Reproducibility Project

SeeDB: efficient data-driven visualization recommendations to support visual analytics

Team : Arjun Sreedharan, Ben Cheung, Hao Liu

Introduction

SeeDB is a visualization recommendation engine to facilitate fast visual analysis introduced in the paper *SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics* [1]. Given a dataset, there could be many visualizations. SeeDB scores a utility value for each visualization and recommends those of top utility value as most “interesting” or “useful”. Scoring “interestingness” or “usefulness” requires a metric. The paper uses Kullback–Leibler divergence for that.

The paper discussed use of two optimizations, the sharing-based and the pruning-based, and claimed achieving multiple orders of magnitude speedup. We implemented both the naive approach and the two optimizations. We first applied shared based optimization and on top of that pruning based one.

Basic Assumptions

From the census data source, we identified fields that were applicable to aggregating on and grouping by. Basically, finite (small set as well like age) numbers and expected options for fields are groupable: ‘'age', 'workclass', 'education', 'education_num', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country'. And numbers in general that aren’t explicably finite (large sets like capital loss) were considered aggregatable: 'capital_gain', 'capital_loss', 'hours_per_week'. For our aggregate functions, we applied: min, max, mean, count, sum.
For the user defined query, splitting the married and unmarried ended up filtering the table for marital_status being either ‘married’ or ‘Never-married’, respectively.
Once only k Views remain, even if it is not the last phase, the phased execution terminates, and the k remaining Views are returned immediately.

Infrastructure

Preliminary work of database setup and configuration of “adult.data” is on Postgresql. Implementation is coded in Python 3 with major lib uses of pandas, psycopg2.
The plotting is done by matplotlib.

From the original census data set, we segregated the dimensions and measures. We have a *partitioner* that splits the dataset into multiple equally sized portions to facilitated phased execution. This is materialized using SQL views. For e.g. if there are *m* partitions, we create SQL views *part_1, part_2,... part_m*. After that, we run each phase one after the other.

In each phase querying is done with Sharing-based Optimizations (Sec. 4.1 of the paper), and we deal with scale by discarding low utility views as described in Pruning-based Optimizations (Sec. 4.2 of the paper). We use a data-structure *mapOfPriors* to keep track of the mean utility for every aggregate view which is used to derive the confidence interval around it helping discard low-utility views in each phase.

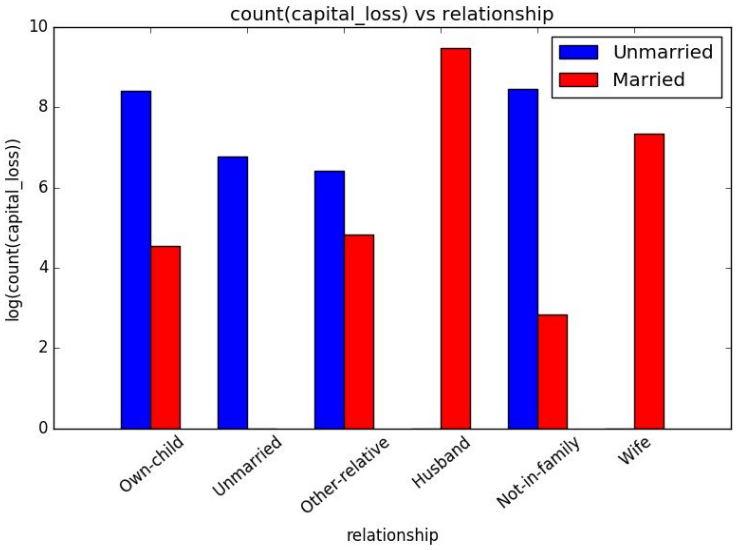
Finally, *matplotlib* is used to create visualizations from the already found lists of best *(a, f, m)* tuples.

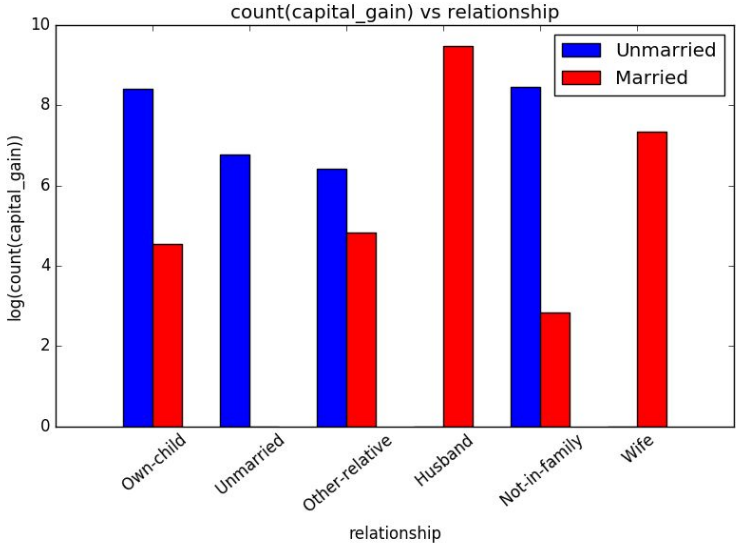
Results

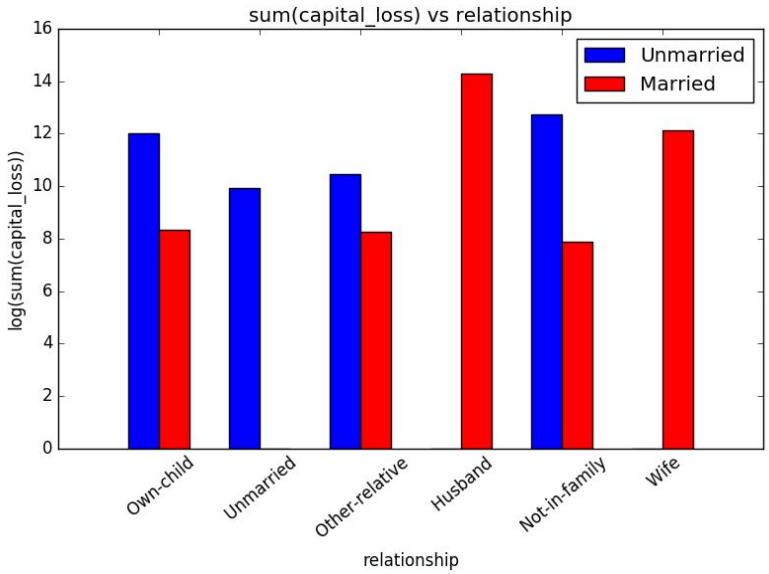
We found the options most interesting visualizations are all of group by “relationship”, which are:
count(capital loss) group by relationship,
count(capital gain) group by relationship,
sum(capital loss) group by relationship,
count(hours per week) group by relationship,
sum(capital gain) group by relationship.
Graphs come at the end of this section.

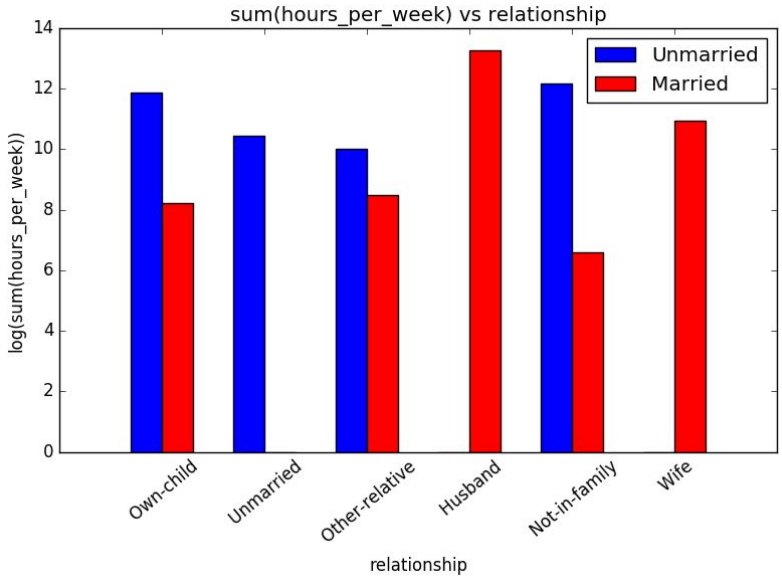
The red bar and blue bar are in fact significantly different. It’s not reflected in the bar charts as they show logarithmic values. It is because these aggregation values differ in magnitude and can’t be plotted properly without taking logarithm.

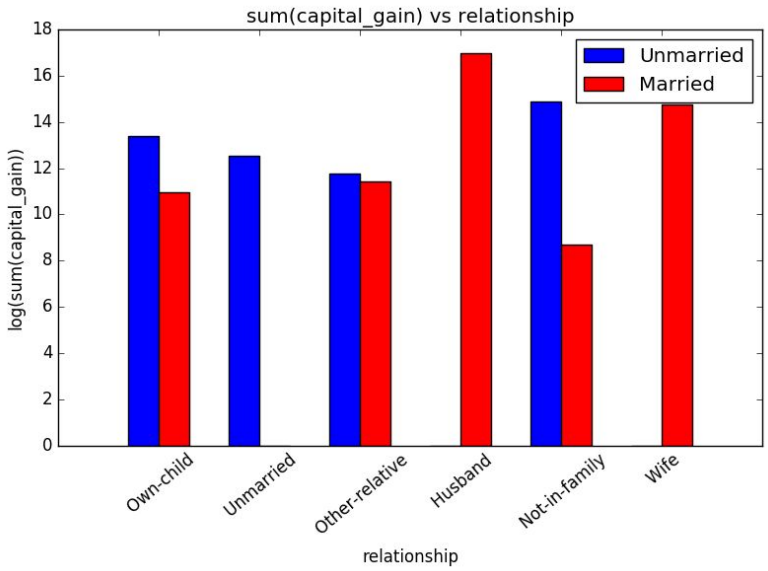
This can be reasoned intuitively that aggregations by “relationship” can differ most dramatically between the reference group of “unmarried” and the target group of “married”, since being “married” or not is correlated with the “relationship” value. For instance, the “relationship” value “unmarried” does not occur at all in the target group “married”, therefore making the discrepancies of aggregations by “relationship” significant. Another example would be “Husband” and “Wife” correspond to 0 entries in the target group “unmarried”, which skews aggregations in the reference group and makes the views more “interesting”.


count(capital_loss) vs relationship
Unmarried
Married
log(count(capital_loss))
0
2
4
6
8
10
Own-child
Unmarried
Other-relative
Husband
Not-in-family
Wife
relationship


count(capital_gain) vs relationship
Unmarried
Married
log(count(capital_gain))
0
2
4
6
8
10
Own-child
Unmarried
Other-relative
Husband
Not-in-family
Wife
relationship


sum(capital_loss) vs relationship
Unmarried
Married
log(sum(capital_loss))
0
2
4
6
8
10
12
14
16
Own-child
Unmarried
Other-relative
Husband
Not-in-family
Wife
relationship


sum(hours_per_week) vs relationship
Unmarried
Married
log(sum(hours_per_week))
0
2
4
6
8
10
12
14
Own-child
Unmarried
Other-relative
Husband
Not-in-family
Wife
relationship



Comments

We also wondered, what was the best number of partitions and alpha value were in terms of reducing running time. We need to note, that changing delta may affect the results of the top k outputted. As increasing delta will increase the rejection rate and increase the risk of rejecting a view that should rightfully belong in the top k. Regardless, we tried the computation on number of partitions from 2 to 40 and delta in [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]. Usually, statisticians won't go below a 90% confidence, so we did not here. And we saw that the trend of increasing the number of partitions also increased the amount of time needed. So we did not try past 40. We ran the computations 3 times for each partition_number and delta pairing and found the average, sorting by best average. We found at delta = 0.07 and partition at 2, we get the fastest times. But it is to note, that we also wrote a naive implementation of this (just the section 2 algorithm, for the true best utility for the whole data set) and we found different results. On top of that, the naive ran faster than the optimizations. This was because of the overhead costs in our optimizations (connection opening and closings to psql server, python pandas dataframe manipulation costs, etc) that did not outweigh the benefits of reducing the number of queries run. Given a much larger dataset, reducing the number of queries will become more fruitful with the current overhead.

References

[1] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis, "Seedb: Efficient data-driven visualization recommendations to support visual analytics".

Appendix

The entire code for the project is available in github under the code name probable-waffle (https://github.com/taichouhb/probable-waffle)