SubTab: Data Exploration with Informative Sub-Tables

Kathy Razmadze
Tel Aviv University

Yael Amsterdamer
Bar-Ilan University

Amit Somech
Bar-Ilan University

Susan B. Davidson
University of Pennsylvania

Tova Milo
Tel Aviv University

Abstract

We demonstrate SubTab, a framework for creating small, informative sub-tables of large data tables to speed up data exploration. Given a table with n rows and m columns where n and m are large, SubTab creates a sub-table T_{sub} with $k << n$ rows and $l << m$ columns, i.e. a subset of k rows of the table projected over a subset of l columns. The rows and columns are chosen as representatives of prominent data patterns within and across columns in the input table. SubTab can also be used for query results, enabling the user to quickly understand the results and determine subsequent queries.

1 Introduction

Data exploration is an important first step in data analytics. During this step, the analyst tries to understand an unfamiliar dataset and determine what part of the data is relevant to their task by displaying the table, looking at the table description, or visualizing column values. However, when displaying a large table only a small subset of the table is typically shown – and without input from the user, the subset is arbitrary. For example, the default display of Pandas[1] tables using the Python display() command includes the first and last several rows and columns. Frequently, this is not very informative as the sub-table may contain a lot of missing values and/or fail to capture the range of possible values in a column; it may also elide columns that are important for further exploration.

Example 1.1. *Consider a table T taken from the Kaggle* flights *dataset*[2] *which contains 31 columns and ∼6M rows. The analyst is using T to predict flight cancellations, and hence is interested in a specific target column, CANCELLED. The analyst starts by visually inspecting the data using Pandas* display(T)*, which yields the table displayed at the top of Figure 1. This display of T is not informative for the analysis task, as it does not include the target column. More crucially, its usefulness for data exploration is limited: e.g., the last five columns*

[1] Pandas: Python Data Analysis Library. https://pandas.pydata.org/
[2] https://www.kaggle.com/usdot/flight-delays?select=flights.csv

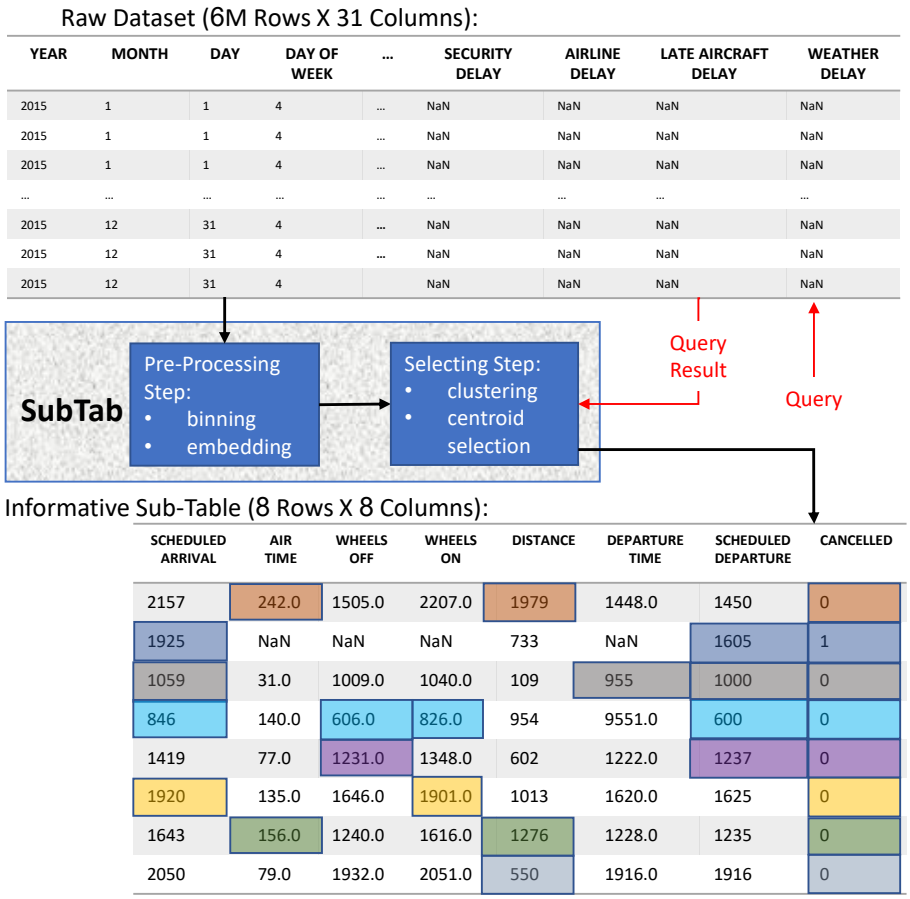


Figure 1: System Architecture

contain only NaN *values, and other columns include many repetitions of arbitrary values.* □

The goal of SubTab is to support the data exploration task by selecting small, informative sub-tables through which analysts can view data. Given a table T with n rows and m columns, SubTab creates a sub-table with $k << n$ rows and $l << m$ columns, which is a subset of k rows projected over a subset of l columns of T.

Intuitively, a sub-table is *informative* if it captures data patterns within and across columns in T. We formalize this intuition based on a combination of two complementary metrics: coverage and diversity. *Coverage*, measures how well the sub-table covers patterns in the data, where we define patterns through the standard notion of *association rules*. For the second metric, *diversity*, we rely on the average pair-wise similarity between the rows of the sub-table.

If one or more *target columns* are known in advance to be the focus of the analysis, they must be included in the l selected columns, and we measure cell coverage only according to association rules that include the target columns.

Example 1.2. *Continuing with the* flights *dataset, the output of* SubTab *is shown at the bottom of Figure 1. This sub-table is informative in a number of ways. First, it represents diverse, prominent association rules that include the target column CANCELLED. We visualize this by highlighting, in each row, the cells that participate in a rule that holds for this row (many more rules hold, to avoid visual clutter we only highlight one per row). For example, the first row of the sub-table exemplifies an association rule stating that long flights (AIR_TIME* $\in [198.0, 422.0]$ *and DISTANCE* $\in [1546.0, 2724.0]$*) are likely* not *to be cancelled (highlighted in orange). The second row of the sub-table exemplifies an association rule stating that short afternoon flights (according to the SCHEDULED_DEPARTURE and SCHEDULED_ARRIVAL columns) are likely to be cancelled (CANCELLED* $= 1$*, highlighted in blue). Beyond these association rules, the sub-table gives useful insights about the column values, by showing diverse rows and diverse values per column.* □

Unfortunately, we can show that optimizing this informativeness score directly is NP-hard (see full paper in [9]). Moreover, even computing the association rules is not practical in our setting: while there are several efficient techniques for mining association rules (e.g., [10]), they are not fast enough for an interactive setting.

We therefore consider a sub-table computation method which indirectly accounts for association rules using *table embedding* [1, 2]: we compute a vector representation for table cells, rows and columns which is based on co-occurrences of values. Hence, the patterns captured by this representation implicitly correspond to association rules. To do this, given a table T we use *binning* [10] to split each column's values into a small set of meaningful groups. We then use Word2Vec [6] to learn an embedding for table cells in vectors, a technique which was shown to be useful (e.g., [1, 2]).

An important benefit of our solution design is in *responding to queries over* T: we compute the binning and cell embedding once upon loading the data table (*Pre-processing* step in Figure 1). Then, if the analyst issues a selection-projection (SP) query on T and wishes to view its result as a sub-table, we need only to compute the vector representation of rows and columns in $Q(T)$ based

on the cells of T that appear in them, and re-execute clustering and centroid selection (*Selecting* step in Figure 1). This significantly speeds up sub-table computation compared with computing everything from scratch (a few seconds instead of up to a minute for large tables). Note that our solution design is independent of the concrete choices of cell embedding, binning and clustering techniques. The specific components that we use allow for *fast response* in an interactive setting. Moreover, they work well in practice and apply to a variety of datasets, even when compared with more complex alternatives [9].

Demo We show how `SubTab` is used in data exploration sessions over several real-life datasets. Our `SubTab` implementation seamlessly integrates with Pandas, and replaces its default query results presentation, so users can keep using the same exploratory operations they are used to. Participants will use Jupyter notebooks[3] to explore a dataset of their choice, and observe the generated sub-table for each query they perform, see Section 3 for more details.

Related work Existing work primarily addresses the task of selecting *either* rows or columns. Representative *rows* selection has been studied mainly in the context of *results diversification* (e.g., [3]), and data visualizations (e.g., [8]). Choosing *columns* is done mainly for *feature selection* (e.g., [4]). *Database summarization* techniques (e.g., [7]) were suggested for *approximated query processing*, with the goal of constructing a small summary table over which queries are evaluated faster than on the original data. The summary augments and compresses the original data and need not preserve the original data representation. Our work differs by generating informative sub-tables, which contains the actual data from representative rows and columns in the original table.

2 Approach

In this section, we define our metrics of cell coverage and diversity, and the problem of finding an optimal sub-table based on their combination. We then briefly discuss the infeasibility of directly computing an optimal sub-table according to our metrics, followed by an efficient embedding based algorithm that can empirically be shown to perform well in practice.

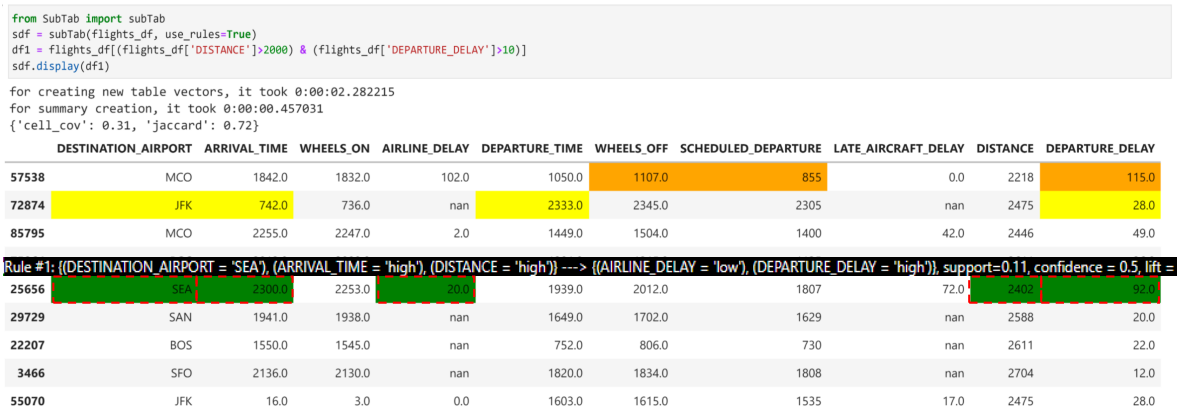
2.1 Informativeness Metrics

Cell coverage A table T is a set of tuples associated with a set of columns U, such that $t(u)$ is the cell of tuple $t \in T$ in column $u \in U$. An association rule R over T has the form

$$\{(u_1, v_1), \ldots (u_r, v_{r_R})\} \rightarrow \{(u_{r_R+1}, v_{r_R+1}), \ldots (u_{r_R+p_R}, v_{r_R+p_R})$$

where each $u_i \in U$ is an attribute and each v_i is a cell value [10]. Denote by U_R the set of attributes $\{u_1, \ldots, u_{r_R+p_R}\}$, which we assume w.l.o.g. to be distinct. Denote by $T_R \subseteq T$ the subset of tuples for which R holds, i.e., $t(u_i) = v_i$ for every $t \in T_R$ and $1 \leq i \leq r_R + p_R$.

[3]Project Jupyter. `https://jupyter.org`



	DESTINATION_AIRPORT	ARRIVAL_TIME	WHEELS_ON	AIRLINE_DELAY	DEPARTURE_TIME	WHEELS_OFF	SCHEDULED_DEPARTURE	LATE_AIRCRAFT_DELAY	DISTANCE	DEPARTURE_DELAY
57538	MCO	1842.0	1832.0	102.0	1050.0	1107.0	855	0.0	2218	115.0
72874	JFK	742.0	736.0	nan	2333.0	2345.0	2305	nan	2475	28.0
85795	MCO	2255.0	2247.0	2.0	1449.0	1504.0	1400	42.0	2446	49.0
25656	SEA	2300.0	2253.0	20.0	1939.0	2012.0	1807	72.0	2402	92.0
29729	SAN	1941.0	1938.0	nan	1649.0	1702.0	1629	nan	2588	20.0
22207	BOS	1550.0	1545.0	nan	752.0	806.0	730	nan	2611	22.0
3466	SFO	2136.0	2130.0	nan	1820.0	1834.0	1808	nan	2704	12.0
55070	JFK	16.0	3.0	0.0	1603.0	1615.0	1535	17.0	2475	28.0

Figure 2: Example usage of SubTab – presenting an informative, 10X10 sub-table for a large query results-set

We are given a sub-table T_{sub} whose tuples are a subset of T's tuples projected over a subset of the attributes $U_{\text{sub}} \subseteq U$, and a set $\mathcal{R}$ of prominent[4] association rules mined from T.To measure the *coverage* of T_{sub} with respect to T and $\mathcal{R}$ we consider the following.

(q1) Which of the rules of $\mathcal{R}$ are captured or *covered* by T_{sub}?

(q2) What is the marginal contribution of each covered rule to T_{sub}'s informativeness?

(q3) How do these contributions aggregate to an overall numerical score for T_{sub}?

Formal definitions for q1-q3 are given below, leading to a metric which captures the ratio of cells in T that are describable by association rules in $\mathcal{R}$ that are represented in T_{sub}.

Definition 2.1 (Cell coverage)**.** *Let T be a table, $\mathcal{R}$ a set of association rules mined from T, and T_{sub} a sub-table of T.*

(d1) A rule $R \in \mathcal{R}$ is said to be covered *by T_{sub} if $U_R \subseteq U_{sub}$, the attributes of T_{sub}, and $\{T_{sub}\}_R \neq \emptyset$, i.e., there exists a tuple $t \in T_{sub}$ for which R holds. Denote by $\mathcal{R}_{sub}$ the set of all rules in $\mathcal{R}$ that are covered by T_{sub}.*

(d2) The marginal contribution *of a rule $r \in \mathcal{R}$ is denoted by $cell(R,T) := \{t(u) \mid t \in T_R \wedge u \in U_R\}$, i.e., the subset of table cells it describes.*

(d3) The cell coverage *of T_{sub} with respect to $T, \mathcal{R}$ is denoted by*

$$cellCov_{\mathcal{R}}(T, T_{sub}) := \frac{1}{N} \left| \bigcup_{R \in \mathcal{R}_{sub}} cell(R,T) \right|$$

where the normalization factor $N := \left|\bigcup_{R\in\mathcal{R}} cell(R,T)\right|$ ensures that the value of $cellCov_{\mathcal{R}}(T, T_{sub})$ is in $[0,1]$.

[4]There are standard metrics we can use to measure the prominence of association rules in T, such as Support and Confidence [10].

Pre-processing ($\tilde{T}$) // Raw table

$T \leftarrow$ sample, normalize and bin $\tilde{T}$;

return T;

Embedding Computation (T) // Pre-processed table

$S \leftarrow$ rows and columns of T as text;

$\mathcal{M} \leftarrow \mathsf{Word2Vec}(S, \mathsf{windowSize} = \max\{n, m\})$;

return $\mathcal{M}$; // cell-to-vector model: $\mathcal{M} : T \times U \to \mathbb{R}^\gamma$

Centroid Selection (T, k, l, Q, U^*)

$rowVecs, colVecs \leftarrow$ empty dictionaries;

if $Q \neq \mathsf{NULL}$ **then** $T \leftarrow Q(T)$;

$U \leftarrow$ columns of T;

for $t \in T$ **do**

  $v \leftarrow avg_{u \in U}(\mathcal{M}(t(u)))$;

  $rowVecs \leftarrow rowVecs \cup \{v \mapsto t\}$;

$\mathcal{C} \leftarrow \mathsf{cluster}(rowVecs, k)$;

$T_{\text{sub}} \leftarrow rowVecs.\, getValues(\mathsf{centroids}(\mathcal{C}))$;

for $u \in U - U^*$ **do**

  $v \leftarrow avg_{t \in T}(\mathcal{M}(t(u)))$;

  $colVecs \leftarrow colVecs \cup \{v \mapsto u\}$;

$\mathcal{C} \leftarrow \mathsf{cluster}(colVecs, l - |U^*|)$;

$U_{\text{sub}} \leftarrow U^* \cup colVecs.\, getValues(\mathsf{centroids}(\mathcal{C}))$;

$T_{\text{sub}} \leftarrow \Pi_{U_{\text{sub}}} T_{\text{sub}}$;

return $T_{\text{sub}}, U_{\text{sub}}$;

Algorithm 1: Sub-table selection.

Diversity Sub-tables with high cell coverage may seem repetitive to humans: values that do not participate in rules are ignored, and the association rules themselves may be overlapping and leading to repeated values. This calls for combining cell coverage with a diversity metric. Following previous work on diversity in other contexts, the metric that we use is based on a pairwise Jaccard similarity metric [5]. To handle categorical and continuous columns uniformly, we use a *binning* function to split continuous column values into sub-ranges based on their distribution.

Definition 2.2 (Diversity metric)**.** *Let T_{sub} be a sub-table of T with attributes U_{sub}. Let $\mathcal{B}$ be a* binning *function mapping each column $u \in U$ to a finite set of distinct bins: for a continuous column u, for every value $t(u)$ there exists exactly one bin* $[\text{minVal}, \text{maxVal}] \in \mathcal{B}(u)$ *such that* $\text{minVal} \leq t(u) < \text{maxVal}$. *For a categorical column $u \in U$, for every category value c in u there is exactly one bin $[c, c] \in \mathcal{B}(u)$ that matches it.*

Given a pair of tuples $t, t' \in T_{sub}$, their similarity is defined by

$$Jaccard(t, t', T_{sub}) := \frac{|\{u \in U_{sub} \mid \exists B \in \mathcal{B}(u).\ t(u), t'(u) \in B\}|}{|U|}$$

We then define the diversity of T_{sub} according to the average similarity between its tuples, namely, $divers(T_{sub}, \mathcal{B}) := 1 - avg_{t,t' \in T_{sub}} Jaccard(t, t', T_{sub})$.

Optimization problem Based on the metrics defined above, the optimization goal of SubTab (which also accounts for optional target columns) can be

stated as follows. We are given as input a table T, dimensions k, l, a set of target columns U^* such that $|U^*| \leq l$, a set of association rules $\mathcal{R}$ mined from T, a binning $\mathcal{B}$ of U's columns as in Def. 2.2 above and a parameter $\alpha \in [0, 1]$ (by default, $\alpha = 0.5$). If $U^* \neq \emptyset$ let $\mathcal{R}^* := \{R \in \mathcal{R} \mid \{u_{r_1}, \ldots, u_{r_R+p_R}\} \cap U^* \neq \emptyset\}$, i.e., we retain only the rules whose attributes includes a target attribute, and otherwise $\mathcal{R}^* = \mathcal{R}$. Our goal is to find a $k \times l$ sub-table T_{sub} where $U^* \subseteq U_{\text{sub}}$, i.e., that includes the target attributes, and that among all such tables maximizes the following score:

$$\mathit{combined}(T_{\text{sub}}, T, \mathcal{R}^*) = \alpha \cdot \mathit{cellCov}_{\mathcal{R}^*}(T, T_{\text{sub}}) + (1 - \alpha) \cdot \mathit{divers}(T_{\text{sub}}, \mathcal{B})$$

2.2 Impracticality of an Optimal Solution

Unfortunately, we can show that computing a sub-table that optimizes *combined* is NP-hard (by a reduction from Vertex Cover in a graph with degree 3, details omitted). However, when the sub-table columns are fixed, *cellCov* is a monotone submodular function with respect to the rows. Hence, it follows that if we can enumerate all possibilities for column selection we can execute a *greedy* algorithm over each possibility, and take the sub-table with maximal score to achieve an $(1 - \frac{1}{e})$ approximation of the optimal cell coverage. This approach only works for rows, since *cellCov* is not submodular with respect to columns.

There are several issues with this approach: first, it is typically infeasible to enumerate all $\binom{m}{l}$ possibilities for column selection. Second, it only accounts for cell coverage, and not for diversity. Third, even if columns are fixed and diversity is ignored, the greedy approach is still impractical, due to the need to compute association rules: as mentioned in Section 1, mining rules generally does not meet reasonable requirements for interactive time response.

2.3 Computing Sub-Tables by Table Embedding

Given the impracticality results explained above, our solution takes an approach based on table embedding, which does not explicitly calculate rules nor directly performs coverage optimization. Our algorithm is detailed in Algorithm 1 and explained below.

Algorithm 1 includes three functions: Pre-processing, Embedding Computation and Centroid Selection. Pre-processing/Embedding Computation operate over the raw table $\tilde{T}$/pre-processed table T, respectively, and can be executed as soon as the data is loaded. Centroid Selection is called for each sub-table display, and hence gets as input the sub-table dimensions k, l, an optional SP query Q and a (possibly empty) set of target attributes U^*.

To allow fast response times, Pre-processing first takes a 100K-tuple random sample from the table, then normalizes values (e.g., removes illegal characters) and bins continuous columns, such that values are replaced by bin names (e.g., splits the *Distance* column into short, medium and long distances). Embedding Computation then learns a model $\mathcal{M}$ (line 5) which returns, for each table cell, a vector representation capturing the context (other cells in the same rows/columns) in which it appears. This implicitly captures data patterns in which cells participate, such as association rules.

In Centroid Selection we select rows and columns in the query output that represent diverse data patterns. For that, Q (if provided) is first executed over

the table T; the query output is then considered as the table (line 7). We then capture compute the vector of each row as the average of its cells vectors (averaging is done entry-by-entry), now corresponding to data patterns in which the *row* participates, and save a mapping from vectors to rows (lines 9-11). We cluster the vectors into k clusters and select their centroids as the rows of T_{sub}. We similarly compute U_{sub}, the columns of T_{sub}, as the average of their cell vectors. Since the columns of U^* must be included in the sub-table, we exclude them from clustering, compute only $l - |U^*|$ clusters and then add U^* columns to the selected centroids.

Note that since SubTab does not directly attempt to optimize the cell-coverage and diversity metrics, there are no guarantees it will always obtain high scores. However, when rules are prominent rather then arbitrary, they are likely to be captured by the cell embedding and our solution achieves high scores (See [9]).

3 System and Demo Overview

We implemented SubTab as a local Python library that hooks into Pandas, so users can explore datasets, e.g. in Jupyter notebooks, using the same operations they are used to. To use SubTab the user must import the library and initialize a subtab object on the dataset, given as a Pandas DataFrame. Our system then computes the cells' vector representations, which takes less than a minute. The user can then start exploring the dataset using Pandas queries, and view the generated sub-tables using the display() method of the SubTab object, as illustrated in Figure 2. Since the system is already initialized, generating a sub-table only takes a few seconds.

To assist users in analyzing the resulting sub-table, we developed a visual layer that automatically highlights the captured patterns. As shown in Figure 2, cells associated with the same pattern are highlighted using the same color. By hovering over a highlighted pattern, the user can then see more details, such as its support, confidence, and lift score. This graphical interface also displays the coverage, diversity and combined scores for each sub-table.

SubTab uses Kernel Density Estimation implemented by *sciPy* (`https://scipy.org`) to bin continuous column values. It uses the *gensim* library (`https://radimrehurek.com/gensim`) for computing cell-vectors (cell embedding into vector representations), using a window-size configuration that eliminates the the word proximity factor in natural language sentences, since we assume rows and columns are unordered. The embedding method is configured to generate cell-vectors of size 50 with a minimum cell occurrences frequency of 10. Clustering over the rows and columns is done using the *sklearn* (`https://scikit-learn.org`) implementation of K-means.

Demo scenario Our demo begins with a sample EDA session performed on a dataset (such as the Kaggle Flights dataset described above), using a Jupyter Notebook interface. For each query, we will initially see its intermediate results via the Pandas default sub-table. Next, we will activate SubTab and replay the EDA session. Instead of the default sub-tables, SubTab will generate informative sub-tables for each query in the session. Next, we will invite participants to choose another dataset and optionally one exploration goals. They will then explore it using a Jupyter Notebook interface (with our assistance). After each

query, the participant will be able to inspect the sub-table and evaluate its usefuless by eye as well as through our visual interface, and compare it with the default Pandas sub-table by toggling between the two. We also provide the audience a look under-the-hood of the sub-table generation process.

Acknowledgements This research was partly supported by the Israel Science Foundation (grants No. 639/17 and 2015/21), a grant from the Israel Ministry of Science and Technology, BSF - the Binational US-Israel Science foundation, Tel Aviv University Data Science center, and the Mortimer and Raymond Sackler Institute of Advanced Studies.

References

[1] R. Bordawekar and O. Shmueli. Exploiting latent information in relational databases via word embedding and application to degrees of disclosure. In *CIDR*, 2019.

[2] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *SIGMOD*, 2020.

[3] M. Drosou and E. Pitoura. Search result diversification. *ACM SIGMOD Record*, 39(1):41–47, 2010.

[4] W. Duch, T. Wieczorek, J. Biesiada, and M. Blachnik. Comparison of feature ranking methods based on information entropy. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 1415–1419. IEEE, 2004.

[5] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1015–1030, 2015.

[6] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[7] L. Orr, M. Balazinska, and D. Suciu. Probabilistic database summarization for interactive data exploration. *Proceedings of the VLDB Endowment*, 10(10), 2017.

[8] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, 2016.

[9] K. Razmadze, Y. Amsterdamer, A. Somech, S. B. Davidson, and T. Milo. Selecting sub-tables for data exploration. *arXiv preprint arXiv:2203.02754*, 2022.

[10] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *SIGMOD*, 1996.