Optimal Histograms with Outliers: a Python Implementation and Experimentation

Alexandros Kalimeris
kalimerisalx@gmail.com
National and Kapodistrian University of Athens
Athens, Greece

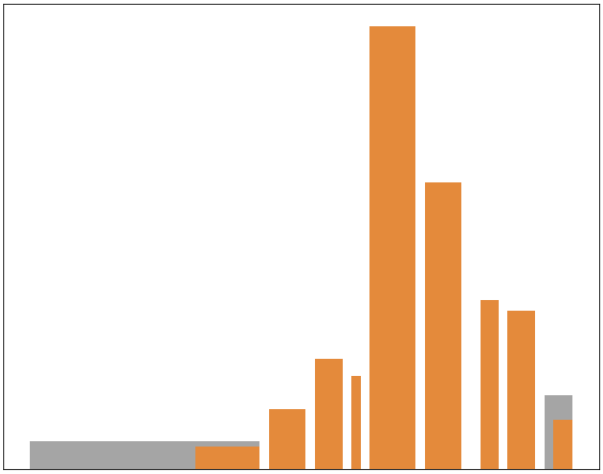

Figure 1. A sample v-optimal histogram with no deletions (gray) visually compared to one with deletions (orange)

Abstract

Histograms are a traditional and well-studied way of summarizing data. They have been extensively used in many applications that require frequency estimates, such as query plan cost estimation. In this project, we will be reviewing the paper "Optimal Histograms with Outliers" [2] that is examining the creation of v-optimal histograms while allowing for the deletion of possible outliers in the dataset.

The paper we are examining, proposes a set of polynomial-time dynamic programming algorithms for the task. After thoroughly studying the paper, we implemented the algorithms and recreated the most important experiments that are presented, as well as incorporated new datasets. All the algorithms described in the paper and used in the experiments have been implemented in a compact python library that will be openly available online[1].

Keywords: histograms, v-optimal histograms, dynamic programming, frequency estimation

[1]https://github.com/AlexandrosKal/opt_hist

1 Introduction

The construction of a histogram over a dataset is a very useful and intuitive tool for summarizing datasets. A well constructed histogram can provide us with great insight of the distribution of the datapoints in our dataset. However, covering a big dataset with a histogram is often accompanied with an amount of error.

There are different kinds of histograms, equi-width, equi-depth and v-optimal. The kind of histogram we are going to explore is the v-optimal. The goal of a v-optimal histogram is to have the smallest variance possible within each bucket, which provides for a smaller error. Research done by Poosala and Ioannidis [5] has demonstrated that the most accurate estimation of data is achieved by a v-optimal histogram using value as a sort parameter and frequency as a source parameter.

The goal of the paper we are reviewing in this report is to create a histogram that is concise while, at the same time, it captures the dataset as closely as possible. This task is achieved by finding an optimal histogram with at most β buckets and a number k of outliers, that we do not need to

cover in the histogram, as those points are usually responsible for increasing the error.

This review will be organized as follows: First we are going to go over the original paper and its contributions, then we are going to discuss our implementation, afterwards we are going to present the reproduced experiments of the original paper as well as experiments on new data and finally discuss and present our conclusions.

2 Overview of the paper: Optimal Histograms with Outliers

Having briefly mentioned some of the main points of the paper that we are studying, in this section, we are going to explain its concepts, goals and contributions in greater detail.

2.1 Histogram error, buckets and outliers

As we pointed out earlier, covering a dataset with a histogram with a limited number of buckets usually comes with a significant amount of error. The easiest and most simple way to reduce the error is to add more buckets in our histogram. In many cases, this is not a great solution as doing so decreases the expressiveness of the histogram and, especially in the case of humans interacting with the histogram, decreases the ability of the user to capture the "big picture" behind the data. Thus, finding another way to decrease the error of a histogram is indeed an important task.

The authors of the paper consider that most datasets contain an amount of "outliers", that are points in the dataset that significantly deviate from the distribution that characterizes the data, and the identification and removal of those elements will significantly decrease the error of the histogram.

2.2 Key Definitions

Here we are going to provide some key definitions of the notions that are being discussed in both the paper and this report. Starting with the frequency vector $\overrightarrow{f_D}$ where $\overrightarrow{f_D}[q]$ denotes the number of the occurrences of the element q in the dataset D, for example:

$$D = \{\{0, 0, 1, 2, 3, 3\}\} \text{ then, } \overrightarrow{f_D} = \{0 : 2, 1 : 1, 2 : 1, 3 : 2\}$$

Next we are going to define the notion of a summary. A summary B is a set of buckets $([p, q] : n)$ where p and q represent the edges of the interval (range) of the bucket and n is the number of elements of the dataset that fall inside that interval.

Considering histograms are a tool that provides us with frequency estimations, it is crucial to define the frequency estimation vector $\overrightarrow{e_B}$. A frequency estimation vector associated with a summary B, is denoting the estimated frequency of elements in the summary. For an element r:

$$\overrightarrow{e_B}[r] = \frac{n}{q - p + 1}$$

is its estimated frequency if r is within the range of a bucket of the summary, otherwise $\overrightarrow{e_B}[r] = 0$.

Finally we have to define how we are calculating the error of the summary B w.r.t a dataset D . For the purposes of the paper and this report, the error is calculated as the distance of the actual frequency vector $\overrightarrow{f_D}$ from the estimated frequency vector $\overrightarrow{e_B}$. Many distance metrics can be used, in this project we are going to use the sum squared error (SSE) defined as:

$$err(B, D) = \sum_{([p,q]:n) \in B} \left(\sum_{p \le r \le q} (\overrightarrow{f_D}[r])^2 - \frac{n}{q - p + 1} \right)$$

2.3 Contributions

Given what we have discussed above, we are now going to present the main contributions of the paper. Ultimately, the goal is to create a histogram with a bound β on the number buckets and a bound k on the number of ouliers we are allowed to delete. Towards that end, the authors of the paper make the following main contributions.

2.3.1 Intuitive Deletion Strategies. The authors propose two new intuitive notions for an optimal histogram with deletions. The first one called *Summarize then Delete*, where we first compute a summary with the least error without any deletion and afterwards, on the buckets we have obtained, we try to perform up to k deletions so as to lower variance within each bucket.

The second strategy called *Delete while Summarizing* proposes a strategy that tries to perform the deletions *while* building the histogram. There is also another deletion strategy mentioned in the paper called *Delete then Summarize*, that method is used more like a baseline to compare with the others. According to that method, we first delete at most k outliers from our dataset with the assistance of an outlier detection algorithm and then we create a summary with no deletions on the data that is supposedly outlier-free.

The authors of the paper consider two cases when performing deletions. The first case considers deletions called Arbitrary meaning that they can be performed at any position of the bucket in order to reduce the in-bucket variance. The second case considers deletions called Consistent, those deletions can only be performed at the left or the right edge of the bucket and delete all the occurrences of that edge value. A bucket range resulting from Consistent deletions will have the same number of elements as the original dataset within that range thus keeping consistency.

2.3.2 Algorithms. The authors of the paper provide detailed implementations in pseudo-code of their algorithms as well as in depth proofs for their complexity and correctness. The algorithms follow a bottom-up approach, starting

by implementing simple algorithms that work for a single bucket or a summary with no deletions and then using those as building blocks for the algorithms for summaries with multiple buckets and deletions.

2.3.3 Extensive experimentation. Another contribution of the authors, is the extensive experimentation and comparison of the different algorithms and approaches, in real-life as well as synthetic datasets in order to provide an informed view of each algorithm's capabilities and limitations.

3 Implementation

As a part of this project we implemented all the algorithms described in the paper in a compact Python library. We have chosen Python due to its ease of use that allows for any researcher or student to easily delve into the code understand it, make changes and experiment with new ideas by himself. Python as an interpreted language, is slower than Java that the authors of the original paper used for the experiments, however in our opinion the accessibility that Python offers is more beneficial for a didactic approach. In the end, any idea or change deemed really useful could be sped up by using some techniques that will be discussed later.

3.1 Design Approach and Packages

The algorithms are implemented following an Object Oriented approach. There is a base Summary class that all the other summary methods are derived from. We think that this design is more intuitive and easy to use, as each summary instance has the arrays and variables it needs as class attributes already instantiated as well as the respective methods.

The implementation uses some popular Python packages for some functionalities. For the numerical operations needed the popular python package NumPy[4] is used. For the loading and handling of the data we use the Pandas[8] package. We also use a python module called stopit[7]in order to impose time limits on the algorithms during the experiments.

We should also note that for the *Delete then Summarize* strategy we could not find any implementation of the outlier detection algorithm[1] mentioned in the paper so we used an outlier detection algorithm[3] from sklearn[9] as a substitute during the experiments and as we will see later the results were quite similar.

3.2 Extensions

The algorithms that are presented in the original paper compute only the error value of each histogram, that is enough to compare the performance of the algorithms but in order to use those algorithms in practice, the algorithm should, during its execution, compute and keep track of the bucket edges and elements inside each bucket in order to return a complete summary to the user. In this project, we have added matrices to the dynamic programming algorithms that keep track of the edges and the deletions during the creation of the summary. By backtracking on those matrices, we are able to get the full summary in a usable form that can be easily visualized as we saw in (Figure 1). This functionality was implemented for the *Summarize then Delete with Consistent Deletions* (SD CON) and for the *Delete while Summarizing with Consistent Deletions* (CON) as well as the *No Deletion* approach. As we will see in the experiments later, those algorithms are the most usable ones, mainly due to execution time limitations. A visual comparison between the summarizing with no deletions and summarizing with k optimal deletions according to the CON algorithm can me seen in (Figure 2).

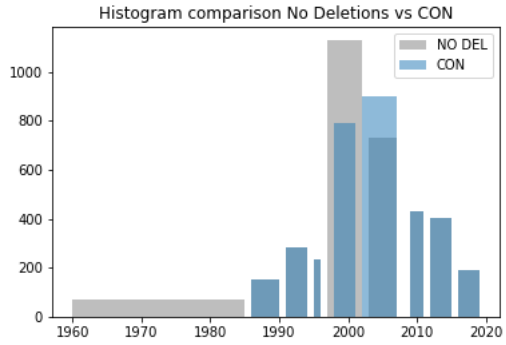

Figure 2. A visual comparison of the resulting v-optimal histogram without deletions and the one resulting from CON algorithm.

3.3 Python and execution time

Since Python is an Interpreted language it is naturally slower than other compiled languages such as Java or C++. The speed of standard python is enough for educational or research purposes. Though, if we want to use the algorithms on real large-scale datasets in practice, we need to consider how to speed things up. An option we are going to explore in this project is using just-in-time compilation with the help of the Numba [6] module in order to speed up the execution of an algorithm. Other options include creating python bindings to C++ code or if necessary changing the development platform to a compiled language.

4 Experiments

In order to review the capabilities of the algorithms and also the reproducibility of the results of the paper, we are going to conduct an extensive series of experiments on both synthetic and real datasets. We should also mention that for the rest of this report the algorithms will be referred as follows:

- No deletion Summary: NO DEL.
- Delete then Summarize: DS.
- Summarize then Delete with Consistent Deletions: SD CON.

- Summarize then Delete with Arbitrary Deletions: SD ARB.
- Summarize while Deleting with Consistent Deletions: CON.
- Summarize while Deleting with Arbitrary Deletions: ARB.

4.1 Datasets

We use the some of some of the real-life datasets mentioned in the original paper taken from the UCI KDD Archive [2] and the UCI Machine learning repository[3]. Specifically, the Income dataset (referred to as Adult in the repository). We use the columns CapitalGain and HoursWeek, which is the number of work hours per week. We also explore a new dataset [4] that is related to used cars in Belarus, we specifically use the *year_produced* column. The datasets are also available on the project github repository.

For synthetic experiments, we use the same distributions that are mentioned in the original paper. We perform experiments using a normal distribution with variance equal to 0.25 of the range and a permuted zipf distribution with an exponent of 0.85. The range that both distributions draw samples from is [1, 100] and the experiments are performed in 3 iterations from which we take the mean error and execution time to represent on the graphs (this exact method and parameters were used in the paper).

4.2 Experimental Results

In this section we are going to present the results of the experiments and a brief discussion/explanation of those. To make things more clear to the reader, in the plots that involve error, the number depicted is the error ratio of the errors resulting by the respective algorithm divided by the error of applying the NO DEL algorithm.

Due to the size of the datasets we used a 10% uniform sample of each dataset. The summarising algorithms are allowed to run for 120 seconds to keep the execution time of the experiments reasonable. For the synthetic datasets, we draw 2000 samples unless otherwise stated. By default, we use a parameter $\rho = 2\%$ of the size of the dataset for the number of the allowed deletions and $\beta = 10$ buckets.

We should also note that due to the wide variety of experiments performed, in order to avoid cluttering the report with an excessive amount of figures, we will showcase the comparison of some indicative plots from the paper with our own results. For the others, we will discuss the results and any claims made can be easily verified by looking at the figures of the paper as well as the results in the project repository.

[2] http://kdd.ics.uci.edu/
[3] http://archive.ics.uci.edu/ml/index.php
[4] https://www.kaggle.com/lepchenkov/usedcarscatalog

All the experiments where run on my personal laptop that has 16GB of RAM and an AMD Ryzen 5 4600H CPU.

4.2.1 Synthetic Data. We are going to start with the experimental results by first taking a look at the synthetic datasets. As we can see in (Figure 3) the results are quite simillar to the ones presented in the paper that we are trying to reproduce. DS yields the highest amount of error followed by SD CON, followed by CON followed by SD ARB and ARB. When we look at the execution time (Figure 4) we see that the order is reversed with ARB being by far the slowest algorithm. This is expected because it examines all the possible bucket edge combinations as well as deletions from anywhere inside each bucket.

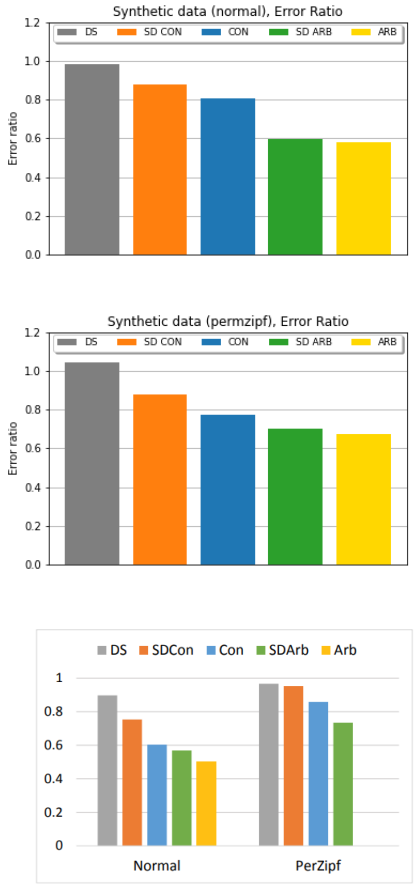

Figure 3. Error ratio obtained from the experiments on synthetic data (top, middle), compared to the figure taken directly from the paper (bottom).

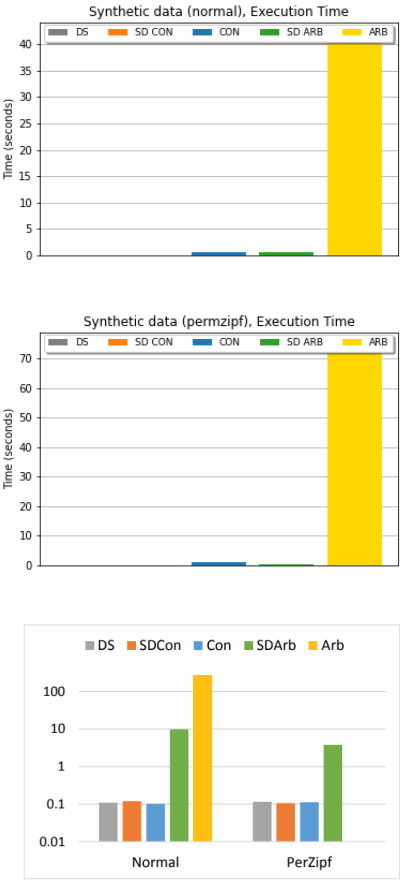

Figure 4. Execution time obtained from the experiments on synthetic data (top, middle), compared to the figure taken directly from the paper (bottom).

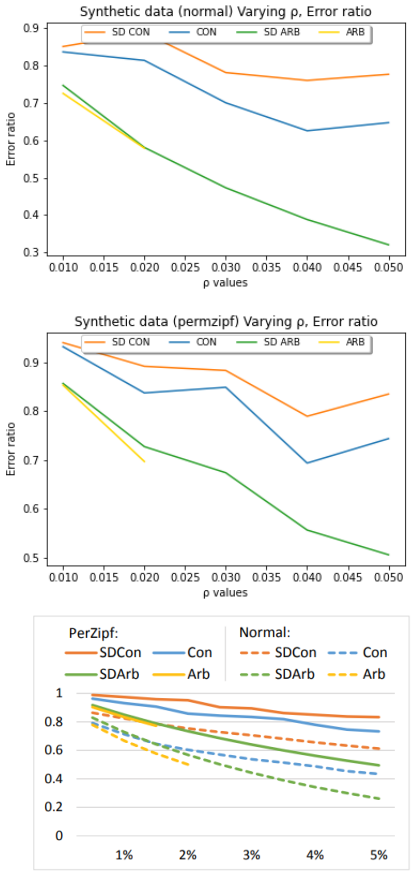

Figure 5. Error ratio obtained from the experiments on synthetic data while varying the percentage of deletions (top, middle), compared to the figure taken directly from the paper (bottom).

4.2.2 Synthetic Data Varying ρ.

In a similar fashion as the previous experiment, we keep the number of buckets constant and examine the behavior of the algorithms while we vary the percentage of deletions ρ. As observed in the graphs (Figure 5) the error tends to decrease as the percentage increases. On the other hand (Figure 6) the execution time for the ARB algorithm quickly explodes, for the SD ARB we see a significant but not that dramatic increase in time while for the other two algorithms the time remains quite low. The same results are observed in the figures from the paper.

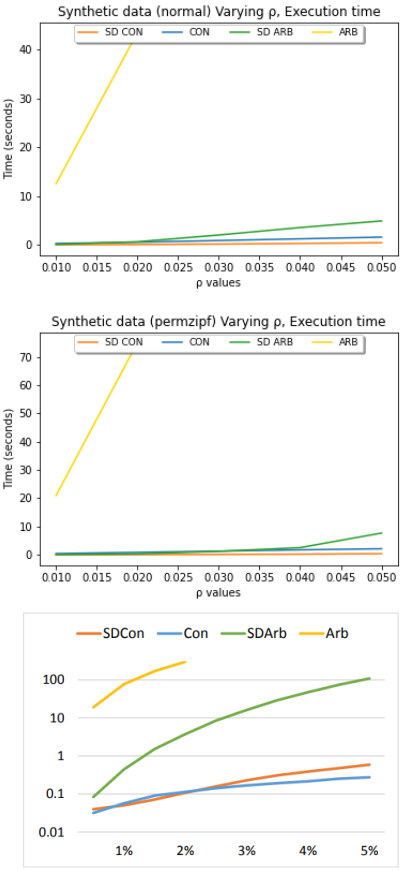

Figure 6. Execution time obtained from the experiments on synthetic data while varying the percentage of deletions (top, middle) compared to the figure directly taken from the paper (bottom).

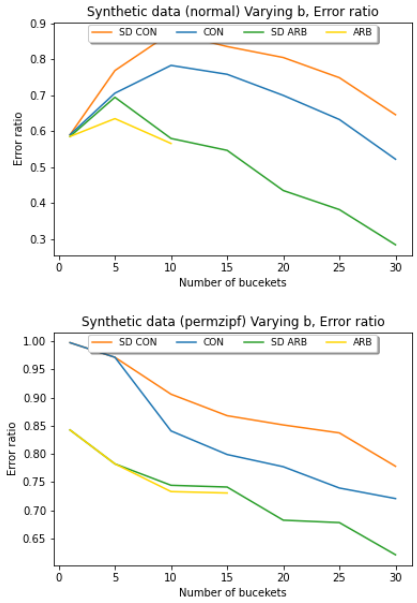

Figure 7. Error ratio obtained from the experiments on synthetic data while varying the number of available buckets.

4.2.3 Synthetic Data Varying β.

Performing the same experiment but this time with varying number of available buckets, we observed quite similar results to the previous ones. Matching the behavior we observe in the paper. We will showcase one indicative figure (Figure 7).

4.2.4 Synthetic Data Standard vs JIT-Compilation.

As we can easily notice in the figure (Figure 8), adding a simple optimization of just-in-time compiling some for loops with the assistance of the Numba python package, can lead to a significant speed up as the size of the dataset increases.

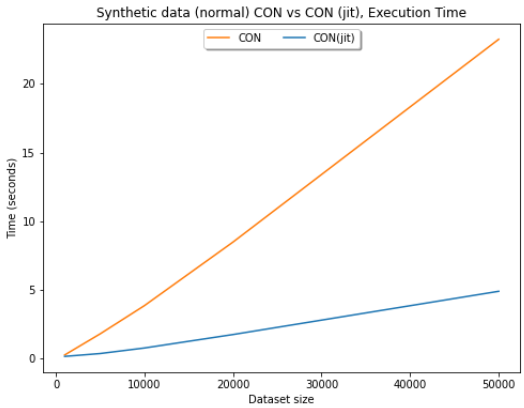

Figure 8. Execution time of the CON algorithm when ran with standard interpreted python and when ran with JIT-compilation

4.2.5 Real Data.

We will now move to the results of experiments on real datasets. We ran the experiments described in the paper on the *capital gain* and *hours per week* columns of the Income dataset used in the paper, as well as the new used car dataset. In order to conserve some space we are going to

showcase the results form the *capital gain* column as we had a small difference from the results shown in the paper while *hours per week* showcased really similar behavior to the one described in the paper. We will also present the results on the new dataset.

On the capital gain dataset we again observe (Figure 9) a pattern that seems to be the norm. SD ARB and ARB provide the highest error reduction while at the same time being the two slower algorithms in execution time (with ARB being the slowest, timing out in the experiments). It is worth noting here, that the DS algorithm produced a better error than SD CON. Given the results of the other experiments DS always performs the worst regarding the error ratio, so this most likely seems to be an anomaly due to some nuance of the data we sample. Or it might be evidence that if an outlier detection algorithm fits a specific dataset it has quite satisfying results.

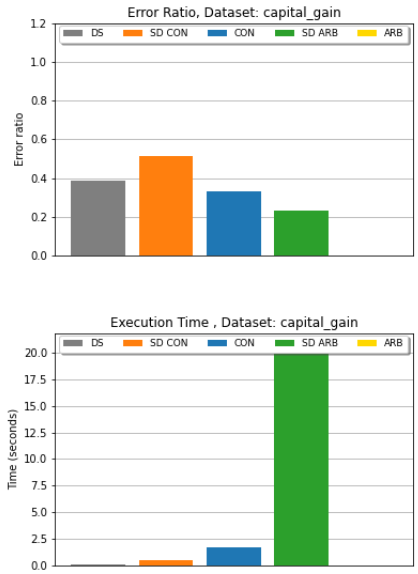

Figure 9. Error ratio (top) and execution time (bottom) on the capital gain dataset (ARB bar is empty because it timed out in the experiment).

On the car dataset we again notice the same pattern (Figure 10) that now begins to solidify itself as it is not only reproducible in the datasets the authors of the paper used, but it is also observed in a new dataset. The ARB algorithm provides the lowest error while being the slowest, followed by SD ARB that is providing similar error while being less slow, then we have CON that is again on the middle ground providing a balanced tradeoff between error reduction and execution time. Lastly, SD CON is the fastest but it provides the lowest error reduction almost the same as DS.

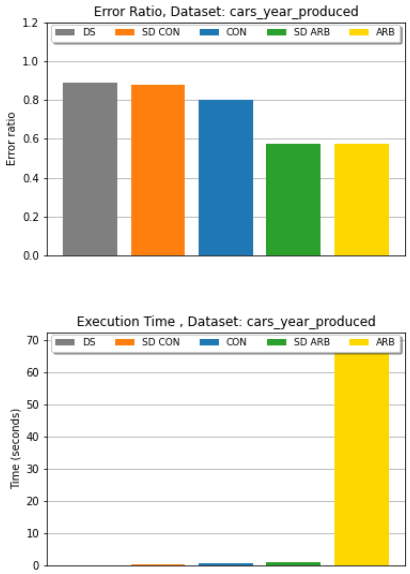

Figure 10. Error ratio (top) and execution time (bottom) on the cars year produced dataset.

4.2.6 Real Data Varying ρ and β.

We again notice the same repeating pattern as in the paper and in the synthetic data above. Some indicative results from our experiments on the cars dataset are presented in (Figure 11).

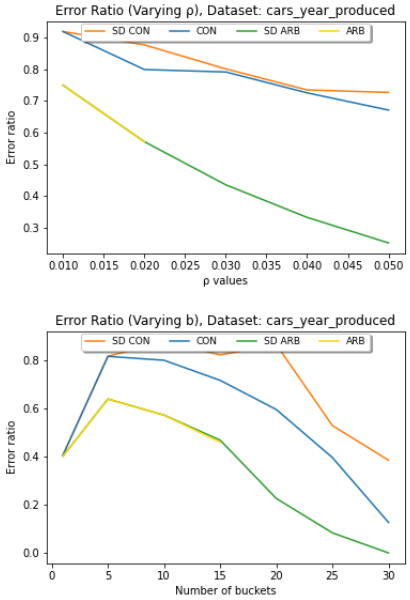

Figure 11. Error when varying ρ (top) and β (bottom) on the cars dataset

4.2.7 Sampling versus Optimal Deletion. A common practice in systems that construct histograms for large scale datasets is to costruct the histogram on a sample of the full dataset. One can think that constructing the histogram on a sample is achieving similar results to the approach of the algorithms presented in the paper. In order to test that we reproduce an experiment described in the paper. We take a synthetic dataset (we test both the normal and permzipf case) and take a sample of it. Then we execute the NO DEL algorithm to create a summary on it. Then we fit the whole dataset in the buckets we obtained by NO DEL on the sample and consider any element not fitting a deletion. Finally we execute the con algorithm on the full dataset with the same number of deletions and compare. The results can be seen in (Figure 12).

As we can see in the in the permzipf distribution we get no deletions because of the way the data is distributed so the methods yield the same results. On the normal distribution data though, we see both in our results as well as in the figures of the paper, that the sampling method not only is worse but it increases the error, making it clear that sampling cannot be an effective technique in order to reduce the error of a summary.

5 Conclusions

In this report, we studied the paper Optimal Histograms with Outliers by Rachel Behar and Sara Cohen, that is examining the topic of constructing optimal histograms while, at the same time, allowing the deletion of outliers. We also presented our effort in implementing the algorithms proposed by the authors, reproducing the experiments as well as experimenting on new data. Through our experimentation, we got quite similar results to the original paper even on the new data, that leads us to reinforce the conclusions that the authors presented. Deleting elements while constructing the summary (histogram), is a really effective way of creating an optimal histogram. It is also evident, that the algorithm ARB provides the best error reduction however its runtime makes it unusable for large or even medium-sized datasets. Furthermore, the algorithm CON seems to be the algorithm provides the best tradeoff between runtime and error. The performance of the algorithms while effective for moderate sized datasets is degrading for extreme scale datasets. This, naturally makes approximate algorithms allow for faster runtimr a valid goal to consider for the future.

We should also mention that providing openly available Python code and our extension in SD CON and CON functionality that returns the full summary details, are steps forward that will be of use to any researcher or student that wants to dive into this topic in the future.

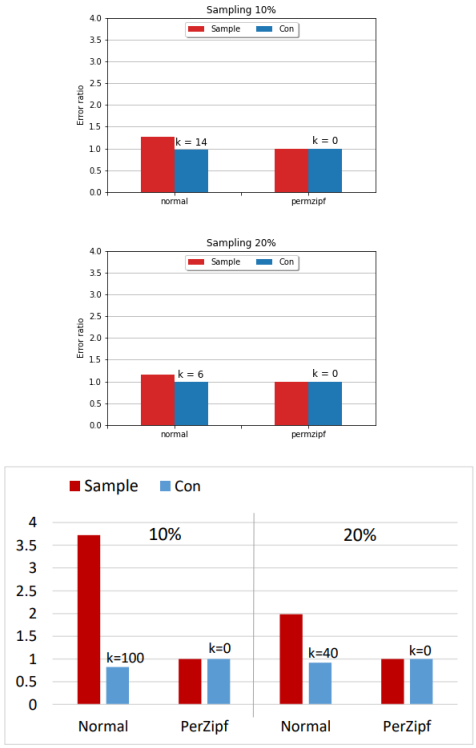

Figure 12. Comparison of the Sampling method with the CON algorithm. Tested with a 10% (top) and a 20% (middle) sample in both normal and permzipf distributions. The figure presented in the paper is also shown for comparison (bottom).

References

[1] Stephen D. Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, Lise Getoor, Ted E. Senator, Pedro M. Domingos, and Christos Faloutsos (Eds.). ACM, 29–38. https://doi.org/10.1145/956750.956758

[2] Rachel Behar and Sara Cohen. 2020. Optimal Histograms with Outliers. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 181–192. https://doi.org/10.5441/002/edbt.2020.17

[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein (Eds.). ACM, 93–104. https://doi.org/10.1145/342009.335388

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre

Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[5] Yannis E. Ioannidis and Viswanath Poosala. 1995. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, Michael J. Carey and Donovan A. Schneider (Eds.). ACM Press, 233–244. https://doi.org/10.1145/223784.223841

[6] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A LLVM-Based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (Austin, Texas) *(LLVM '15)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/2833157.2833162

[7] Gilles Lenfant. [n.d.]. stopit. https://pypi.org/project/stopit/

[8] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.