COMP90056 Stream Computing and Applications
Assignment 2 (15 marks) — Frequent Items in a Data Stream
Posted date: Monday 5 October 2020
Due date: 17:00 Monday 26 October 2020

Background

For this assignment you will investigate the performance of counter-based algorithms that estimate the most frequent items of a data stream. These are known as StickySampling, LossyCounting and SpaceSaving, described in [1] and [2]. Similar to Assignment 1, you will summarise your findings and provide recommendations in a report.

Task 1: Implementation

There is no specified programming language for this assignment. You may use whichever programming language you wish.

Implement each of the data sources, algorithms and performance metrics below.

- A stream of integers as input for the algorithms with frequencies that obey a Zipf (power-law) distribution. Here the i^{th} most frequent item has probability *proportional* to $1/i^z$, where z is a positive real-valued parameter. Note that the most frequent items could appear anywhere in the stream, not only at or near the beginning.
- The StickySampling algorithm, according to Section 4.1 of [1]. This requires three input parameters: support s (proportion of items that defines an item as frequent), error tolerance ϵ, and probability of failure δ.
- The LossyCounting algorithm, according to Section 4.2 of [1]. This requires two input parameters: support s, and error tolerance ϵ.
- The SpaceSaving algorithm, according to Fig. 1 of [2] (or lecture notes). This requires a number of counters t as input parameter ($t = 1/s$).
- Assume $\epsilon = s/10$ in your implementations that depend on ϵ.
- You may use a hash map or equivalent structure to store the tracked items.
- Define the *precision* of an estimate to be the proportion of tracked items that are actually frequent (have frequency greater than sN where N is the stream size).
- Define the relative error E_{rel} of a frequency estimate $\hat{f}$ compared to its true value f as $|\hat{f} - f|/f$.
- Compute the speed at which the *frequent items* problem is solved for each algorithm.
- Compute the number of items of the stream that are tracked at any point in the StickySampling and LossyCounting algorithms.

Task 2: Report

Complete a short report analysing the three algorithms including the following points. For any randomised algorithms, steps will need to be taken to make results more definitive. Plots should be well described and clear to visualise. Refer to Section 4 of [3] for a good example of analysis including plots to describe the performance of algorithms (your report is not expected to be as detailed). Avoid solely answering the points listed as if you were making entries in an experimental logbook. Instead, it should read as a **technical report** with appropriate subheadings.

- *Set-up*—A description of the experimental set-up to aid reproducibility. For example, include the CPU frequency, memory, operating system, programming language, compiler (if applicable). **[1 mark]**
- Generate input data according to the power-law distribution and plot the distribution of frequencies of items for $z = 1.1, 1.4, 1.7, 2.0$ with a logarithmic plot to demonstrate that they are as desired. In these cases, how many items have frequency at least 1% of the total number of items? **[2 marks]**

Table 1: Theoretical Performance of the Algorithms

	StickySampling	LossyCounting	SpaceSaving
Update time			
Memory			
Accuracy			

- Describe in your own words (one paragraph) the intuition behind how the StickySampling algorithm identifies frequent items. Cite any references used apart from [1]. **[1 mark]**
- Complete Table 1 with theoretical values (in big-O notation) as a function of algorithm parameters. Provide some justification of these formulas (they may depend on your implementation) and point out any assumptions made. This will be used as a reference to compare with results you obtain in the practical implementation. **[1 mark]**
- Evaluate the impact of different values of z (listed above) on the performance of the three algorithms. Fix $s = 0.001$ and choose an appropriately large stream length. Plot precision vs skew and runtime vs skew, commenting on your results. **[2 marks]**
- Evaluate the influence of support on memory and runtime for each of the three algorithms. Plot the maximum number of tracked items vs support and also the runtime vs support (minimum s at least 10^{-4}). Using these, obtain a third plot of the maximum number of tracked items vs runtime for all three algorithms. Point out where the algorithms differ and possible reasons for this. **[3 marks]**
- Plot average relative error vs support for an appropriate choice of parameters for the three algorithms commenting on your results. **[2 marks]**
- *Practical considerations*—point out any differences between practice and theory when running the algorithms, and possible explanations for the difference. **[1 mark]**
- *Conclusion and recommendation*—which would be your recommended algorithm for practical use? Indicate how your answer depends on the application (e.g., speed, accuracy or memory requirements). What additional experiments would you suggest running to gain further insight? **[2 marks]**

Marking Scheme

The marks for each component in the report are as shown above. The grading will be based on the clarity and rigour of your description or analysis for each part. The code will not be assessed, but should be presented and documented well enough that others could reproduce the whole experimental study of your report and obtain similar results.

Submissions

You should lodge your submission for Assignment 2 via the LMS (i.e., Canvas). You must identify yourself in each of your source files and the report. Poor-quality scans of solutions written or printed on paper will not be accepted. Solutions generated directly on a computer are of course acceptable. Submit two files:

- A report.pdf file comprising your report for the experimental study.
- A code.zip file containing all your source files of the implementations for the experiments.

Do not include the testing files, as these might be large. REPEAT: DO NOT INCLUDE TESTING FILES! It is very important, so that you can justify ownership of your work, that you detail your contributions in comments in your code, and in your report.

Administrative Matters

Late Work

The late penalty for non-exam assessment is **two marks per day (or part thereof) overdue**. Requests for extensions or adjustment must follow the University policy (the Melbourne School of Engineering "owns" this subject), including the requirement for appropriate evidence. Late submissions should also be lodged via the LMS, but, as a courtesy, please also email Chaitanya Rao when you submit late. If you make both on-time and late submissions, please consult the subject instructor as soon as possible to determine which submission will be assessed.

Individual work

You are reminded that your submission for this Assignment is to be your own individual work. Students are expected to be familiar with and to observe the University's Academic Integrity policy `http://academicintegrity.unimelb.edu.au/`. For the purpose of ensuring academic integrity, every submission attempt by a student may be inspected, regardless of the number of attempts made. Students who allow other students access to their work run the risk of also being penalised, even if they themselves are sole authors of the submission in question. By submitting your work electronically, you are declaring that this is your own work. Automated similarity checking software may be used to compare submissions.

You may re-use code provided by the teaching staff, and you may refer to resources on the Web or in published or similar sources. Indeed, you are encouraged to read beyond the standard teaching materials. However, all such sources must be cited fully and, apart from code provided by the teaching staff, you must not copy code.

Finally

Despite all these stern words, we are here to help! Frequently asked questions about the Assignment will be answered in the LMS discussion group.

References

[1] Manku, G. and Motwani, R. Approximate frequency counts over data streams. VLDB, pp. 346–357, 2002.

[2] Metwally, A., Agrawal, D., and El Abbadi, A. *Efficient computation of frequent and top-k elements in data streams.* Database Theory, ICDT 2005, 10th International Conference, Edinburgh, UK, January 5–7, 2005, pp. 398–412, 2005.

[3] Luo, G., Wang, L., Yi, K., and Cormode, G. *Quantiles over data streams: experimental comparisons, new analyses, and further improvements.* The VLDB Journal vol. 25, pp. 449–472, 2016.

COMP90056 team
October 2020