Finding Frequent n-grams in Amazon Reviews using Lossy Counting Algorithm

Jiali Lu
jialilu.physics@gmail.com

1. Introduction

N-grams are useful in natural language processing (NLP) tasks. For example, extending word2vec to n-grams can further improve the downstream text classification accuracy [1]. However, working with n-grams can be memory intensive. Here I show how Lossy Counting [2] can be used to find the frequent n-grams and their frequencies.

2. Setup

For this study, I use the review texts from the training set of the Amazon polarity dataset. There are 3.6 million reviews and 276 million words in total. As for finding the frequent n-grams, I consider 5-gram and choose a frequency threshold of $s = 1 \times 10^{-6}$.

As for the lossy counter, I choose the error bound $\epsilon = s/2$. In my implementation, I further improved the speed and accuracy of lossy counter by introducing a cache counter. I specify $\text{flush\_limit} = 5/\epsilon$ for the cache counter.

3. Results

3.1. Memory Footprint

The lossy counting algorithm reduces memory footprint from nearly 32GB to 1GB. This is illustrated in Fig 1. As more reviews are processed, the memory usage in the exact counter keeps increasing. (The small jumps in the curve are due to the size doubling of the hash table behind the counter.) On the other hand, the memory usage of lossy counter periodically increase (due to normal counting) and then decrease (due to pruning of rare n-grams). The pruning keeps the memory usage of lossy counter no more than 1GB, while the exact counter claims nearly all 32GB of available memory.

3.2. Accuracy

The lossy counting provides remarkably accurate result. For the 4567 frequent items above threshold $s = 1 \times 10^{-6}$, the lossy counting algorithm had 0 false positive and 10 false negatives, achieving a F1-score of 0.9989. The margin between the counts of the 10 missed 5-grams and threshold is tiny – no more than 3. In comparison, the standard deviations of their counts are around 15.

The lossy counting algorithm also provides very accurate approximation of the frequencies. As seen in Fig 2, for

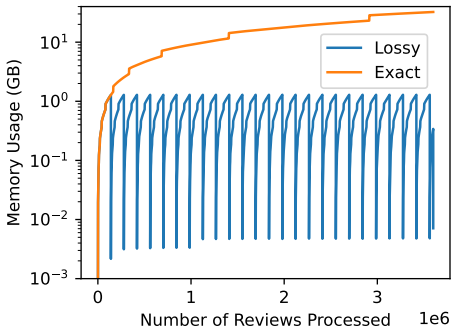

Figure 1. Memory usage by lossy counter and exact counter. Lossy counter periodically removes rare n-grams from counting and is more memory efficient.

5-grams with true frequency higher than the threshold s (marked by the dashed line on the right), the error in frequency estimation is negligible. The error only becomes significant whe the true frequency is around $\epsilon = s/2$.

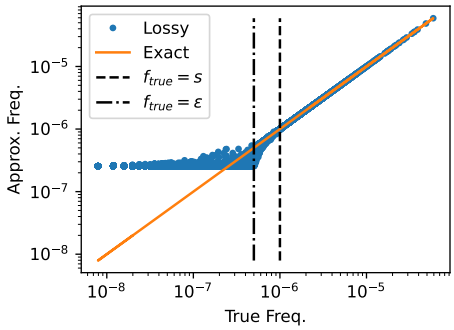

Figure 2. Approximate frequency by lossy counter versus the true frequency. The error is negligible for n-gram with frequency above the threshold s.

3.3. Running Time

While being memory efficient and remarkably accurate, my implementation of lossy counting is also very fast. Compared to the 275 seconds running time by the exact counter (using the efficient CPython implementation of Counter object from collections module), the running time of my lossy counter is only 301 seconds. This translates to an overhead of 26 second, or 10% of the baseline running time.

4. Conclusion

Lossy counting is a memory efficient algorithm for finding frequent items. When applied to finding frequent 5-grams in Amazon reviews, it reduces memory footprint from nearly 32GB to 1GB, while achieving near perfect accuracy and introducing minimal overhead.

References

[1] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759.

[2] Manku, G. S., & Motwani, R. (2002, January). Approximate frequency counts over data streams. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases (pp. 346-357). Morgan Kaufmann.