Comp 480/580: Assignment #2

Rice University — Due Date: Tuesday, 03/03/2020

1 Count-Min Sketches Or Count-Sketches

Task: We will implement Count-Min Sketches, Count-Sketches, and compare them. The goal is to find most frequent words in a search log using the sketches and compare them with the exact counts.

Implementation: We have to implement three things 1.) Count-Min Sketch. 2) Count-Sketch and 3) A plain dictionary to keep track of exact counts for evaluation.

1.1 Data and Processing

- Assignment 1 will come handy; we will use the same AOL dataset. So your codes and hash functions written in Assignment 1 will be directly useful.
- Download Aol dataset, which includes anonymized user ids and clicks data. **(url)**
- Data prepossessing: This time, we will only use query keywords from this file. So if the keywords are "pop up ads," then we treat it as three words, and we will insert all the three words into our sketch, with count 1 each.
- For all the keywords, we will insert it into our sketch and dictionary one by one.

Sketches: For sketches, we will implement insert and query functionality, which will insert a token in the sketch, which is simply adding "1" to the count, every-time a token is observed (Streaming model). The query functionality will return an estimate of the frequency (or total count) of a given token. The sketches as two parameters, 1.) d, which is the number of the hash functions, and 2) R, which is the range of the hash functions. For this assignment, we will keep d to be fixed at value 4. We will experiment with the five values of R $\{2^{10},\ 2^{12},\ 2^{14},\ 2^{16},\ 2^{18}\}$

Dictionary: Keep a standard dictionary of tokens and count. Implement it in any convenient way you like. (this might be heavy on memory, so be careful. There are around 4 million unique tokens in the dataset.)

Frequent, Random, and Infrequent Tokens: Using the dictionary, find 100 most frequent tokens, call it *Freq-100* along with their counts. Similarly, find 100 most infrequent tokens (*Infreq-100*) and get their counts. Also, get 100 random tokens from the dictionary *Rand-100* along with their counts. Is it OK if *Rand-100* has any overlap with *Freq-100* or *Infreq-100*. Note the space used by the dictionary.

Plots the errors: Now for every value of range R, we will generate three plots of errors, one each for *Freq-100*, *Rand-100* and *Infreq-100*. The x-axis is the 100 words (sorted based on their frequency. The y-axis is the mean square error defined as $(\hat{C}_i - C_i)^2$, where $\hat{C}_i$ is the estimated count from the sketch and C_i is the true count from the dictionary. In each of the plot, we will have two lines, one for the Count-Min Sketch and the other for the Count-Sketch. Thus in total, we have 3 x 5 = 15 plots.

Sketches with Heaps: Finally, add the functionality of returning the identity of approximate top-1000 items from your sketches. We will maintain a little dictionary of top-1000 along with their counts in a min-heap. For each of the sketches (count-min and count-sketch), while updating the count of a token, estimate the token's total count so far and check it with the smallest value in the little dictionary. If the token is already among the top 1000, update the count. If not, then check with the minimum count and update if found better. After all the insertions, report the top-1000 elements. Plot the size of the intersection of top-1000 from the sketch with the actual top-100 from the dictionary (Yes, we report 1000 approximate frequent and see how many of 100 most freqent did we identify). The plot will have the x-axis as the values of R and y-axis as the size of intersection. We need one plot with two lines, one for count-min and other for count-sketch.

A Flaw: The above method for reporting top-1000 is approximate, which is fine. However, there is a severe flaw when we are using count-sketches, where we allow deletions (or negative updates). What is the Flaw, and make a simple case where it will hurt finding the identity of the frequent elements reported?

1.2 What to Submit:

In about a page report, describe your findings and write conclusions.

Submit codes, plots, report, makefile (if any), and the main script to run the code that generates the plots. Submit one compressed file via Canvas. We will only type the texts in the command line, and it should generate all the outputs. Make sure you fix the random seeds so that multiple runs of the code produces the same output.