Lossy Conservative Update sketch*

Amit Goyal and Hal Daumé III
Dept. of Computer Science, University of Maryland, College Park, MD 20742
{amit,hal}@umiacs.umd.edu

Count-Min (CM) sketch [2] is a compact summary data structure to store the frequencies of all items in the input stream. It uses hashing to map frequency of an item onto a small sketch without explicitly storing the item itself. CM sketch is a two-dimensional array with width w and depth d. The user chosen parameters ε and δ determines the width and depth of the two-dimensional array: w=$\frac{2}{\varepsilon}$ and d=$\log(\frac{1}{\delta})$. ε controls the amount of tolerable error in the returned count and δ controls the probability with which the returned count is not within this acceptable error. The depth d denotes the number of pairwise-independent hash functions. Each of these hash functions h_k:$\{x_1 \ldots x_N\} \rightarrow \{1 \ldots w\}$, $1 \leq k \leq d$, takes an item from the input stream and maps it into a counter indexed by the corresponding hash function. For example, $h_2(x) = 5$ indicates that the item "x" is mapped to the 5^{th} position in the second row of the sketch. The space used is $O(wd)$ that is sub-linear in the size of input.

Update Procedure: When a new item "x" with count c, the sketch is updated by: $\forall 1 \leq k \leq d$

$$sketch[k, h_k(x)] \leftarrow sketch[k, h_k(x)] + c$$

Query Procedure: Since multiple items can be hashed to the same position, the stored frequency in any one row is guaranteed to *overestimate* the true count. Thus, to answer the point query, we return the minimum over all the positions indexed by the k hash functions. The answer to Query(x) is: $\hat{c} = \min_k \ sketch[k, h_k(x)]$.

There is a variant of CM sketch with conservative update (CU sketch) [2, 5] that is similar to CM sketch except the update operation. It is based on the idea of conservative update [3] introduced in the context of networking.

Lossy Conservative Update (LCU) sketch Prior work using CU sketch [5] shows that this method is more prone to over-estimation error on low-frequency items. However, in many Natural Language Processing (NLP) applications, items frequently follow a Zipfian distribution, and hence making an error on low-frequent items can be problematic. In addition, if counts of low-frequent items are over-estimated, their association scores like Pointwise Mutual Information (PMI) becomes high too as PMI is sensitive to low-frequency item counts [1].

To overcome this problem, we combine the idea of lossy counting [7] on top of CU (LCU) sketch. We conceptually divide the stream into *windows*, each containing $1/\gamma$ items. We fix γ such that size of each window is equal to size of the sketch i.e. $O(wd)$. In future, we will explore other settings of γ. Note that there are γN windows. We propose four different techniques of decrementing certain counts at window boundaries:

- LCU-ALL: At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($sketch[i, j] > 0$), then $sketch[i, j] = sketch[i, j] - 1$. The only difference is in CU sketch, we do not have items stored explicitly. Hence, we are decrementing the whole sketch itself rather than explicitly decrementing the counts of all items by 1. Intuitively, over here we are only storing the frequent items into the sketch.

- LCU-1: At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($sketch[i, j] > 0$ and $sketch[i, j] <= 1$), then $sketch[i, j] = sketch[i, j] - 1$. Intuitively, in this approach, we are trying to reduce some error over low-frequent counts by turning the count of 1's to zero at the window boundary.

- LCU-WS: At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($sketch[i, j] > 0$ and $sketch[i, j] <= windowIdX$), then $sketch[i, j] = sketch[i, j] - 1$. In this technique, we do not want to decrement the frequent counts. Hence, we decrement only those counts by 1 which are less than or equal to current window index.

- LCU-SWS: Here we decrement the counts of sketch more conservatively. We only decrement counts if they are less than or equal to ceil value of square root of current window index. At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($sketch[i, j] > 0$ and $sketch[i, j] <= \lceil \sqrt{windowIdX} \rceil$), then $sketch[i, j] = sketch[i, j] - 1$.

*Topic: other applications and Preference: oral

The motivation behind using these techniques is that by reducing certain counts in the sketch, we can reduce the over-estimation error in low-frequency items without incurring large under-estimation error on mid, and high frequency counts. We conjecture that all the good properties of CU sketch holds. In addition for all approaches, all reported frequencies $\hat{f}$ will have both under and over estimation error: $f - \gamma N \leq \hat{f} \leq f + \varepsilon N$.

Experiments We perform experiments to compare the errors incurred in the approximate counts of different sketches to their true counts. We use a subset of 2 million sentences (Subset) from Gigaword [6] corpus to compute the frequency of words and their contexts. The context for a given word "x" is defined as the surrounding words appearing in a window of 2 words to the left and 2 words to the right. The context words are concatenated along with their positions -2, -1, +1, and +2. We store the counts of all words (excluding numbers, and stop words), their contexts, and counts of word-context pairs in all the different sketches.

To evaluate the amount of error in approximate counts of sketches, we group all items with same true frequency into a single bucket. This grouping done based on frequency is to distinguish which sketch makes mistakes on low, mid, and high frequency items. Average Relative error is defined as the average of absolute difference between the approximated, and the true value divided by the true value over all the items in each bucket.

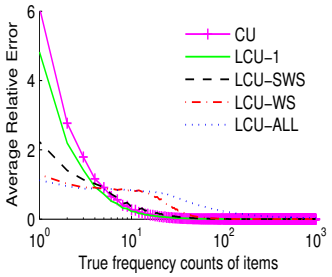

Figure 1: 10 million counter models with different decrement count strategies with fixed depth=3.

In Figure 1, first we fix the size of sketches to 10 million ($10M$) counters with four bytes of memory per each counter (thus it only requires 40 MB of main memory). We use $depth$ of 3, and $width$ $\frac{10M}{3}$ as it performs the best for CU sketch [5], and our goal is to show that using LCU is better than the CU sketch. We can make several key observations from Figure 1: • Using the simple LCU-1 over CU: reduces the error slightly over the low-frequency items. • However, if we use LCU-SWS over CU and LCU-1, the error over low-frequency items is reduced by a significant factor of at most 3. • If we move to LCU-WS over LCU-SWS, the error over low-frequency items is again further reduced by a significant factor of at most 2. However, this reduction comes at the expense of generating some small error on mid-frequency counts. • LCU-ALL has similar errors like LCU-WS but they are bigger than LCU-WS. • To summarize, overall LCU-SWS, and LCU-WS perform better than CU, LCU-1, and LCU-ALL. In addition, both have errors over different range of counts.

Applications LCU can useful in many NLP and Machine Learning (ML) problems where having large amounts of data is beneficial. In NLP, for large scale language modeling [4] counting unique n-grams, and for large scale Noun similarity [8] counting unique pairs of words, and their contexts. In ML, this idea can be used with Hash kernels [9] for dimensionality reduction, and feature selection for reducing dimensionality of large feature-sets.

References

[1] K. Church and P. Hanks. Word Association Norms, Mutual Information and Lexicography. In *Proceedings of ACL*.

[2] Graham Cormode. Encyclopedia entry on 'Count-Min Sketch'. In *Encyclopedia of Database Systems*, pages 511–516. Springer, 2009.

[3] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *SIGCOMM*, 32(4), 2002.

[4] Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. Streaming for large scale NLP: Language modeling. In *NAACL*, 2009.

[5] Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. Sketching techniques for Large Scale NLP. In *6th WAC Workshop at NAACL-HLT*, 2010.

[6] D. Graff. English Gigaword. Linguistic Data Consortium, Philadelphia, PA, January 2003.

[7] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[8] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of EMNLP*, 2009.

[9] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, December 2009.