SemDiff: Explaining Semantic Changes in Evolving Databases

Shiyi He
University of Utah
Salt Lake City, Utah, USA
shiyi.he@utah.edu

Anna Fariha
University of Utah
Salt Lake City, Utah, USA
afariha@cs.utah.edu

ABSTRACT

Due to the proliferation of machine learning techniques and the availability of data, we now heavily rely on data-driven decision making. Data represents real-world entities; these entities change, so does the corresponding data. To make sense of the change patterns and gain insights, we need to analyze these data changes. E.g., gender-based reward disparity—discovered by comparing historical versions of employee salary data—may uncover gender biases in performance rewards. However, existing data change exploration techniques list all changes exhaustively (e.g., 5000 cells have changed values), which overwhelms human analysts, preventing them from discovering the salient insights regarding change trends.

Motivated by these challenges, we present SemDiff, a system for explaining *semantic* differences between two snapshots of an evolving database, in an effective, concise, and interpretable way. Our key observation is that changes in databases are typically systematic, resulting from a series of batch updates over different data partitions. We leverage this observation to generate a "change explanation"—a set of transformations that can be applied to an earlier version of the database to obtain the later version—that ensures that (1) the explanation accurately captures the most salient changes in the data and (2) humans can reasonably interpret the explanation. Under the hood, SemDiff compares database versions, infers feasible transformations by fitting multiple regression lines over different partitions of the data, and introduces a ranking function to rank the explanations balancing the trade-offs between accuracy and interpretability. Through an intuitive and interactive interface, SemDiff enables the user to customize the system to obtain their preferred explanation. We demonstrate how SemDiff can help users effectively reason about data evolution over real-world datasets.

1 INTRODUCTION

The task of data understanding is of prime importance in today's data-driven world. To make sense of data, existing data summarization systems [5, 7] enable users to interactively understand the content of a static database. However, just like any living organism, data evolves, which demands a mechanism to understand data evolution effectively and efficiently. One of the most important change dimensions is *temporal* change, which captures how a database changes over time. We aim to explain how two temporal snapshots of a database *semantically* differ from one another.

Whenever there is a change, the next accompanying questions are regarding the mechanism, quantification, and cause of the change. When something changes, the follow-up questions are: *how* (mechanism), *how much* (quantification), and *why* (cause). Change logs are often unavailable or inaccessible by the end users. Even if available, change logs may not be in a format for easy human consumption, and, thus, interpreting such large change logs to answer the above questions poses a significant hurdle for data consumers.

name	gen	edu	exp	salary	bonus
Anne	F	PhD	2	$230,000	$23,000
Bob	M	PhD	3	$250,000	$25,000
Amber	F	MS	5	$160,000	$16,000
Allen	M	MS	1	$130,000	$13,000
Cathy	F	BS	2	$110,000	$11,000
Tom	M	MS	4	$150,000	$15,000
James	M	BS	3	$120,000	$12,000
Lucy	F	MS	4	$150,000	$15,000
Frank	M	PhD	1	$210,000	$21,000

(a)

name	gen	edu	exp	salary	bonus
Anne	F	PhD	3	$230,000	$25,150
Bob	M	PhD	4	$250,000	$27,250
Amber	F	MS	6	$160,000	$17,440
Allen	M	MS	2	$130,000	$13,790
Cathy	F	BS	3	$110,000	$11,000
Tom	M	MS	5	$150,000	$16,400
James	M	BS	4	$120,000	$12,000
Lucy	F	MS	5	$150,000	$16,400
Frank	M	PhD	2	$210,000	$23,050

(b)

Figure 1: Two versions of a dataset about employee salaries: (a) 2016, and (b) 2017. The two datasets differ in the attribute bonus. When they differ (highlighted in yellow), their differences range from 8% to 10%. We are interested in understanding the change trend better.

Data versioning techniques [4] can partially address the problem of tracing data change, i.e., the locations and quantities of the changes. However, they fail to reveal the underlying reasons and mechanisms driving the changes. Merely knowing which cells changed and their change quantities is inadequate for users to understand the reasons behind the changes.

Example 1. *Figure 1 presents two snapshots of a salary database in (a) 2016 and (b) 2017. In 2016,* `bonus` *was a flat 10% of* `salary` *for all employees. In contrast, we observed no such straightforward trend in 2017. In some cases, the value of* `bonus` *differs from last year's value (highlighted in yellow), while in some cases they are identical (for Cathy and James). Furthermore, the difference ranges from 8% to 10% and is not identical for everyone. Simply knowing that* `bonus` *changed from last year leaves one unsatisfied, as it is not obvious what is the underlying trend behind such non-uniform changes.*

Turns out that the company opted for a policy to reward long-serving employees and promote educational advancement. In 2017, the company decided to depart from a flat-rate bonus to a customized one. The new scheme for bonus calculation is influenced by three principles: (1) no one should receive bonus any less than last year's bonus. (2) employees with higher level of education should be rewarded more, and (3) long-serving employees should be rewarded more.

This is not immediately apparent by just looking at the data, since `bonus` *for 2017 is no longer directly tied to* `salary`*, as was the case in 2016. Instead, it is calculated based on a combination of last year's* `bonus`*, employee's education (*`edu`*), and years of experience (*`exp`*). Specifically, the following rules accurately explain the change trend:*

- ***R1****: Employees who have a PhD receive a 5% increase on last year's bonus, plus flat $1000.*
- ***R2****: Employees who have an MS and served for at least 3 years receive a 4% increase on last year's bonus, plus flat $800.*
- ***R3****: Employees who have an MS and served for less than 3 years receive a 3% increase on last year's bonus, plus flat $400.*

There are two desirable properties for a "change explanation": (1) it should *precise*, i.e., be able to explain the changes *accurately*,

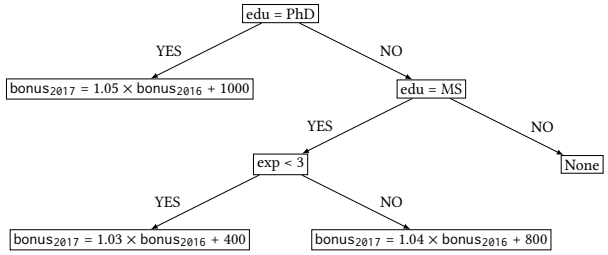

Figure 2: A regression tree explaining diff in datasets in Figure 1.

and (2) it should be *interpretable* and *succinct* for easy human consumption. Note that there is a natural tension between these two desirable properties. Consider the following change explanation:

- ***R4**: Everyone receives about 6% increase on last year's bonus.*

R4 is more interpretable (as it is more succinct and easier to consume by humans) than the explanation {**R1**, **R2**, **R3**}; however, **R4** poorly captures the trend of change and is less accurate than {**R1**, **R2**, **R3**}. In contrast, one can provide a change explanation by listing each individual cell that changed. However, such an explanation—despite being very precise—would lack interpretability as it will overwhelm the user with too much details.

SemDiff: To meet the requirements of accuracy and interpretability, we developed SemDiff, a system for producing *semantic explanation of changes* between two temporal snapshots of a database while striking the right balance between accuracy and interpretability. Our key observation is that changes in databases are typically systematic, and are based on some underlying principled mechanisms. Our goal is to "reverse engineer" those underlying mechanisms by observing the data. More specifically, we assume that, given a *source* dataset (earlier version) and a *target* dataset (later version), the latter is obtained via a set of systematic change operations over the former. Our goal is to first reverse engineer and subsequently distill those change operations into an interpretable format.

The core challenges in finding the best change explanation are to (1) find the right data partitions such that (2) each partition conforms to a uniform "transformation" of reasonable complexity. To solve the first challenge, SemDiff uses k-means clustering to guide the search for data partitions based on certain data attributes (e.g., education and year of experience). Once approximate partitions are discovered, SemDiff applies linear regression to find the most suitable transformations to capture the changes within each partition (e.g., $\text{bonus}_{2017} = 1.05 \times \text{bonus}_{2016} + 1000$). Due to the simplistic nature of linear regression, such transformations have lower complexity in terms of data attributes. The output of SemDiff is a regression tree (Figure 2), which is succinct and interpretable.

Furthermore, SemDiff enhances user experience by (1) *customization*—users can specify system parameters such as the maximum number of attributes they want to see in the change explanation—and (2) *visualization*—they can interactively inspect different partitions of the data and the corresponding change trends. In our demonstration, participants will witness how SemDiff generates explanations from two datasets tailored to user preferences, and enables them to effectively gain insights about data changes.

Related work. Prior work [3] have studied the problem of exploring the entire history of changes in a database, but are limited to syntactic or raw changes, suitable for historical change exploration involving a particular entity. Database comparator tools such as PostgresCompare[1], RDBMS version control system OrpheusDB [4] etc. only look for syntactic changes—values are changed, objects are altered, rows are removed or added—which is not concise enough to provide an overall insight of change. Data-diff [9] explores change in distributions of datasets, specialized in the context of data wrangling. Muller et al. [6] describe change in two datasets in terms of "update distance", defined by the minimal number of insert, delete, and modification operations necessary. However, none of these works focus on summarizing changes between two databases.

Explain-Da-V [8] is closest to SemDiff as they also explain transformations that convert a source dataset to a target dataset. However, they focus on semantics of schema and data format transformations (e.g., data extraction, row deletion, adding attributes representing length of an attribute). In contrast, we focus on semantic changes where values of attributes may change based on interactions among other attributes, with the goal of explaining the evolution of real world entities represented by datasets.

We proceed to describe our solution sketch in Section 2 and then provide the demonstration outline in Section 3.

2 SOLUTION SKETCH

Given a source dataset $\mathcal{D}_s$, a target dataset $\mathcal{D}_t$, and an attribute of interest (a_i), we aim to identify a list of *explanations* that can capture the changes observed between $\mathcal{D}_s(a_i)$ and $\mathcal{D}_t(a_i)$. Each explanation consists of a set of data transformations over different data partitions. We assign each explanation a score that captures how well it strikes the balance between accuracy and interpretability, and then rank all explanations based on their scores. We focus on data modification and support linear transformations over numerical attributes.

Explanation and conditional transformation. Our unit to explain changes is *conditional transformation* (CT), which comprises a *condition* and a *transformation*. An explanation $E = \{CT_1, CT_2, \dots\}$ comprises a set of CTs. The condition explains why a change happened, and the transformation describes the change itself. For instance, the following CT explains that the salary of all employees with a PhD was increased by 5% and then 1000 was added.

$$\underbrace{\text{edu = PhD}}_{\text{Condition}} \longrightarrow \underbrace{\text{new\_bonus = 1.05 } \times \text{ old\_bonus + 1000}}_{\text{Transformation}}$$

Desiderata for change explanation. A desirable change explanation must ensure that (1) all or most of the data changes are sufficiently explained and (2) the explanation itself is interpretable and succinct for human consumption. To this end, we introduce $Score(E) \in [0, 1]$ for an explanation E, which indicates how well E can explain the differences between $\mathcal{D}_s(a_i)$ and $\mathcal{D}_t(a_i)$.

$$Score(E) = \alpha \times Accuracy(E) + (1 - \alpha) \times Interpretability(E)$$

Here, *Accuracy* is measured using the inverse of the L_1 distance between $\hat{\mathcal{D}}_s(a_i)$ and $\mathcal{D}_t(a_i)$, where we want to explain the changes observed in attribute a_i, and $\hat{\mathcal{D}}_s$ is the transformed database obtained by applying the CTs in E on $\mathcal{D}_s$.

[1] PostgresCompare: www.postgrescompare.com/

The second term is *Interpretability*. We use several intuitions to model interpretability:

- *Prefer smaller explanation.* An explanation with fewer CTs is preferable, as it leads to increased conciseness.
- *Prefer smaller description for conditions and shorter equation for transformations.* For example, the transformation "All Female employees received 5% bonus" is more interpretable than "All Asian, European Females, or Females working in HR received a 5% bonus". We also prefer a transformation with fewer components.
- *Prefer conditions with higher data coverage.* A condition that targets a small partition of the data reveals very little information. Thus, we prefer conditions that have higher coverage, i.e., select a larger data partition.
- *Prefer simpler conditions and transformations.* Conditions and transformations may involve numeric constants. We prefer the ones involving more "normal" values. For instance, the condition "Age > 25" has a higher normality than "Age > 23.796". Similarly, the value 5% for a salary increase is more "normal" (and interpretable) than 2.479%. We rely on domain expertise to learn such notions of normality.

Enhancing interpretability while not compromising accuracy too much positively impacts the effectiveness of an explanation, thereby yielding a higher overall score. To control the interplay between accuracy and interpretability, we use a customizable parameter α with the default value set to 0.5.

SemDiff architecture. Figure 3 shows the SemDiff architecture. SemDiff consists of two components: the *setup assistant*, which helps the users choose system parameters and the *diff discovery engine*, which is responsible for reporting the change explanations.

Setup assistant. For datasets with a large number of attributes, the search space for possible explanations can explode. Ideally, one can seek user guidance to reduce the set of attributes to search over. However, this can overwhelm the user. On the other hand, not involving the user at all in such a crucial step may result in explanations involving irrelevant attributes. To resolve this issue, SemDiff follows a hybrid approach: it presents to the user a shortlist of attributes that are most likely to be effective for explaining the change in the target attribute. To this end, SemDiff estimates influence of other attributes on the target attribute using correlation analysis and set cover. Based on the estimates, SemDiff presents candidate sets of attributes for both condition ($\mathcal{A}_{cond}$) and transformation ($\mathcal{A}_{tran}$), which the user can narrow down further.

Diff discovery engine. The goal of the diff discovery engine is to find a list of change explanations $EL = \{(E_1, S_1), (E_2, S_2), \ldots\}$, ordered by decreasing order of their scores ($S_1 \geq S_2 \geq \ldots$). For each explanation E_i, SemDiff discovers different data partitions specified by conditions (defined by a subset of attributes in $\mathcal{A}_{cond}$), where each partition conforms to a specific transformation (defined by a subset of attributes in $\mathcal{A}_{tran}$). SemDiff required two parameters, k and c, where c defines the maximum number of attributes $\in \mathcal{A}_{cond}$ to be used for partition discovery, and t denotes the maximum number of numerical attributes $\in \mathcal{A}_{tran}$ used to fit the linear regression model within each partition.

Based on the attribute sets $\mathcal{A}_{cond}$, $\mathcal{A}_{tran}$, and the parameters c and t, SemDiff enumerates all possible combinations of attributes

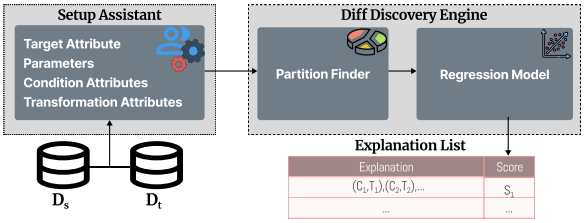

Figure 3: SemDiff has two main components: (1) a setup assistant to help users choose system parameters such as target attribute and attributes to consider for conditions and transformations, and (2) a diff discovery engine that discovers the change explanation based on data partitioning and fitting regression lines.

to use for partitioning and generating transformations. For example, for the parameters: $c = 3$ and $t = 2$, SemDiff will choose all subsets of $\mathcal{A}_{cond}$ having cardinality ≤ 3 as partitioning attributes and all subsets of $\mathcal{A}_{tran}$ having cardinality ≤ 2 as regression attributes.

Partition discovery. For a specific set of transformation attributes $\mathcal{T} \subseteq \mathcal{A}_{tran}$ and condition attributes $\mathcal{C} \subseteq \mathcal{A}_{cond}$ SemDiff first fits a linear regression model, over the entire data, for the target attribute (a_i) based on the attributes in $\mathcal{T}$. This guides SemDiff to perform K-means clustering along the regression line to discover potentially meaningful partitions in terms of the attributes in $\mathcal{C}$.

Transformation discovery. Once partitions are discovered, for each partition, SemDiff again fits a regression model based on the transformation attributes in $\mathcal{T}$ to generate a transformation. All such transformations over different partitions, together, result in a change explanation, which can be modeled using a regression tree similar to the one shown in Figure 2. Once all explanations $\{E_1, E_2, \ldots\}$ are generated within the specified parameters, SemDiff computes *Score* of each explanation, ranks the generated explanations according to the descending order of their respective scores, and returns the ranked list of explanations $\{(E_1, S_1), (E_2, S_2), \ldots\}$.

3 DEMONSTRATION

We will demonstrate SemDiff on a real-world dataset representing salary information for all active, permanent employees of Montgomery County, MD for the years of 2020 [1] and 2021 [2]. The dataset contains information about employee salaries over 8 attributes including Department, Department Name, Division, Gender, Base Salary, Overtime Pay, Longevity Pay, and Grade.

Figure 4 shows a screenshot of SemDiff's graphical user interface. During the demonstration, we will guide the participants through ten steps. We have annotated each step with a circle.

Step ① (Uploading datasets) The user uploads two datasets they want to compare. For reader's convenience, we will use the toy datasets used in Example 1 throughout this demo scenario.

Step ② (Selecting the target attribute) Next, the user chooses the target attribute whose changes they want an explanation for. For our scenario, the user chooses "bonus".

Step ③ (Setting parameters) Next, the user chooses the maximum number of condition attributes to use for partitioning (3) and the maximum number of transformation attributes (2).

Steps ④ & ⑤ (Attribute selection) SemDiff presents a ranked list of attributes it deems most promising to be chosen as condition attributes ④ and transformation attributes ⑤. By default,

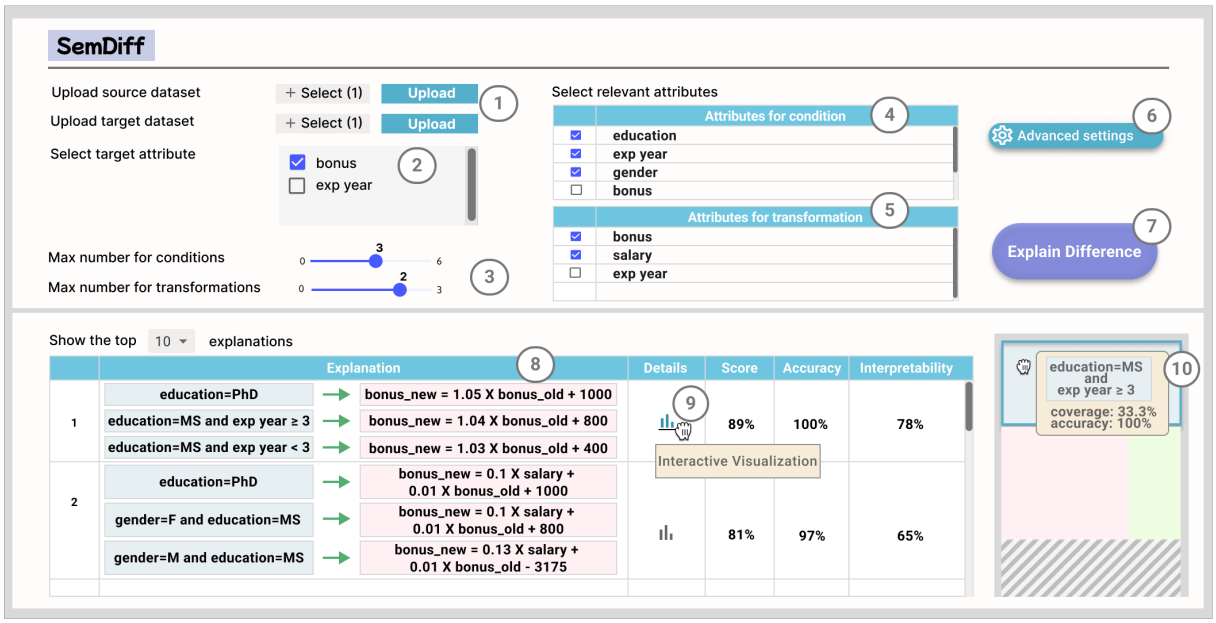

Figure 4: The SemDiff demo: (1) upload datasets, (2) select the target attribute, (3) specify the maximum number of attributes for condition and transformation, (4) SemDiff presents candidate attributes for condition automatically, users can customize (5) SemDiff presents candidate attributes for transformation automatically, users can customize (6) tune advanced system parameters to meet specific needs, (7) request change explanations from SemDiff, (8) SemDiff presents a list of ranked explanations, with their overall score, and scores for accuracy and interpretability, (9) click on an explanation for more details, (10) data partitions are visualized with more details.

SemDiff selects the top three results from each list. Users have the option to either accept this default setting or interactively filter out undesired attributes or select some other attributes. In our case, the user selects "education", "exp year", and "gender" as potential condition attributes and "bonus" (of the previous year) and "salary" as potential transformation attributes.

Step (6) (Advanced settings) If the user wishes, they can configure advanced system parameters. For example, the default value of α is set to 0.5. However, users have the discretion to either accept this default setting or adjust the parameters to meet their specific requirements. For example, if they seek a more interpretable explanation, they can tune α to a lower value, thereby, prioritizing interpretability over accuracy.

Steps (7) & (8) (Change explanations) The user requests to generate change explanation (7) and SemDiff displays the explanations (8). Each explanation comprises a set of conditional transformations—where conditions are in light blue and transformations are in light pink—followed by an option to visualize (10), overall score (linear combination of accuracy and interpretability), accuracy, and interpretability scores. In this scenario, the first explanation produced by SemDiff reflects the scenario described in Example 1, which incurs a very high score of 89%. By default, SemDiff presents top 10 explanations based on their scores. If fewer than 10 explanations are available, SemDiff displays all.

Steps (9) & (10) (Visualization) To better understand an explanation, the user requests for more details (9). SemDiff offers an interactive visualization (10) comprising several non-overlapping rectangles, each representing a data partition achieved via applying the conditions. The size of each rectangle corresponds to its data coverage. E.g., 33.3% employees fall within the top partition. For each partition, additional details—such as partitioning condition, data coverage, accuracy of the transformation—are revealed when the user hovers over these rectangles. The bottom partition, marked by diagonal patterns, indicates that no change was observed there.

Demonstration engagement. After our guided demonstration, participants will be able to plug their own datasets into SemDiff. Through the demonstration, we will showcase how SemDiff can semantically explain changes between two datasets.

Target audience and implementation status. Our target users are data analysts, decision makers, and data enthusiasts who want to understand data change trends. We have implemented the backend of SemDiff and the frontend (UI) development is under progress.

REFERENCES

[1] 2020. Employee Salaries 2020 Dataset. https://data.montgomerycountymd.gov/Human-Resources/Employee-Salaries-2020/he7s-ebwb/about_data.
[2] 2021. Employee Salaries 2021 Dataset. https://data.montgomerycountymd.gov/Human-Resources/Employee-Salaries-2021/kmkb-bmhe/about_data.
[3] T. Bleifuß, L. Bornemann, T. Johnson, D. Kalashnikov, F. Naumann, and D. Srivastava. 2018. Exploring Change - A New Dimension of Data Analytics. *PVLDB* (2018).
[4] S. Huang, L. Xu, J. Liu, A. Elmore, and A. Parameswaran. 2017. OrpheusDB: Bolt-on Versioning for Relational Databases. *PVLDB* (2017).
[5] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. 2019. Interactive Data Exploration with Smart Drill-Down. *IEEE* (2019).
[6] H Müller, J C Freytag, and U Leser. 2006. Describing differences between databases. In *CIKM*.
[7] S. Sarawagi. 2001. User-cognizant multidimensional analysis. *PVLDB* (2001).
[8] R. Shraga and R. Miller. 2023. Explaining Dataset Changes for Semantic Data Versioning with Explain-Da-V. *PVLDB* (2023).
[9] C. Sutton, T. Hobson, J. Geddes, and R. Caruana. 2018. Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling. In *SIGKDD*.