Data Sampling for Big Data

Risto Tuomainen

March 31, 2016

Table of Contents

Big Data and Sampling

- Handling small data is easy while handling big data is difficult, so why not make big data small?
- From statistics we know that sampling for often give results that are practically as good as the exact values
- The methods here come not from actual Big Data literature, but rather from earlier OLAP research
- However, even if the methods themselves predate Big Data, they have been put to use recently for example in BlinkDB system for Big Data
- Central to big data applications would be sampling stream data. We will not be discussing that however, but rather focus on static datasets

Sampling

- Sampling has always been central to statistics, and has been extensively researched
- In survey research collecting data is expensive (while analyzing it is cheap), so sampling to limit data
- In big data analyzing data is expensive (while collecting it is cheap), and again sampling to limit data
- Most straightforward idea is to just sample uniformly at random

Contrast to classical setting

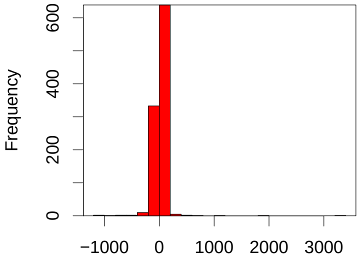
- Often in statistics we think about sampling values at random from some parametric distribution is order to estimate the parameters
- For example we might sample a normal distribution to estimate the mean, and in this case we know very well how for example sample mean function behaves (Student's t-distribution etc.)
- Now we are doing something different, that is sampling a finite collection which we can, if we so desire, scan several times. We can for example find the minimum and maximum values in this collection.
- This difference in setting will lead to some theory which is not so familiar from elementary statistics

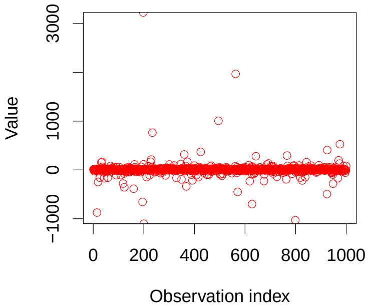
Problems with uniform sampling

- Uniform sampling will sometimes yield abysmal results
- In this presentation we are concerned with a specific failure mode:
- If the data is spread on a large interval, obtaining useful estimates can require large samples
- A very specific method to handle this situation

Sampling sparse data

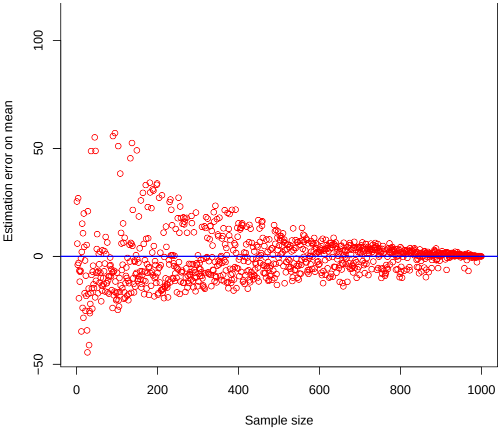
- Sparse here means that the values are spread over a long interval (not to be mixed with the usual definition of sparseness)





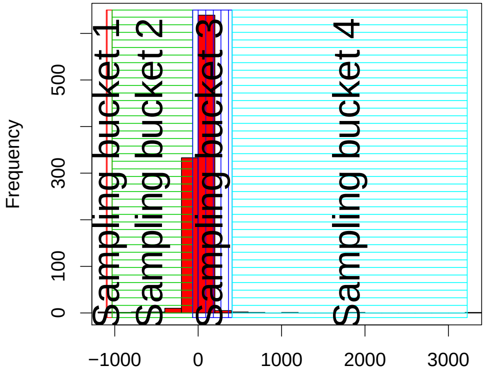
How badly does uniform sampling work for sparse datasets?

- Pretty badly, plot shows estimation errors in red, blue line is 0



Stratified sampling

- Idea is to split the dataset into buckets and ensure each is represented in the sample
- This way we can have outliers (which are disproportionaly important for estimates) represented



Using a stratified sample

- Suppose we have k buckets, and we sample N_i points from bucket i.
- An estimator for sample mean is then

$$\frac{N_1 \bar{X}_1 + N_2 \bar{X}_2 + \cdots + N_k \bar{X}_k}{\sum_{i=1}^{k} N_i}$$

Choosing a good sampling scheme

- In the previous toy-example, we used just some buckets that looked good with respect to the distribution
- How to do this systematically?
- How to reason about the behaviour of the resulting estimates?

Hoeffding equation

- Let us fix and error bound ϵ, which is the distance of the true value and estimate
- Also, set δ to be the probability that ϵ is greater than some t
- $t = (b-a)\sqrt{\frac{1}{2n}\log\frac{2}{1-\delta}}$
- Where a is minimum of the dataset and b is the maximum.
- Dependence of the range of the data is intutive
- We can solve for the sample required for achieving some error bound.

Finding good buckets

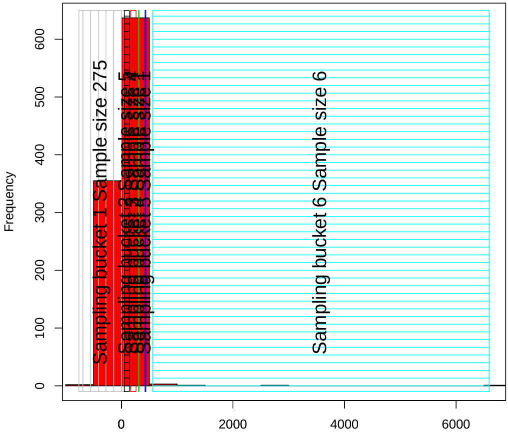
- Finding the optimal buckets: split the dataset into parts each of which is characterized by the error, and minimize the sample size (think rod cutting!)
- Total error is obtained as weighted average of errors over all buckets: $\sum_{i=1}^{K} N_i \epsilon_i / \sum_{i=1}^{K} N_i$
- Unfortunately the dynamic programming solution runs in $O(N^4)$ and is of no practical relevance for big data (and of dubious relevance for any purpose)
- A more practical solution runs in $O(N \log N)$, and in practice can deliver good results

The algorithm

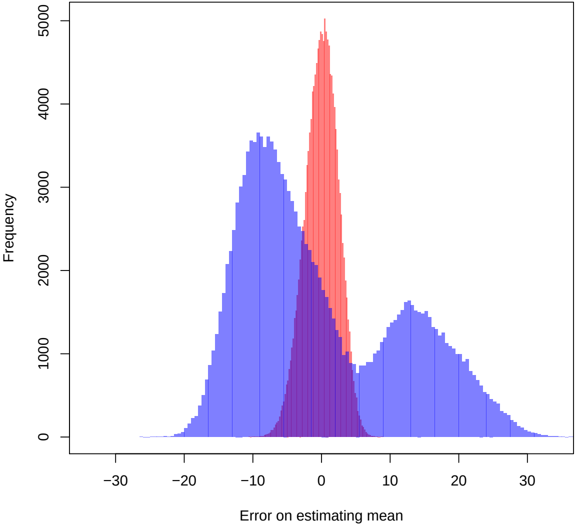
- Choosing optimal error for each bucket separately is difficult, so instead we fix an error bound ϵ_0.
- Now clearly the total error equals the maximum error for each bucket
- A greedy approximation of the optimal scheme: traverse the data set, and at each step see if it seems better to add the element to the previous bucket or to create a new bucket with that element alone.
- Never worse than uniform sampling, oftentimes better even if not optimal

Resulting buckets

- The same dataset, using my own R implementation with $\delta = 0.9$ and $\epsilon = 5$



Comparison on 150 000 samples of 300



Inserting new records I

- Suppose we want to append a new record to the data
- Often it will be possible to avoid computing everything from a scratch
- Algorithm: find the bucket to which new record belongs to
- Compute the new error bound, taking into account the updated range and sample size (of the bucket)
- If the global error bound requirement is still satisfied, no computation will be needed
- Record is added to the bucket according to reservoir sampling
- We can be sure of satisfying global error requirement, but optimality of is not considered. Then again, our algorithm is approximation in any case

Inserting new records II

- It is possible that the record is outside of the range of the bucket
- In this case only possibility is creating a new bucket with that record alone
- If this happens very often, all will be lost

Conclusions

- Nice method, but limitations are severe
- Notice that computing the mean or sum of any data is $O(N)$. Now we are sampling with an $O(N \log N)$ method, what sense might that make?
  - If samples can somehow be reused in many queries, this can still be worth it
  - Or if samples can be incrementally updated this might be useful
- Methods for incrementing the sample are relatively easy
- Also notice that the method used theoretical error bound results established for sum of random variables (and thus the mean), but how about arbitrary functions?
- In any case, certainly no silver bullet for handling big data