Leveraging Data-Analysis Session Logs for Efficient, Personalized, Interactive View Recommendation

Xiaozhong Zhang, Xiaoyu Ge and Panos K. Chrysanthis
Department of Computer Science, University of Pittsburgh
{xiaozhong, xiaoyu, panos}@cs.pitt.edu

***Abstract*—View recommendation has been recently adopted to assist data analysts in better understanding the data. In order to recommend useful views, existing view recommendation approaches propose a variety of utility functions, each suitable for a different usage scenario. However, as the "interestingness" of a recommended view is user-dependent, no single utility function can represent users' preferences and intentions in all cases. With the richly available choices for utility functions, identifying the most appropriate ones along with their tunable parameters remains a challenge even for expert users. To help identify the most appropriate utility function, existing works have made attempts in two different directions, 1) providing generic recommendations of utility functions by learning offline from historical logs, and 2) providing personalized recommendations of utility functions by learning from the interactions with each particular user. Both proposed approaches exhibit clear advantages and disadvantages. In this work, to benefit from both approaches, we device a novel hybrid interactive view recommendation solution, namely *HolisticViewSeeker* (HVS), that effectively combines the offline learning with the online interactive learning to provide personalized view recommendation. Our experimental evaluations conducted on real-world data show that HVS outperforms both state-of-the-art online and offline approaches by a significant margin in multiple respects.**

I. Introduction

A. Motivation

Visual data analysis tools such as Tableau [2] and Voyager [29]. have been extensively utilized in facilitating knowledge workers (e.g., scientists, researchers, and data analysts) in exploring and making insightful discoveries (e.g., structure, patterns, and causal relationships) on large and complex datasets obtained from various data sources. However, the effectiveness of these tools depends heavily on the expertise and experience of the user. Consequently, producing a visualization that captures interesting trends/patterns is a non-trivial issue. Consider, for example, a view comparing the player 3-point attempt rate (3PAr) of a selected NBA team with that of all teams in the league (Figure 1), which may explain why the selected team on the left (black) outperformed the league average on the right (gridded) and won a championship [5], [6]. In this figure, the 3-point attempt rate is a measure attribute that contains measurable value and can be aggregated (e.g., sum, max, min), and the minutes played (MP) is a dimension attribute, which is the attribute on which the measure attribute (i.e., 3PAr) are viewed.

Particularly, Figure 1 shows that for all NBA players [1], the 3PAr decreases as they play more games throughout the season. As it was observed in [5], [6], this is perfectly understandable since the fatigue incurred from playing more games can affect their fitness and reduce their 3PAr. Interestingly, for the Golden State Warriors (GSW) players (black), that pattern is very different from that general pattern. As Figure 1 shows, the 3PAr performance of GSW players was not affected by the time spent on the field. In fact, their 3PAr is almost four times that of the players in the other teams during the last third of the season. This indicates the high fitness and consistency of the GSW players that distinguishes them from the other players in the league. It might also indicate a good play strategy in distributing 3-point attempts throughout the season. Both can shed some light on understanding GSW's

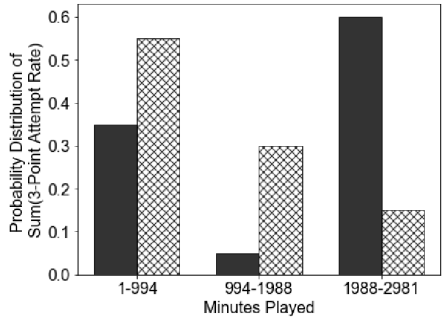

Fig. 1: Example of a view providing an insight about the performance of Golden State Warriors (black) VS. all NBA teams (gridded).

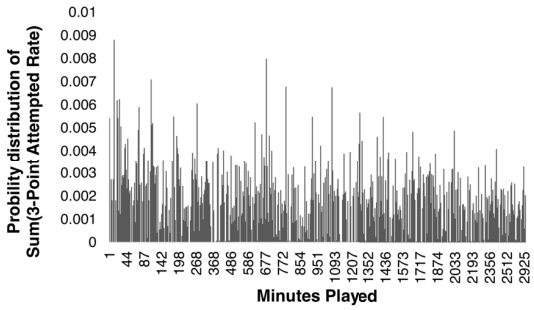

Fig. 2: View on the 3PAr of all players in the 2015 NBA.

978-1-7281-6737-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CIC48465.2019.00022

championship win. In contrast of Figure 1, the view on all NBA players in Figure 2 hardly reveals any useful insights. As illustrated in the above example, to obtain useful insights, the analyst needs to constantly examine the relationships among various attributes and consider various aggregate functions before reaching interesting discoveries [5]. Typically, such an approach is ad-hoc, labor-intensive, and not scalable, especially for high-dimensional datasets.

To address this challenge, several methods for recommending visualizations have recently been proposed (e.g., [28], [14], [13], [5], [6], [19], [17]). These methods automatically generate all possible views of data, and *recommend* the *top-k interesting* views, according to some utility function (e.g., *deviations* [28], *usability* [5], *diversity* [11]) that measures the interestingness of data. Even though each utility function might be suitable for certain scenarios, identifying the most appropriate ones and their tunable parameters for a given user and exploration task remains a challenge even for expert users.

B. The Problem

Recent works [30], [20] have proposed techniques to train a predictive model that recommends the most appropriate utility function for different exploration tasks. These predictive models are learned in two different ways. One way to construct these predictive models is by learning from a large collection of multi-user historical session logs that are commonly available in data analytic platforms [20]. Such an offline learning method would produce a *generic* predictive model. A second way to construct these predictive models is by learning from the interactions between the machine and each specific user, which leads to a *personalized* predictive model. We denote the former as *offline learning approach* and the latter as *online learning approach*. Clearly, the offline learned predictive model demands less user effort during the exploration. However, as the objective of the data exploration is dynamic and will change according to the dataset, task and the user that is performing the task, thus, provides a recommendation solely based on offline learned models often fail to match the user's true intention. On the other hand, the online learned predictive model requires more user effort during the exploration, but it provides a personalized recommendation that accurately reflects the user's real intent. Thus, the specific challenge our paper addresses is bridging the gap between these two approaches for personalized view recommendation.

C. Our Solution

Motivated by the need to support *personalized* visualization recommendation, while minimizing user effort, we propose a novel hybrid *interactive view recommendation* solution, namely *HolisticViewSeeker* (HVS) for efficient exploration of large, high-dimensional datasets that combines the advantage of both offline and online learning approaches. HVS focuses on the interactive view recommendation while using the offline learned predictive model to boost the online learning process and in turn, reduce user effort.

The main idea of HVS involves two stages, 1) an offline processing stage and 2) an online interaction stage. During the offline processing stage, HVS performs an analysis of the historical exploration session logs and identify unique patterns or characteristics in the interactive data analysis (IDA) sessions. Specifically, we train a predictive model to recommend the utility function (i.e., measure) that best reflects the user's intention at a given point during a user's exploration. Since this recommended utility function is learned based on the historical session logs that consist of multiple users, it most likely does not reflect the preference of one individual user. Therefore, during the online interaction stage, we use the recommended utility function obtained from the previous stage as a "good" starting point and utilize subsequent user interactions to refine the recommendation as well as the predictive model towards each user's specific needs.

We define a user interaction as a sequence of actions, where the system first selects a view from all potential views that can be generated at a given point of a user's exploration session, and then the user provides simple numeric feedback (e.g., 80%) that reflects their confidence in the relevance of the selected view with respect to the exploration. Subsequently, HVS learns and identifies the *most appropriate utility function* based on user feedback on all selected views.

To verify the effectiveness of our proposed solution, we implemented a testbed and experimentally evaluated HVS using real-world datasets obtained from [20]. Our evaluation results have confirmed HVS' effectiveness. When compared to the state-of-the-art offline learning approach, HVS is able to reach 100% accuracy in all test cases, in contrast, the offline learning approach only achieves around 63% to 69%. Furthermore, when compared to the state-of-the-art online learning approach, HVS reduced overall the user effort by up to 42%. As modern data analysis platforms are well optimized for efficiency, the user effort is becoming the dominating factor in determining the productivity of ad-hoc data analysis tasks, a 42% reduction in user's effort can, therefore, be translate to a 72% improvement in the overall productivity of the analysis tasks given a fixed number of working hours.

D. Summary of Contribution

The key contributions of this paper are the following:

1) Designing a novel view recommendation solution, coined *HolisticViewSeeker* (HVS), that efficiently and effectively identifies the user's "ideal" utility function, and in turn, recommends the most appropriate set of views, tailored for each user and exploration task.
2) Proposing a wisely defined learning problem that enables the combination of the offline and online learning approach to minimize the user effort required for producing personalized view recommendations.
3) Implementing a testbed of our proposed HVS solution, and showing the effectiveness and efficiency of our proposed approach on real-world datasets using six combinations of offline/online machine learners.

Outline The rest of the paper is structured as follows. Section II defines formally our problem. Section III presents our proposed approach. Section IV and V describe our experimental testbed and results, respectively. Section VI discusses related work, and Section VII concludes the paper.

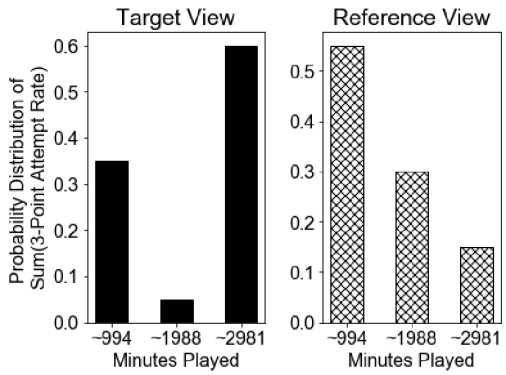

Fig. 3: A target view and its corresponding reference view.

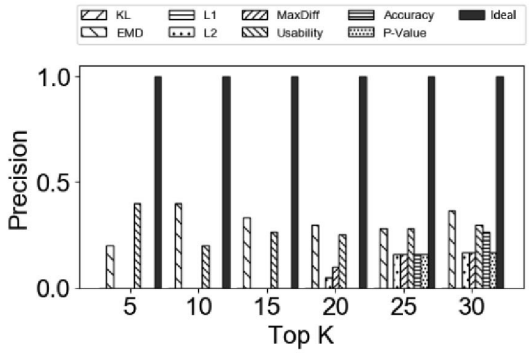

Fig. 4: Precision comparison of eight utility measures [30]: Kullback-Leibler divergence (KL), Earth Mover Distance (EMD), L1 distance (L1), L2 distance (L2), Maximum deviation in any individual bin (MaxDiff), usability, accuracy, and p-value.

II. Background

In this section, we present the necessary background details of our work. We first 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 present the problem of *Interactive View Recommendation*.

A. 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 dimension attributes $A = \{a_1, a_2, a_3, ...\}$ and a set of measure attributes $M = \{m_1, m_2, m_3, ...\}$. 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 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 used to group the data, then the values of the measure attribute m in each group are aggregated using function f.

As illustrated in Figure 3, a standard approach for measuring the *interestingness* of a view v_i is to create a reference view v_i^R and a target view v_i^T. The reference view v_i^R visualizes the results of grouping the data in the whole database D by a, and then aggregating the values in m with f, whereas the target view v_i^T represents a view with the same set of triple (a, m, f) applied to a subset of the data D_Q that is produced by a given user query Q. 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 target views, utility scores are required to rank all the target views. To compute such utility scores, existing literature has proposed a large number of utility functions; some commonly used ones include *deviation* [28], *accuracy* [5], *usability* [5], and *diversity* [11]. Furthermore, these methods also contain their own parameters, and any of these methods can further be combined linearly with other methods to form composite utility functions, thus leading to enormous search space for the utility functions. Figure 4 (obtained from [30]) compares 8 different utility functions with an ideal utility function that captures the user's intent. Clearly, as shown in this figure, when an inappropriate utility function is used, the recommend view can deviate greatly from the views that are truly preferred by the user, thus significantly hinders the usability of the recommended views.

In general, a 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.

B. Interactive View Recommendation

The above definition of a typical view recommendation problem assumes that the utility function is given. In this section, we will formalize the problem of *interactive view recommendation*, which is the problem that lies in the epicenter of this work. The interactive view recommendation assumes that the composition of the utility function is not given but is discovered.

Definition 2: (*Interactive View Recommendation 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 functions $U = \{u_1, u_2, ..., u_n\}$, and the size of the preferred view recommendations k. The goal is to interactively find the utility function $u^*()$, which produces a set of k views $V^* = \{v_1^*, v_2^*, ..., v_k^*\}$, constructed from R and most accurately captures the user's preference. Such that $u^*()$ can be any arbitrary combination of the utility functions in U.

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

$$InteractiveVS = VS \times |u_1()| \times \cdots \times |u_n()| \quad (2)$$

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 in the form of (a, m, f) to a tuple in the form of $(a, m, f, u_1(), ..., u_n())$, and central to our solution is discovering the ideal utility function $u^*()$ interactively through user feedback, specifically:

$$u^*() = \beta_1 u_1() + \cdots + \beta_n u_n() \quad (3)$$

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 β_i = 1, and all other β = 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^*.

Definition 3: (*Interactive View Recommendation Problem*) Find the solution of the View Utility Function Selection problem, such as the computational delay between each subsequent interaction (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 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.

III. The HolisticViewSeeker

In this section, we formally present our proposed *HolisticViewSeeker* (HVS) for interactive view recommendation. Our solution operates in two stages: 1) *offline processing* stage and 2) *online interaction* stage. In the offline processing stage, we train an offline learner that utilizes an *interactive data analysis log* to recommend the utility function ${u^p}_r$ for each distinct result r contained in the interactive data analysis logs. In the online interaction stage, the online learner uses the utility function ${u^p}_r$ recommended by the offline learner to initialize its estimation of the ideal utility function ${u^*}_r$ for the current result r and continues refining ${u^p}_r$ by interacting with the current user.

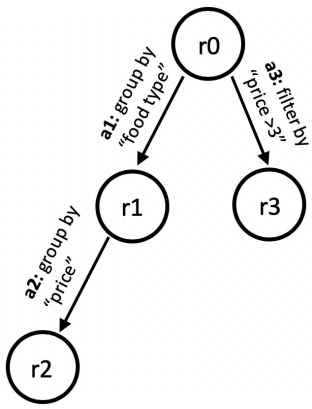

Fig. 5: An Example of an IDA Session Tree

A. Interactive Data Analysis Log

An *Interactive Data Analysis* (IDA) session log [20] records and organizes interactive data analysis actions during an exploration session. An IDA session initiates when a dataset D is loaded into an analysis system. Subsequently, during an IDA session, the user would perform a series of actions (e.g., filter, group, sort, project) $a_1, a_2, .., a_n$ and then examine each one of them to refine the search space gradually. We denote the result of an acton a_t, performed at time t as r_t.

Initially, each session starts with result r_0, which represents the initial state (i.e., root state) of the dataset before any action is performed. Furthermore, during an IDA session, the user can backtrack to any previous result and then take an alternative path from that result, thus building the log in a tree-like structure. In light of this workflow pattern, we model each IDA session as an ordered tree, denoted as T, where each result r_i represents a node of T, and each performed action a_i represents an edge of T. Figure 5 illustrates an example of an analysis session tree. Here in the tree, directed edges represent actions a_1 (group be "food type"), 1_2 (group by "price"), and a_3 (filter by "price> 3"), and the nodes r_1 - r_3 represent their corresponding results. The root node, r_0, represents the initial state of the dataset before any action was invoked.

B. Offline Processing Stage

In this stage, our goal is to train a utility function prediction model, which takes in a result r and makes the prediction ${u^p}_r$ of the utility function. In order to train the model, we need first to perform two tasks. The first task is to generate a feature vector v_r for each result r that represents what the user sees at each step. The second task is to obtain the ground truth (i.e., the ideal utility function ${u^*}_r$) for each result r from the interactive data analysis log, which will be used as the label for each v_r.

Result Representation For the first task, we need to form a feature vector v_r to represent what the user sees at the current

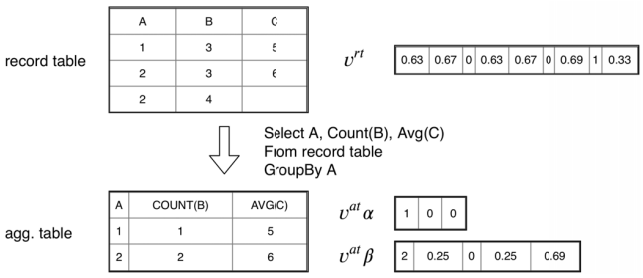

Fig. 6: Representation for Record and Aggregation Table

result r. Usually, in an interactive data analysis session s, there are two kinds of tables that are shown to the user: record tables and aggregation tables. A record table RT_r contains 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 r_0. An aggregation table AT_r contains aggregated values that are generated by applying the collection of grouping actions on the path between the current result r, and the root result r_0 on the corresponding record table RT_r.

In order to produce the feature vector that represents the record table, the simplest way is to use the record table as the representation directly. 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 the scenario of interactive data analysis 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 opt for using metadata of the records to form the representation of the record table. After comparing a variety of candidate measures, we land on three measures that capture different characteristics of the records in the record table. The first measure is the *entropy of the data*, which measures the amount of information contained in the data. The second measure is the *number of unique values in the data*, which measures the value diversity of the data. The third measure is the *number of null values in the data*, which captures local patterns that are present in the attributes. We calculate the three measures for each attribute (i.e, column) of the current record table RT_r and concatenate them to form the feature representation v^{rt} for RT_r.

Example 1: Consider the `record table` in Figure 6. The `record table` has three attributes A, B and C, and its vector representation v^{rt} has 9 dimensions. They are *entropy*, *unique value ratio* and *null value ratio* of the three attributes, respectively. □

Aside from the record table, the user might also be presented with an aggregation table AT_r of the current result r. Again, using the aggregation table itself as the representation might not be feasible, because the aggregation table can also be very large, especially for group-by attributes that have a large number of unique values. Therefore, we again turn to metadata to represent the aggregation table.

We designed two vectors ${v^{at}}_\alpha$ and ${v^{at}}_\beta$ to describe different aspects of the table and concatenate them to form the representation v^{at} of the aggregation table. The first vector ${v^{at}}_\alpha$ records the group-by attributes. To represent the group-by attributes, we set the length of ${v^{at}}_\alpha$ to be equal to the arity of the record table and set the bit to 1 if the corresponding attribute is a group-by attribute. The second vector ${v^{at}}_\beta$ describes the characteristics of the aggregated values for the aggregation functions in the aggregation table. Specifically, it records the number of groups, group size variance as well as variance for COUNT function and entropy for other aggregation functions for each attribute. We explain the vector representation of an aggregation table in the following example.

Example 2: Continuing with the previous example (Example 1) in Figure 6, assume that the aggregation table v^{at} is generated using the query:

```
SELECT A, COUNT(B), AVG(C)
FROM RECORD TABLE
GROUP BY A;
```

The vector representation of v^{at} is formed from the concatenation of vectors ${v^{at}}_\alpha$ and ${v^{at}}_\beta$ as defined above.

The vector representation ${v^{at}}_\alpha$ will have 3 dimensions with values of $(1, 0, 0)$ because only attribute A is used in the GROUP BY clause.

The vector representation ${v^{at}}_\beta$ will have 5 dimensions. The 1st and 2nd dimensions are number of groups and group size variance. The 3rd dimension is 0 because there is no aggregation for attribute A. The last two dimensions are variance of COUNT(B) and entropy of AVG(C) across the groups. □

In order to get the vector representation of a result v_r, we concatenate the vectors ${v^{at}}_\alpha$ and ${v^{at}}_\beta$ to form the vector representation of v^{at} (as in Example 2), which is then concatenated with v^{rt} (as in Example 1).

Ideal Utility Function Representation In the second task, we need to find the ideal utility function ${u^*}_r$ corresponding to the current result r.

Recall that, in our study, an ideal utility function u^* is a linear combination of a set of *individual utility measures* (e.g., diversity, concentration, usability) $\{u_1, u_2, ..., u_n\}$ in the form of $u^* = \beta_1 u_1 + \cdots + \beta_n u_n$ (Eq. 3).

Therefore, finding the ideal utility function u^* is equal to finding the optimal weights $\{\beta_1, \beta_2, ..., \beta_n\}$ of u^*. Here, we rely on the interactive data analysis log to help us find these optimal weights by analyzing the subsequent actions of each result r_i in an analysis session. Without loss of generality and for simplicity, we first look at the scenario when only one subsequent action a_{i+1} exits. In the cases when a_{i+1} is a grouping action, then the result of a_{i+1} will actually be a view. Since the user chose this view among all possible views, we assume that this view should have a high value according to the ideal utility function u^* in the user's mind. To give a more concrete explanation, if the chosen view v_1 has a high value in one individual utility measure u_1 and low values in all other u_i's, and a non-chosen view v_2 has a similar high

Algorithm 1 The Online Interaction Stage

Require: The raw data set D_R, a subset D_Q specified by a query and a initial view utility estimation ${u^p}_r$
Ensure: A fine-turned and personalized view utility estimation ${u^p}_r$
1: Unlabeled view set $U \leftarrow generateViews(D_Q, D_R)$
2: $L \leftarrow \emptyset$
3: $QS \leftarrow$ initialize query selector
4: **loop**
5: Choose one x from U using QS
6: Solicit user's label on x
7: $L \leftarrow L \cup \{x\}$
8: $U \leftarrow U - \{x\}$
9: ${u^p}_r \leftarrow$ refine ${u^p}_r$ using L
10: $QS \leftarrow$ refine QS using L
11: $T \leftarrow$ recommend top views using ${u^p}_r$
12: **if** the user is satisfied with T or the user wants to stop **then**
13: Break
14: **end if**
15: **end loop**
16: Return the most recent ${u^p}_r$

value in u_2 and low values in all other u_i's, then v_1 should be ranked higher than v_2 based on the u^* in the user's mind.

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 surprisingly simple way to discover such weights for u^*, that complies to 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. As we will show later in the experiments (in Section V), this approach is simple yet effective, and therefore, we adopted it in our study. In the cases when multiple subsequent actions exist for the current result r, we use the average values of the utility measures 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 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 [3] to transform the distributions of the u_i's into normal shapes. Then we use $0-1$ scaling to ensure that all distributions have a similar scale.

Offline Model Learning After generating the feature vectors for the results r's and the corresponding ideal utility functions ${u^*}_r$'s (i.e., the weights $\{\beta_1, \beta_2, ..., \beta_n\}$) as the labels, we are able to train the offline model. HVS can adopt a variety of regression models, such as *K-Nearest Neighbor* and *Support Vector Regressor*, as its offline model to make the prediction ${u^p}_r$. During the model training, the regression model learns the weight prediction ${u^p}_r$ based on all the features and the corresponding labels of the training results r. Later, during the inference, the model uses regression to predict the weights (i.e., $\{\beta_1, \beta_2, ..., \beta_n\}$ of ${u^p}_r$) for the current results r, to be used as the initial weights of the online model.

To summarize, the goal of the offline processing stage is to train an offline model that takes the representation of a result r and returns a predicted utility function ${u^p}_r$, such that views that are ranked higher by the ideal utility function ${u^*}_r$ should also be ranked higher by ${u^p}_r$. We make the prediction ${u^p}_r$ for the result r using a utility function that is close to the utility functions of other results that are similar to r. By doing so, we are leveraging the interactive data analysis log to help the offline model generate a good estimation of the ${u^*}_r$ for the current result r. This estimated utility function ${u^p}_r$ will be used to initialize the learning in the online interaction stage, which will be introduced in the next section.

C. Online Interaction Stage

During an exploration session, the online interaction stage has the user in the loop and uses user feedback on example views to refine the online model's estimation of the ideal utility function for the current result r. The online interaction stage takes an iterative form. In each iteration, there are four major steps as illustrated in Algorithm 1:

1) (Line 5) The model selects a view based on the current view utility estimation ${u^p}_r$ (i.e., estimation of ${u^*}_r$) among all possible views that can be generated by current result r.
2) (Line 6) The model solicits user feedback on the example view. The feedback takes the form of a number from 0 to 1 that approximates the user's confidence in the relevance of this view with respect to the user's exploration.
3) (Line 7-10) The model refines ${u^p}_r$ based on the newly labeled view.
4) (Line 11-14) The model provides a ranking of all possible views based on the refined ${u^p}_r$. If the user is satisfied with the list, then the system returns the list as the final recommendation. Otherwise, the system continues to the next iteration.

Below we will elaborate the workflow of the online interaction stage and the above four major steps in more details.

Online Interaction Stage Workflow: Before we start the first iteration, we perform an initialization method *generateView*, which takes both the original data set D_R and a subset $D_Q \subset D_R$ as input. Based on the inputs, *generateView* would first generate the vector representation of the current result r in the same way as we did in the offline stage. Later, *generateView* generates the individual utility measures $\{u_1, u_2, ..., u_n\}$ for all the possible views that can be generated from the current result r through all combinations of dimension attributes A, aggregation functions F, and measure attributes M. Then each u_i will be normalized in the same way as the normalization performed in the offline stage.

We now introduce in detail each step of the online interaction stage.

In **Step 1**, the online model selects example views for user feedback based on the current ${u^p}_r$, which is initialized through the offline processing stage. Clearly, the selection of the example views has a large influence on the learning speed of the online model. Thus, in our study, we adopt the active search approach [8] to serve as the query selector QS for the selection of the example views. According to active search, the views that have the highest utility according to ${u^p}_r$ are selected as examples and are presented to the user for feedback. We

TABLE I: 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
Offline model	K-Nearest Neighbor
Online model	Linear Regressor
Performance measurement	Top-k precision
Baseline Models	Offline, Online
Recommend view count (k)	5,6,7,8,9,10
Training set ratio	0.1,0.2,0.3,0.4
Runs for each configuration	5

adopt an active search based on two assumptions: i) The user is usually more concerned about the correct rankings of the views with higher utility than the views with lower utility. This assumption urges us to be more accurate in the estimation of $u^*{}_r$for high-utility views; and ii) The $u^*{}_r$ combines the individual utility measures $\{u_1, u_2, ..., u_n\}$ linearly.

In **Step 2**, the user simply provides a numeric score that reflects their confidence on the relevance of this presented example with respect to the exploration.

In **Step 3**, the model refines $u^p{}_r$ using the acquired feedback. HVS can adopt a variety of regression models as its online model. In our study, we use *Linear Regression* and *Multi-layer Perceptron Regression* models as examples for the online predictive model. After weights have been initialized, during the fine-tuning stage, the weights of the online predictive model will be updated with an iterative learning method such as stochastic gradient descent.

In **Step 4**, we generate view recommendations based on the current $u^p{}_r$. Specifically, we apply $u^p{}_r$ on the individual utility measures of each possible view and rank the views based on the utility scores. The output of the online interaction stage is a list of recommended views that suites the user's intention.

IV. Experimental Testbed

In this section, we describe the settings and evaluation metrics of our experiments.

A. Experiment Setup

We implemented our HVS and two state-of-the-arts baselines [20] and [30] in python. Our experiments were performed on a Linux server with 32 Intel® Xeon® CPU E5-2650 v2 cores and 96GB RAM. A summary of the main settings of our experiments is shown in Table I.

Dataset The dataset we used in our study is the REACT-IDA dataset[22], 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 removed non-grouping actions from the dataset because the $u^*{}_r$ cannot be calculated without aggregated values. We also skipped the actions that start from the initial result r_0, to focus on the utility function prediction after the user has submitted at least one query.

TABLE II: Individual Utility Measures

Measure	Formula	Reference
Diversity (D)	$\sum_{j=1}^{m} p_j^2$	[25]
Concentration (C)	$1 + H(\frac{p_j+\overline{q}}{2}) - \frac{\log_2 m - H(p_j)}{2}$	[11]
Outlier Score (O)	See [15]	[15]
Usability (U)	$1 - \frac{\min(\log m, c)}{c}$	[24]

TABLE III: Average number of IUF (for each result r)

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

Machine Learning Models The main evaluation of HVS uses *K-Nearest Neighbor Regressor* (KNN) as its offline model and *Linear Regressor* (LR) as its online model in order to match the models used in the state-of-the-art works ([30], [20]), which are used as benchmarks in our experiments. We implemented both models according to the descriptions and recommendations of their corresponding publications (i.e., [30] and [20]).

Additionally, we experimented with *Support Vector Regressor* and *Multi-layer Perceptron Regressor* as HVS's offline model, and *Multi-layer Perceptron Regressor* as HVS's online model, to verify the ability of HVS in accommodating different machine learning models.

Simulated Ideal Utility Function (IUF) In our experiments, we simulate ideal utility functions with four different individual utility measures (i.e., u_i's), which describe different characteristics of the views.

Diversity (D) measures the degree of differences in the data values. *Concentration (C)* ranks higher the views with similar data in values. *Outlier Score (O)* will 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 u_i's are shown in Table II. The 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.

According to Eq. 3, the simulated ideal utility function (i.e., $u^*{}_r$) is a composite function containing all the individual utility measures with corresponding weights derived from the data analysis log:

$$u^*{}_r() = \beta_1 \cdot D + \beta_2 \cdot C + \beta_3 \cdot O + \beta_4 \cdot U \qquad (4)$$

Table III shows the average number of distinct $u^*{}_r$ for each result r in the test data with respect to four different training set ratios (i.e., TR, ratio of training set size over dataset size). For each r there are more than one $u^*{}_r$, which means that the cybersecurity dataset is sufficient to test HVS's effectiveness, because using only the $u^p{}_r$ discovered in the offline stage will not be adequate for predicting the correct $u^*{}_r$ and refining utility function in the online stage will be necessary.

User Simulation We simulated the user feedback in the following way. For each example view v that is generated on

certain result r, the simulated user applies the ideal utility function ${u^*}_r$ on the individual utility measures of v to form a utility score as the user feedback.

B. Evaluation Metrics

We evaluated the performance of our solution with respect to recommendation precision (effectiveness) and efficiency.

Effectiveness: We define the precision as the size of the intersection between the top-k views recommended by ${u^p}_r$ and the top-k views recommended by ${u^*}_r$. For the two sets of top-k views V^p and V^* produced by ${u^p}_r$and ${u^*}_r$, respectively, the precision is calculated as:

$$\frac{|V^p \cap V^*|}{k} \tag{5}$$

where $\cap$ is the intersection operator and $|\cdot|$ is the cardinality operator.

Efficiency: Three components are included in the runtime comparison. The first part is the model training time in the offline stage, which applies to both the Offline model and HVS. The second part is the system response time in the online stage, which applies to the HVS and the Online model. The last part is the user inspection time, which applies to the HVS and the Online model. We adopt the method introduced in [27] for the calculation of the user inspection time.

Experiments: 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. Since interactive data analysis is a data exploration problem, we assume that the training data will only constitute a small portion of the total space of all possible exploration behaviors. Therefore, in our experiment, we set the training set size ratio to be 10%, 20%, 30%, and 40% of the dataset size and used the remaining data for testing.

V. Experimental Results

We compare HVS using KNN/LR Offline/Online models with both state-of-the-arts *online* model [30] and *offline* model [20], as stated above. The offline model [20] directly uses the weight initialization from the offline stage to make view recommendation, whereas the online model [30] interacts with the user without having the weight initialization from the offline stage.

Offline vs. HVS The comparison between Offline and HVS is directly based on recommendation precision. The precision for HVS is measured after the model learning has converged.

The comparison results are shown in Figures 7-10. The Offline model achieves an average precision of 63%, 66%, 67% and 69% across all top-k's for training set ratio of 0.1 - 0.4 respectively, while the HVS achieves 100% precision for all training ratios. From Figures 11-14, we can see that the HVS achieves the 100% precision by acquiring a very small amount of additional user labels. The above observation indicates that ${u^p}_r$ from the offline stage cannot capture the interest of the current user well, and an online interaction stage with marginal user effort can improve the recommendation precision significantly.

TABLE IV: Labeled View Count (Different Train Set Ratio)

Model	TR = 0.1	TR = 0.2	TR = 0.3	TR = 0.4
Online	4.8	4.8	4.8	4.8
HVS	3.1	3.0	3.2	2.8

TABLE V: Labeled View Count (Different top k)

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

Online vs. HVS The comparison between Online and HVS is based on the number of labeled views required by both systems to reach a recommendation precision of 100%.

The comparison results are shown in Figures 11-14. Across all top-k's, on average, the HVS requires only 64%, 62%, 66%, 58% labeled views compared to the Online model to achieve a 100% precision for training set ratio of 0.1 - 0.4 respectively. This observation shows that using the ${u^p}_r$ from the offline stage for model initialization in the online stage can significantly reduce the user labeling effort in the online stage. The actual number of labeled views for the two models are shown in Table IV and V.

Another interesting observation is that the training set ratio also slightly affects the recommendation precision of the models. The precision of the Offline model increases with increasing the training set ratio. This is quite understandable. With more training data, the estimated ${u^p}_r$ from the offline stage will better represent common ${u^*}_r$ from different users. Thus the ${u^p}_r$ will be more likely to capture the ${u^*}_r$ of the current user.

Similarly, the precision of the HVS also increases with increasing the training set ratio (i.e., the model requires fewer user labels to reach a 100% precision). The same reason as above goes here. Since with increased training set ratio, the ${u^p}_r$ will be more likely to capture the ${u^*}_r$ of the current user. It is natural that the user effort required in the subsequent online stage will be reduced.

Figures 15-18 show the runtime comparison between the Online model and HVS. We show only the Online model and HVS because we are more interested in the runtime of the online stage, which is more crucial for interactive data exploration. Note also that the time taken to perceive a view differs based on the complexity of the view and the proficiency of the user, for illustration purposed we based this experiment on the assumption that user's inspection of each view takes around 9 seconds [27]. Accordingly, on average, 16.8 seconds can be saved for each view by using HVS. In the scenario of data exploration, when a user submits a large number of queries during the exploration, the saving in runtime by HVS will become very beneficial.

Other Machine Learning Models As previously mentioned, in addition to KNN/LR Offline/Online models, we also evaluated HVS with two other Offline models and one other Online model to verify the ability of HVS in accommodating different machine learning models. Experiments for HVS with all six combinations of Offline/Online models show similar trends, such that HVS improved accuracy compared against

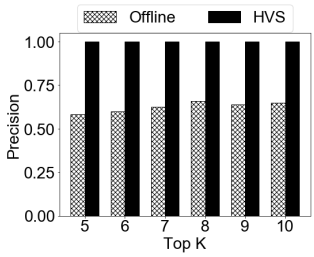

Fig. 7: Accuracy of Offline and HVS (10% train set ratio).

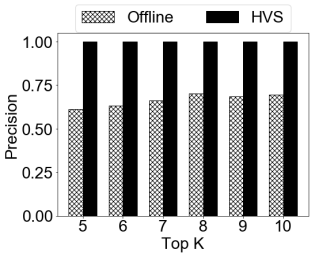

Fig. 8: Accuracy of Offline and HVS (20% train set ratio).

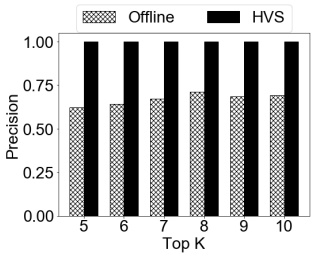

Fig. 9: Accuracy of Offline and HVS (30% train set ratio).

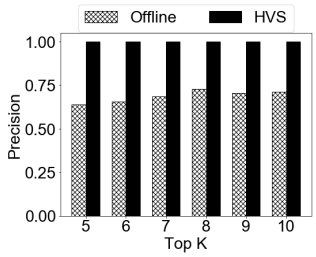

Fig. 10: Accuracy of Offline and HVS (40% train set ratio).

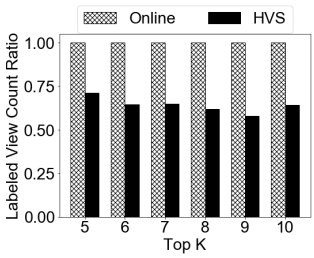

Fig. 11: Accuracy of Online and HVS (10% train set ratio).

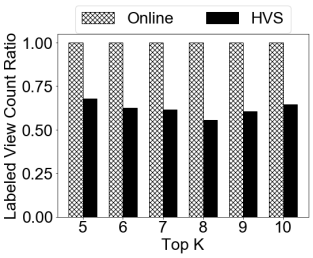

Fig. 12: Accuracy of Online and HVS (20% train set ratio).

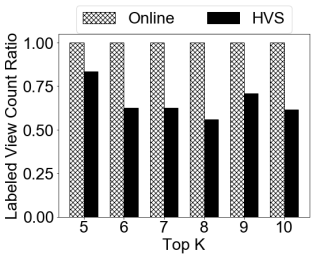

Fig. 13: Accuracy of Online and HVS (30% train set ratio).

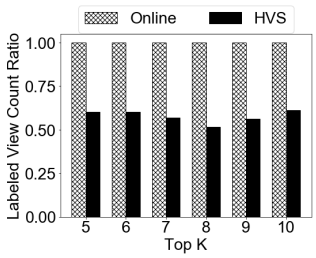

Fig. 14: Accuracy of Online and HVS (40% train set ratio).

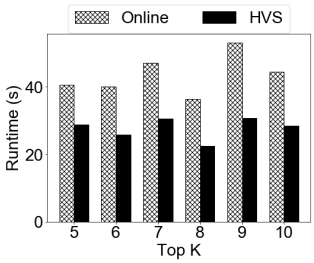

Fig. 15: Runtime of Online and HVS (10% train set ratio).

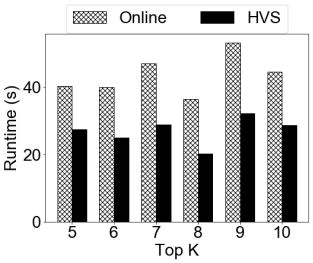

Fig. 16: Runtime of Online and HVS (20% train set ratio).

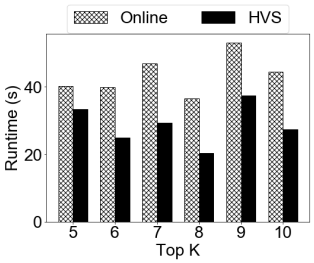

Fig. 17: Runtime of Online and HVS (30% train set ratio).

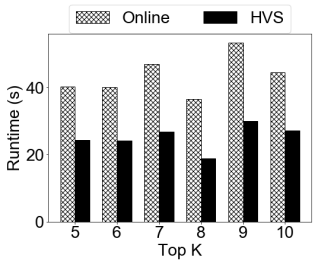

Fig. 18: Runtime of Online and HVS (40% train set ratio).

Offline and reduced user effort against Online. For a training set ratio of 0.3, the average precision ratio across all models for Offline versus HVS is 67% with a standard deviation of 4%, the average labeled view ratio of HVS versus Online is 58% with a standard deviation of 7%, and the average runtime saving of HVS versus Online is 23.3 seconds per view, with a standard deviation of 7.1 seconds.

VI. Related Works

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

Utility Discovery are the most relevant works to us [30], [20], [21]. In [30], the author has proposed an interactive way to perform view recommendation called ViewSeeker, which essentially is the online learning approach that we are referring to in this work. Specifically, ViewSeeker interacts with the user by presenting them with a set of carefully selected example views and obtains user feedback on the examples. Learning from the collected feedback, ViewSeeker is able to identify the ideal utility function that is most suited to a given user and exploration task. In [20], the author has proposed an offline learning solution that recommends an appropriate utility function during each exploration by learning from a large collection of interactive data analysis logs. In [21], a similar offline model was created using deep reinforcement learning. HVS is more advanced than the above works, since it is the first one to attempt to combine the offline learning from data analysis logs and online learning from user interaction to provide a personalized recommendation with minimized user effort.

View Recommendation is another relevant area to ours (e.g., [28], [5], [12], [19], [17]). A key difference among these works is the proposed utility functions. Recent work placed additional constraints, e.g., upper bound on the number of views to be explored and execution time limit when computing the recommended views [12]. The closest 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 [17]. 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, including the one using generic priors, is that all previous work employed predefined utility functions and did not discover the most appropriate utility function that matched an individual user's intention and exploration task.

Interactive Visualization Tools have been studied extensively for the past few years [13], [18], [26], [10], [14], [7], [16]. 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 specify the views to be generated manually. 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 [13]. But, unlike our approach, Profiler is not capable of providing general recommendations for any group-by queries. Another recent example is VizDeck [14], which generates all possible 2D views for a given dataset and allows the user to control manually (e.g., reordering, pinning) these views through a dashboard, rather than using a utility function.

Data Exploration techniques that aim to extract knowledge from data [23] efficiently are complementary to our work. In particular, *example-driven* data exploration approaches [9], [4] assume that the user has minimum prior knowledge of the data and iteratively refine the exploratory query through user interaction by acquiring user feedback on example data records. Our work in this paper is complementary to data exploration techniques and can enhance them by creating visualizations that illustrate interesting patterns during the construction of the exploratory queries.

VII. Conclusion

In this work, we proposed HolisticViewSeeker (HVS) a hybrid solution for efficient, personalized view recommendation for ad-hoc data analysis tasks. HVS combines the benefit of both online and offline learning approaches to provide high-quality view recommendation while minimizing interactive user effort. Our contribution is a novel, interactive view recommendation solution to the fundamental challenges of providing effective results with minimum user effort while making no assumption on the amount of the prior knowledge that the users have.

The crux of HVS is to leverage the offline learning approach to boot the online interactive discovery of the utility function, which is used to select the views that best match the user's exploration task. HVS effectively learns the user's preferred views through simple interactions between the user and the system, while leveraging the IDA session logs to significantly reduce the amount of human effort required for finding user's ideal utility function.

Our experimental results on real-world datasets show that HVS significantly outperforms the state-of-the-art baseline approaches by reducing the human effort up to 42%. Consequently, such a reduction in user effort can lead to up to 72% improvements in the productivity of data analysis task given a fixed number of working hours.

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments.

References

[1] NBA. - http://www.basketball-reference.com.
[2] Tableau public - http://public.tableau.com.
[3] G. E. Box, D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.
[4] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *ACM SIGMOD*, 2014.
[5] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. Muve: Efficient multi-objective view recommendation for visual data exploration. In *IEEE ICDE*, 2016.
[6] H. Ehsan, M. A. Sharaf, P. K. Chrysanthis. Efficient recommendation of aggregate data visualizations. *IEEE TKDE*, 30(2):263–277, 2018.
[7] D. Fisher. Hotmap: Looking at geographic attention. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1184–1191, 2007.
[8] R. Garnett, Y. Krishnamurthy, X. Xiong, J. Schneider, and R. Mann. Bayesian optimal active search and surveying. *arXiv:1206.6406*, 2012.
[9] X. Ge, Y. Xue, Z. Luo, M. A. Sharaf, and P. K. Chrysanthis. Request: A scalable framework for interactive construction of exploratory queries. In *IEEE BigData*, 2016.
[10] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *ACM SIGMOD*, 2010.
[11] R. J. Hilderman and H. J. Hamilton. Knowledge discovery and measures of interest. Vol. 638. Springer Science and Business Media, 2013.
[12] I. A. Ibrahim, A. M. Albarrak, and X. Li. Constrained recommendations for query visualizations. *Knowl. Inf. Syst.*, 51(2):499–529, 2017.
[13] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: integrated statistical analysis and visualization for data quality assessment. In *ACM AVI*, 2012.
[14] A. Key, B. Howe, D. Perry, and C. R. Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *ACM SIGMOD*, 2012.
[15] S. Lin and D. E. Brown. An outlier-based data association method for linking criminal incidents. *Decision Support Systems*, 41(3):604–615, 2006.
[16] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. Devise: Integrated querying and visual exploration of large datasets. In *ACM SIGMOD*, 1997.
[17] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *IEEE ICDE*, 2018.
[18] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE TVCG*, 13(6):1137–1144, 2007.
[19] R. Mafrur, M. A. Sharaf, and H. A. Khan. Dive: Diversifying view recommendation for visual data exploration. In *ACM CIKM*, 2018.
[20] T. Milo, C. Ozeri, and A. Somech. Predicting what is interesting by mining interactive-data-analysis session logs. In *EDBT*, 2019.
[21] T. Milo and A. Somech. Deep reinforcement-learning framework for exploratory data analysis. In *Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2018.
[22] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *ACM SIGKDD*, 2018.
[23] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. New trends on exploratory methods for data analytics. In *PVLDB*, 10(12):1977–1980, 2017.
[24] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.
[25] E. H. Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.
[26] C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. In *INFOVIS*, 2000.
[27] J. Talbot, V. Setlur, and A. Anand. Four experiments on the perception of bar charts. *IEEE TVCG*, 20(12):2152–2160, 2014.
[28] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.
[29] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE TVCG*, (1):1–1, 2016.
[30] X. Zhang, X. Ge, P. K. Chrysanthis, and M. A. Sharaf. Viewseeker: An interactive view recommendation tool. In *BigVis*, 2019.