Advanced Algorithms Assignment 3

Diogo Bento 93391

Resumo – **O presente artigo detalha um estudo do uso de sketches no contexto da Unidade Curricular de Algoritmos Avançados da Universidade de Aveiro.**

Este artigo faz uso de CountMinSketch e minHeap e analisa os resultados obtidos, que serão comparados aos valores obtidos por um contador normal.

Abstract – **This paper details a study of the use sketches, tackled within the context of the Advanced Algorithms Curricular Unit of Universidade de Aveiro.**

This article uses CountMinSketch with minHeap and analyses the results obtained, using values obtained by a normal counter as a benchmark.

I. INTRODUCTION

In software counting event occurrences is commonplace.

Standard procedure is to increment the event's matching variable by 1 whenever the event occurs, but other alternatives may be appealing based on context.

Sketch algorithms are an alternative that does not store what values are being counted to save memory and ultimately results in a lossy, compact synopsis of data.
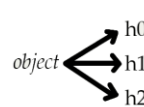
In this project a bounded-size minheap was used as a support structure to the Sketch itself in order to store the most frequent items in a given text.

II. COUNTMINSKETCH

The CountMinSketch algorithm is backed by an integer table with dimensions **d** x **m**, initialized to 0.

When a value is counted using this algorithm **d** hash functions with values ranging from 0 to **m**-1 are calculated.

The table's **d** rows are then incexemented by 1 at the indexes returned by their corresponding hash function.

Fig. 1 – Sketch Count Update

As insertion count increases the probability of a object's matching buckets being correct goes down due to collisions, however due to how this algorithm works each bucket can only contain values equal to or greater than the correct value.

As such out of the **d** values related to a given object it is known that the one with the smallest value has the least error.

Fig. 2 – Sketch Value Retrieval

III. ANALYSIS

A. On obtaining data and visualizations

In order to obtain data the German, French, and English versions of 'A Christmas Carol' were downloaded from Project Gutenberg.

The texts had their headers and footers manually removed following conversion to lowercase and stopword/non-alphabetic character removal.

Sketches were performed with 3 to 10 hash functions and hash value ranges varying from 10 to 200% of the unique word count for every language.

The maximum and average errors were then calculated, both on absolute and relative terms, as well as the deviation and these results were plotted as 3D scatter plots.

Additionally a 100-slot minheap was created for each of these parameter combinations, allowing for approximate identification of the 100 most common terms in a given file without explicitly knowing the document word set.

The values contained in the final heap snapshots are different from those present in the sketch snapshot as they were obtained while streaming data rather than at the end of the streaming process.

Heap snapshot values are equal to or lower than the values contained in the sketch due to having been obtained at a time when less collisions had happened and therefore have less error, but the heap is biased towards including terms seen more recently and excluding older terms due to the amount of frequency-boosting collisions going up over time.

The obtained results can be found at *results.json* and were obtained by running *counters.py*.

The statistical calculations and plotting code are contained in *analysis.py* and running the program with the --**interactive** flag allows a user to interact with graphs instead of outputting them into an image file.

B. Additional Script

An additional script (*frequent.py*) was written to analyze minheap + Sketch algorithm performance but it is not used for any plotting.

This script calculates Precision, Recall, and F-Score when using a bounded minHeap for a given scenario that is provided via command-line arguments.

The list of arguments is as follows:

- **--file**: text source, defaults to "english.txt"
- **--hashes**: number of hash functions, default is 5
- **--slots**: hash function range, default is 700
- **--heapsize**:size of minHeap, defaults to 100
- **--cutoff**: floating point value between 0 and 100 used to determine what terms are worth remembering and comparing to those in the minHeap. Given a cutoff value of value of 50% the program will consider the most common words in descending order until it has accounted for 50% of all words in the original text. Defaults to 50.0.

While this script's metrics were not included in the main analysis, partly due to the fact that adding further for loops would be costly, the data obtained from running this script at varying parameters was informative.

C. Results

For brevity's sake only graphs related to the English text will be displayed.

All graphs will be provided in the */graphs* subdirectory of this report.

The Maximum Absolute Error was generally unpredictable across languagues.

Increasing the amount of hash functions seems to be consistently helpful while an increased hash range benefits greatly at first but then falls off in usefulness.
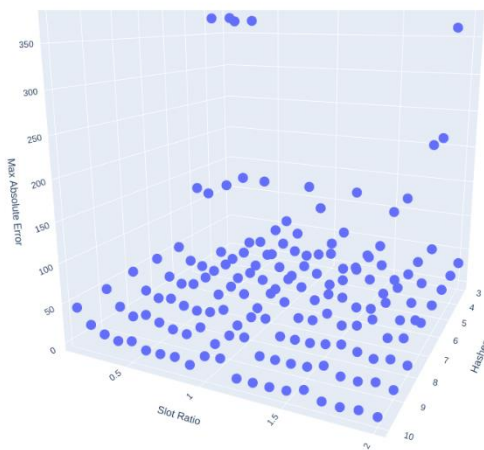

Fig. 3 – Maximum Absolute Error

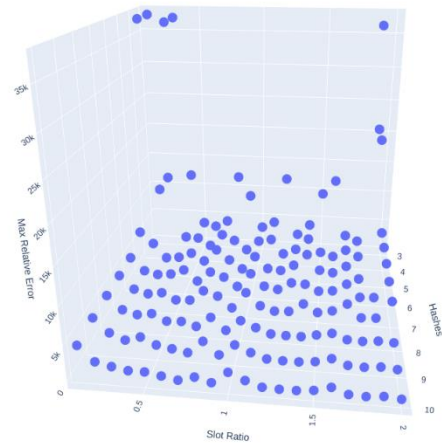Analyzing Maximum Relative Error yields similar results.

Fig. 4 - Maximum Relative Error

The Average Absolute Error behaves similarly for all languages.

The error value seems to display early exponential behaviour as the number of hash functions diminuishes.

Similarly increasing the number of hash buckets decreases error greatly roughly until ratio 0.5.
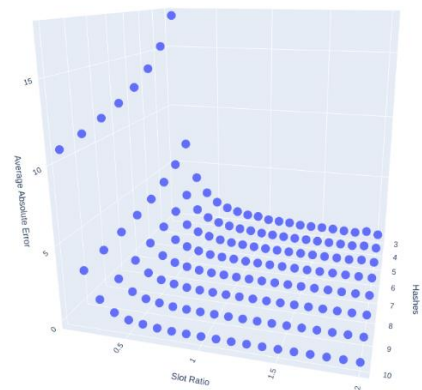

Fig. 5 - Average Absolute Error

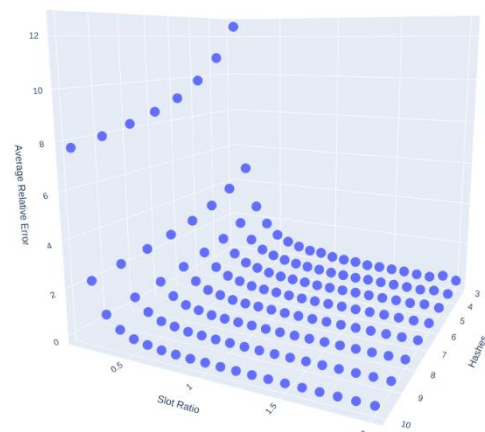The Average Relative error's behaviour is similar.

Fig. 6 - Average Relative Error

Deviation behaves similarly to Average Error, with the distinction that when using a low hash function count it is significantly more unstable, even as the size of each table increased.
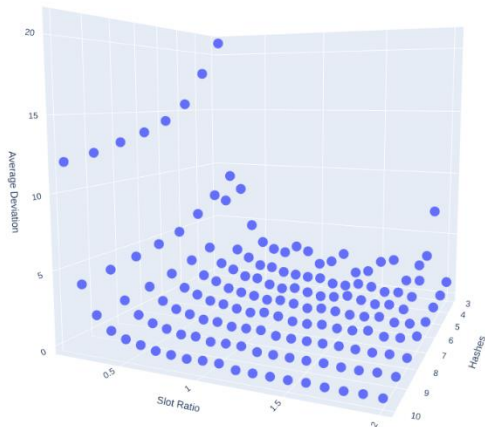

Fig. 7 - Average Deviation

When looking at the estimated 100 most frequent words there were 2 metrics that came to mind:

- Overlap between this word set and the true top 100

- How many of the words are in the correct rank

While the amount of hash functions used provides some measure of improvement, increasing the total range that said function maps to is seemingly much more helpful, especially at first.

Even at lower parameter values overlaps of over 90% can be seen. Any language has a significant fraction of datapoints at 98% overlap or above but English is the only one that reaches 100% overlap, and does so consistently.

Overall all languages behaved very similarly in this metric.
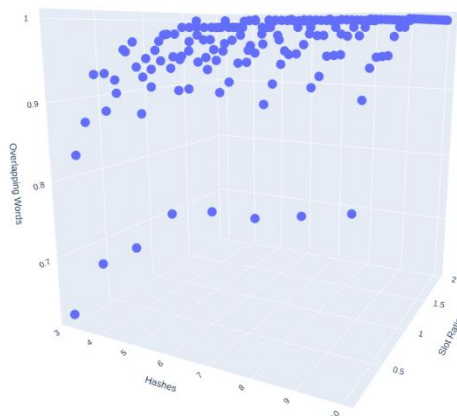

Fig. 8 - Fraction of set overlap

When it comes to exact matches values are generally much lower at around 50% matches, but the French text shows values of up to 55%, thought not very consistently.

For this metric tweaking hash function range does not seem to produce the same significant initial benefit as seen in overlap, while tweaking the amount of hash functions has negligible benefits if any at all.
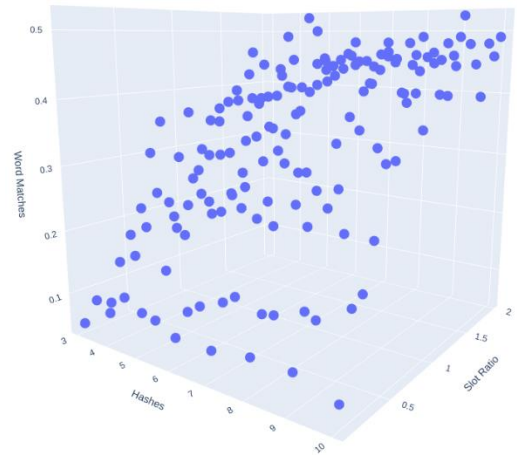

Fig. 8 - Fraction of exact matches