645 Mini-Project Report

CS645, UMass Amherst, Spring 2018
Arhum Savera, Aditya Agrawal

Reproducing partial results from SeeDB: Efficient data-driven visualization recommendations to support visual analytics

1. Introduction

In this mini-project, we have tried to reproduce partial results from the methodology supplied by the authors of SeeDB [1], where the authors propose a visualization recommendation engine, which can be used as a middle layer on top of any database. Given a subset of data, it allows fast visual analysis. SeeDB explores the space of all possible visualizations for the given dataset and determines the 'utility' of a visualization, finally recommending the most promising ones. SeeDB uses sharing optimizations, which allows sharing of resources when querying a database and pruning optimizations, which removes visualizations which have very low utility. We have implemented the proposed algorithms via query rewriting (for shared-based optimizations), using Hoeffding-Serfling inequality for pruning-based optimization and by using K-L Divergence as our utility metric on the census dataset [2]. In the next section, we precisely formulate the problem statement of the project, followed by a discussion of the dataset and it's pre-processing. This is followed by a discussion of the specific algorithms- their implementation details and assumptions. Finally, we present and discuss the results of the project.

2. Problem Statement

To find the top 5 aggregate views/ visualizations using K-L Divergence as utility measure from the census dataset [2], by implementing the SeeDB algorithms- Shared-based Optimization through query rewriting and Pruning-based Optimization using Hoeffding-Serfling inequality, for recommending the most useful visualizations when the user queries for the married people (target/user-specified query), as compared to unmarried people (reference query) from our census dataset [2].

3. Dataset

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics) and was originally meant for the prediction task of determining whether a person makes over 50K a year. It contains both quantitative variables such as age, capital gain and hours per week, and categorical variables such as workclass, marital status and occupation. The dataset

is divided into a training set of 32561 rows and test set of 16281 rows. We choose to work on the training set as it would be representative of the data as whole.

4. Methodology and Implementation

i. Assumptions & Preprocessing

The data was obtained as one csv file. We set up a database using with a schema corresponding to the features in the original file. After loading the data, columns with missing values were set to NULL. As a requirement of our project was to compare visualizations of married people against unmarried people, some transformation was required. The marital-status column in the original data has different values such as divorced, separated and widowed etc. We decided to split the data into two groups. These splits were made in an effort to keep the data balanced and fair. The following describes our categorization of rows as married or unmarried:

Married:

- Married-civ-spouse
- Widowed
- Married-spouse-absent
- Married-AF-spouse
- Separated

Unmarried:

- Divorced
- Never-married

A new database with a single table was created using Postgres. Development was done using Python 3.6. The following were the main packages used:

1. psycopg2: For database connectivity
2. Numpy: For numerical computation
3. Pandas: For easy manipulation of query results
4. Scipy: For using KL divergence

The aggregate functions considered are min, max, sum, avg and count.

ii. Exhaustive Search

Identical to the approach described in the paper, we define 3 sets a,m,f which consist of attributes and possible aggregate functions. For each possible view, we execute an SQL query on both the target and reference view and retrieve the results. We then iterate over these results and store them in appropriate data structures. Some of the issues that were brought to our attention included unequal number of rows or different column values for each of the views. If a column has a certain value (eg: native country: Yugoslavia) in one view but not the other, this results in an uneven set of overall results. Simply inserting the appropriate column value with an aggregate value of 0 is not possible as we primarily use KL divergence as our distance measure for calculating utility. If a value of 0 is included when calculating KL divergence,

the result tends to infinity. To avoid this and to maintain the integrity of results, we resort to alpha smoothing. Each column value with a missing or aggregate value of zero is replaced with alpha (10^{-10} in our experiments). After traversing the results, of each query, we then normalize them to scale each result between 1 and 0 and then calculate the utility for that view. We then keep the top k scoring views for comparing the results which will be obtained later by using the optimizations described by the authors [1].

iii. Sharing Based Optimizations

Using the exhaustive approach, results in the execution of 2×f×a×m queries on the database. We group by the same attribute multiple times only to calculate a different aggregate value on a different column. One aspect of the sharing based optimization remedies this. By combining multiple aggregates, we rewrite the query to retrieve a set of aggregates on multiple column values instead of just one at a time. This greatly reduces execution time as the number of executed queries are reduced. Using this scheme, we define a query to retrieve the set of all possible aggregated columns (m x f) and iterate over different attributes in 'a' to group by. The results are then iterated over and stored to calculate utility in a similar way to the exhaustive approach. We again compute results, only on the basis of sharing-based optimizations, which we found to be consistent with the results produced by exhaustive search.

Along with combining multiple aggregates, for the sake of experimentation we tried to observe results when combining multiple group bys, in addition to the multiple aggregates. We generate a single combined query consisting of all possible aggregates and grouping by all attributes in 'a'. As we were grouping by more than one attribute, we faced some challenges in retrieving results for a single value of a column. For example when grouping by A & B, we can see column results in the form of 1, 2 and 1,3, where first aggregated value corresponds to A and second corresponds to B, which results in multiple tuples with A:1. We then implemented an aggregation mechanism to retrieve corresponding tuples for a column value and then use the multiple aggregate values to get a combined aggregation value (eg: max of max or sum of sums). Calculating the aggregated average required using the counts of that column to get the total sum and overall average, per category of ‘a’. Though our results were consistent with our previous approaches, we experienced a noticeable performance dip in terms of execution time. Even though the number of overall executed queries was lower, there was much more computation involved, making this unfavorable. These findings correspond to problem 4.1 described by the authors- regarding finding an optimal grouping of sets, to begin combining *group by*s, to make sure memory utilization does not exceed a threshold and does not lead to overall slower performance.

iv. Pruning-Based Optimizations

We implement the pruning optimization, as described in SeeDB [1], using Confidence-Interval based pruning, which makes use of worst-case statistical confidence intervals to provide a range (or bound) to the utilities of the views. We initialize 20 partitions of the entire dataset, which we did using row numbers. Initially we start with the entire set of views, and then perform sharing-based optimizations to minimize scans of the database per phase (or partition) and then for every view compute the respective utility scores, using K-L Divergence (as described in Shared-based optimizations section 4.iii). We maintain a dictionary data structure of all the views and iteratively append the updated utility scores. Additionally,

we also maintain a statistics data structure, for maintaining a record of the mean utility score per view, respective lower bound, and upper bounds after every iteration. After the 'sharing-based optimized' queries are executed on the partition, the utility score is appended to the dictionary, mean utility score and CI (Confidence intervals) are updated in the statistics data structure. The confidence intervals are calculated using the Hoeffding-Serfling inequality [3], where we make the assumption that no pruning takes place in the first iteration, as no confidence intervals are established, because setting the m = 1 leads to the epsilon term in the inequality equaling to infinity. After the first iteration, subsequent iterations lead to update in the mean utility scores and confidence intervals for every view. Every iteration of computing utility scores, except the first one, is followed by pruning of low-utility visualizations using the algorithm described by the authors, where every view is removed whose upper bound is lower than the lower bound of the 5th visualization amongst the top 5 visualizations for that iteration. This is possible as the authors claim, with high probability, that all the visualizations which have a upper bound lower than the lowest bound amongst the top 5 visualizations will never appear in top-5 views. This process of removing views, after re-computation of utility scores, mean utility and confidence intervals, after every iteration, leads to decline in the views until we are only left with top k views, at the end of our 20 iterations (where k << total set of views).

5. Results

We can observe below that top 5 visualizations possess distinct deviations between the results of the query on target data (married adults represented in blue) and the reference data (unmarried adults represented in green). K-L divergence allowed the computation of the quantitative distinction between the results of the queries underlying the visualized data. These were selected as the top 5 amongst our entire space of visualizations, as these views emitted the highest K-L Divergence scores/ entropy scores (SciPy), which pointed towards the stark difference which would be rendered in the corresponding visualizations, which in turn could potentially be interesting for the analysts. We can see below the top 5 visualizations, where Fig 1. was the top recommended one and Fig. 5 was the last amongst top-5

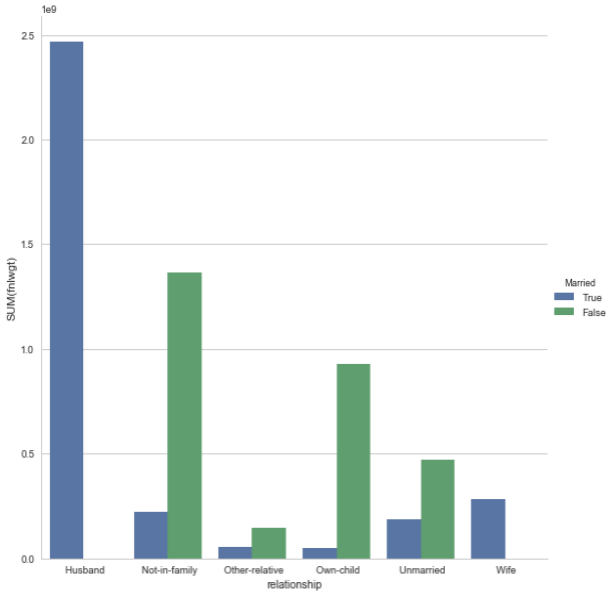

Fig 1. SUM(fnlwgt) vs Relationship

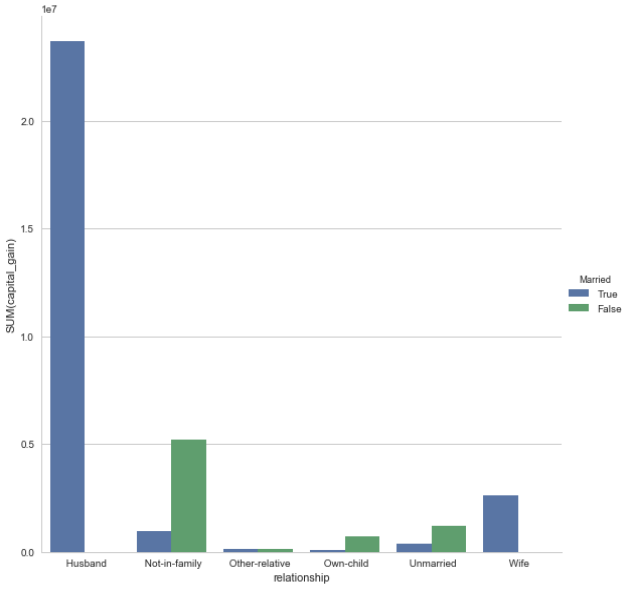

Fig 2. SUM(capital_gain) vs Relationship

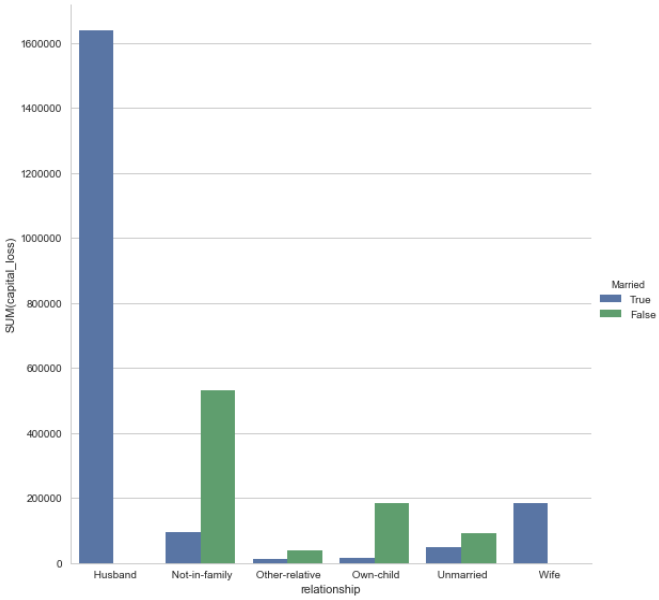

Fig 3. SUM(capital_loss) vs Relationship

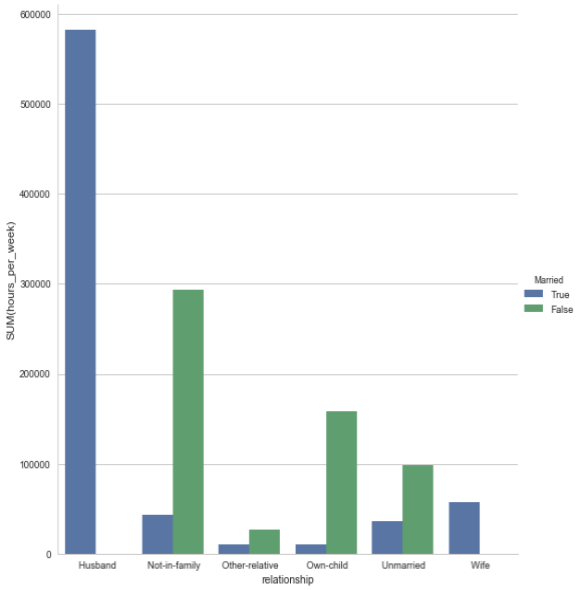

Fig 4. SUM(hours_per_week) vs Relationship

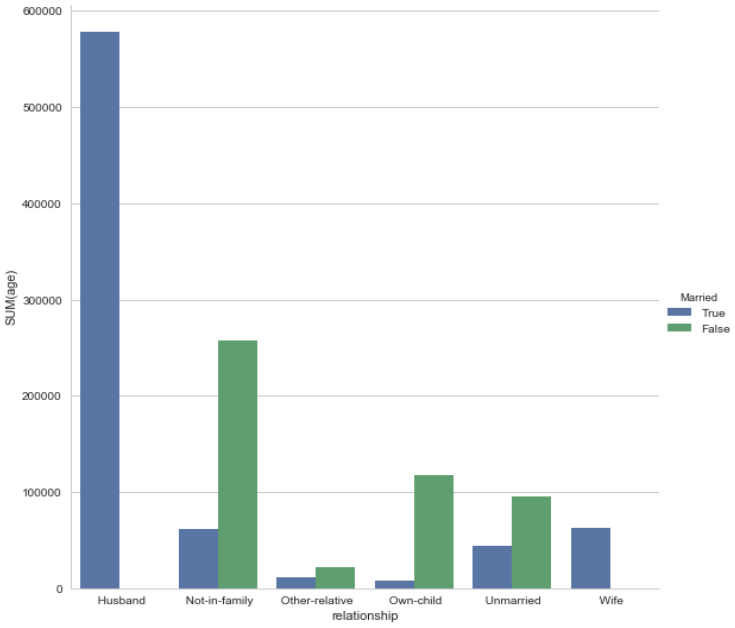

Fig 5. SUM(age) vs Relationship

6. Conclusion

We have presented the results for the top 5 aggregate views, using K-L Divergence as the utility measure, by implementing the algorithms provided in SeeDB [1], which allows us to explore the vast space of visualizations, for a given data subset, by using optimizations for reducing the rendering or querying of less important/useful visualizations. We have implemented Shared-based optimization by rewriting the queries, which allowed minimal querying of the database by sharing computations and Pruning-based optimizations by using the Hoeffding-Serfling inequality, which allowed expunging of redundant views after every phase.

7. References

[1] M. Vartak, S. Rahman, S. Madden, A. Parameswaran and N. Polyzotis, "SeeDB", Proceedings of the VLDB Endowment, vol. 8, no. 13, pp. 2182-2193, 2015.

[2] "UCI Machine Learning Repository: Census Income Data Set", Archive.ics.uci.edu, 2018. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Census+Income. [Accessed: 04- May- 2018].

[3] R. J. Serfling et al. Probability inequalities for the sum in sampling without replacement. The Annals of Statistics, 2(1):39–48, 1974.