individual user's intention and exploration task.

Interactive Visualization Tools have been studied extensively for the past few years [3, 7, 9, 11, 12, 15, 17, 25]. Unlike visualization recommendation tools such as ViewSeeker that recommend visualization automatically by searching through the entire views spaces, traditional interactive visualization tools require the user to manually specify the views to be generated. Recently, a few interactive visualization tools have attempted to automate part of the data analysis and visualization process. For instance, Profiler automatically helps analysts detect anomalies in the data [11]. But, unlike our approach, Profiler is not capable of providing general recommendations for any group-by queries. Another recent example is VizDeck [12], which generates all possible 2D views for a given dataset and allows the user to manually control (e.g., reordering, pinning) these views through a dashboard, rather than using a utility function. The work [3] is the closest to our work in the sense that it also uses user feedback to steer the view exploration. However, the user feedback is only used to train a classifier, which is not capable to estimate the ideal utility function and get the top-k views.

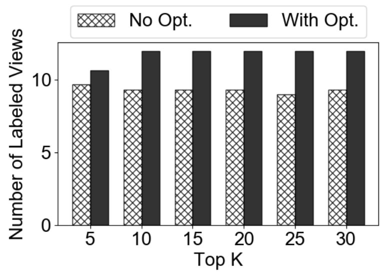
Data Exploration techniques that aim to efficiently extract knowledge from data [20] are complementary to our work. In particular, *example-driven* data exploration approaches [4, 8] assume minimum prior knowledge of the data and share the same underlying approach as ViewSeeker. These works aim to iteratively construct the exploratory query through user interactions as ViewSeeker iteratively discovers the utility function using user feedback. ViewSeeker is well suited to such situations and can enhance example-driven data exploration by creating visualizations that illustrate interesting pattern during the construction of the exploratory queries.

7 CONCLUSION

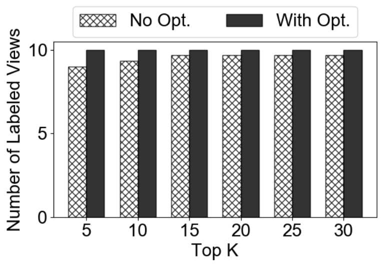
In this work, we addressed the challenge in visual analytics tools of recommending the set of views most preferred by the user during an exploration task. Our contribution, ViewSeeker, is a novel, interactive view recommendation tool that offers a solution to the fundamental challenges of: 1) providing effective results with minimum user effort, 2) enabling efficient and scalable computation, and 3) requiring no special expertise from the users.

The crux of ViewSeeker is the discovery of the utility function, which is used to select the views that best match the user's exploration task. ViewSeeker employs an active learning-based predictive model and effectively learns the user's preferred views through simple interactions between the user and the system. To enable fluent user interactions, we proposed a set of optimization techniques that significantly improved the runtime efficiency of the ViewSeeker. Our experimental results, using both synthetic and real-world datasets, showed that our proposed ViewSeeker outperforms the alternative baseline approaches by a significant margin.

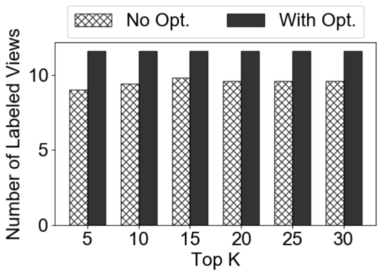
In the future, we plan to conduct a user study to validate the recommendation effectiveness of ViewSeeker, and to extend it to support more visualization types, such as scatter plot, line chart etc.



(a) UD = 0 for Single Component $\mathbf{u}^*()$

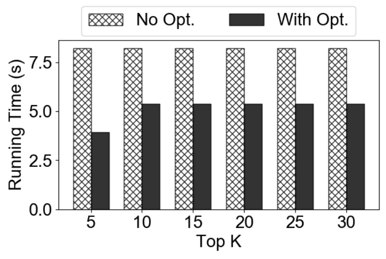


(b) UD = 0 for Two Components $\mathbf{u}^*()$

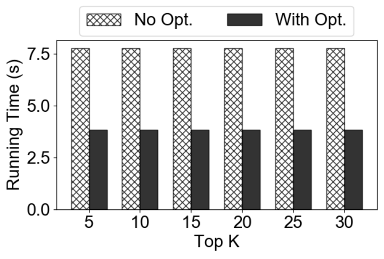


(c) UD = 0 for Three Components $\mathbf{u}^*()$

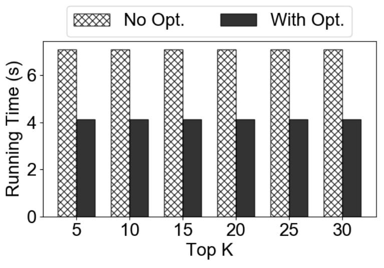
Figure 6: Recommendation Precision with optimization for DIAB dataset with different ideal utility functions $\mathbf{u}^*()$.



(a) RT for Single Component $\mathbf{u}^*()$



(b) RT for Two Components $\mathbf{u}^*()$



(c) RT for Three Components $\mathbf{u}^*()$

Figure 7: System Runtime (RT) with optimization for DIAB dataset with different ideal utility functions $\mathbf{u}^*()$.

Acknowledgments We would like to thank the anonymous reviewers for their helpful comments. This work was supported, in part, by NIH under award U01HL137159. This paper does not represent the views of NIH.

REFERENCES

[1] [n. d.]. https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes. ([n. d.]).
[2] [n. d.]. Tableau public. http://public.tableau.com. ([n. d.]).
[3] Michael Behrisch, Fatih Korkmaz, Lin Shao, and Tobias Schreck. 2014. Feedback-driven interactive exploration of large multidimensional data supported by visual classifier. In *IEEE VAST*.
[4] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. 2014. Explore-by-example: an automatic query steering framework for interactive data exploration.. In *ACM SIGMOD*.
[5] Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis. 2016. MuVE: Efficient Multi-Objective View Recommendation for Visual Data Exploration. In *IEEE ICDE*.
[6] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. 2018. Efficient Recommendation of Aggregate Data Visualizations. *IEEE TKDE* 30, 2 (2018), 263–277.
[7] D. Fisher. 2007. Hotmap: Looking at Geographic Attention. *IEEE TVCG* 13, 6 (2007), 1184–1191.
[8] Xiaoyu Ge, Yanbing Xue, Zhipeng Luo, Mohamed A. Sharaf, and Panos K. Chrysanthis. 2016. REQUEST: A Scalable Framework for Interactive Construction of Exploratory Queries. In *IEEE International Conference on Big Data*.
[9] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. 2010. Google fusion tables: web-centered data management and collaboration. In *ACM SIGMOD*.
[10] Ibrahim A. Ibrahim, Abdullah M. Albarrak, Xue Li. 2017. Constrained Recommendations for Query Visualizations. *Knowl. Inf. Syst.* 51, 2 (2017), 499–529.
[11] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Profiler: integrated statistical analysis and visualization for data quality assessment. In *ACM AVI*.
[12] Alicia Key, Bill Howe, Daniel Perry, and Cecilia R. Aragon. 2012. VizDeck: self-organizing dashboards for visual analytics. In *ACM SIGMOD*.
[13] Martin Krzywinski and Naomi Altman. 2013. Significance, P values and t-tests. In *Nature methods*.
[14] David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *ACM SIGIR*.
[15] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. 1997. DEVise: Integrated Querying and Visual Exploration of Large Datasets. In *ACM SIGMOD*.
[16] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *IEEE ICDE*.
[17] J. Mackinlay, P. Hanrahan, and C. Stolte. 2007. Show Me: Automatic Presentation for Visual Analysis. *IEEE TVCG* 13, 6 (2007), 1137–1144.
[18] Rischan Mafrur, Mohamed A. Sharaf, and Hina A. Khan. 2018. DiVE: Diversifying View Recommendation for Visual Data Exploration. In *ACM CIKM*.
[19] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing visualization design knowledge as constraints: actionable and extensible models in Draco. *IEEE TVCG* 25, 1 (2019), 438–448.
[20] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2017. New Trends on Exploratory Methods for Data Analytics. In *VLDB*. 1977–1981.
[21] Belgin Mutlu, Eduardo Veas, and Christoph Trattner. 2016. Vizrec: Recommending personalized visualizations. *ACM TiiS* 6, 4 (2016), 31.
[22] Burr Settles. 2009. *Active learning literature survey*. TR. U. Wisconsin-Madison.
[23] Burr Settles and Mark Craven. 2008. An Analysis of Active Learning Strategies for Sequence Labeling Tasks. In *ACL EMNLP*.
[24] H. S. Seung, M. Opper, and H. Sompolinsky. 1992. Query by Committee. In *ACM COLT*.
[25] Chris Stolte and Pat Hanrahan. 2000. Polaris: A System for Query, Analysis and Visualization of Multi-Dimensional Relational Databases. In *IEEE INFOVIS*.
[26] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *ACM SIGMOD*.
[27] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. 2015. SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *PVLDB* 8, 13 (2015), 2182–2193.
[28] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE TVCG* 1 (2016).
[29] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *ACM CHI*. 2648–2659.