Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

Lossy Conservative Update (LCU) Sketch: Succinct Approximate Count Storage

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

Abstract

In this paper, we propose a variant of the conservative-update Count-Min sketch to further *reduce the over-estimation error* incurred. Inspired by ideas from lossy counting, we divide a stream of items into multiple windows, and decrement certain counts in the sketch at window boundaries. We refer to this approach as a lossy conservative update (LCU). The reduction in over-estimation error of counts comes at the cost of introducing under-estimation error in counts. However, in our intrinsic evaluations, we show that the reduction in over-estimation is much greater than the under-estimation error introduced by our method LCU. We apply our LCU framework to scale distributional similarity computations to web-scale corpora. We show that this technique is more efficient in terms of memory, and time, and more robust than conservative update with Count-Min (CU) sketch on this task.

Introduction

Many Natural Language Processing (NLP) problems (Ravichandran, Pantel, and Hovy 2005; Brants et al. 2007; Turney 2008) have benefited from having large amounts of data. Many of these problems boil down to counting items. For example, in large scale language modeling (Brants et al. 2007; Goyal, Daumé III, and Venkatasubramanian 2009), we are interested in counting the number of unique n-grams. In large scale Word similarity (Agirre et al. 2009; Pantel et al. 2009), we are interested in frequency of pairs of words, and their contexts. Most of the current approaches employ data structures that reside in main memory and thus do not scale well to large corpora. Hence, this has motivated the use of clusters (Brants et al. 2007; Pantel et al. 2009) as well as streaming, randomized, and approximate algorithms (Ravichandran, Pantel, and Hovy 2005; Goyal et al. 2010b; Van Durme and Lall 2010) to handle large amounts of data.

In this work, we focus on use of streaming approaches especially conservative update with Count-Min sketch (Cormode and Muthukrishnan 2004). Unfortunately prior work using CU sketch shows that this method is more prone to over-estimation error on low-frequency items (Cormode and Hadjieleftheriou 2008; Goyal et al. 2010b). However, in most of the NLP applications, items frequently follow a Zipfian distribution, and hence making an error on low-frequency items can be devastating. In addition, if the counts of low-frequency items are over-estimated, their association scores like Pointwise Mutual Information (PMI) becomes high too as PMI is sensitive to low-frequency item counts (Church and Hanks 1989).

To overcome this problem, we combine the idea of *lossy counting* (Manku and Motwani 2002) on top of CU sketch. We refer to this approach as a lossy conservative update (LCU). We propose 4 variants of LCU that differ in how aggressively they discount items at window boundaries. The one which is most conservative only reduces the error slightly over the low-frequency items. Once, we gradually make it more aggressive, the error of low-frequency items decrease significantly. However, once it becomes more aggressive, it only trades off error between low and mid frequency items.

LCU framework can be useful for many data stream problems in AI. For example, keeping tracking of queries posed to an Internet search engine, tracking the collection of transactions across all branches of a supermarket chain, phone conversations, and ATM transactions. Moreover, in many AI problems, we are interested in storing frequent items (not interested in rare items); e.g. frequent n-grams in language modeling (Goyal, Daumé III, and Venkatasubramanian 2009), and this framework can be applied to store counts of frequent items efficiently in *succinct* space. In this paper, we apply our LCU framework to a NLP problem that is scaling distributional similarity computations to web-scale corpora. We show that this technique is more efficient in terms of memory, and time, and more robust than conservative update with Count-Min (CU) sketch on this task.

Background

Distributional Similarity

Distributional similarity is based on the distributional hypothesis that similar terms appear in similar contexts (Firth 1968; Harris 1954). The context vector for each term is represented by the strength of association between the term and each of the lexical, semantic, syntactic, and/or dependency units that co-occur with it. We use pointwise mutual information to compute association measure between the term

and each of the context to generate the context vector. Once, we have context vectors for each of the terms, cosine similarity measure returns distributional similarity between terms.

The pointwise mutual information (PMI) (Church and Hanks 1989) between a word "w" and its context "c" is:

$$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

Where, $P(w,c)$ is the likelihood that w and c occur together, and $P(w)$ and $P(c)$ are their independent likelihoods.

Streaming

A sketch is a compact summary data structure to store the frequencies of all items in the input stream. Sketching techniques use hashing to map items in streaming data onto a small sketch vector that can be updated and queried in constant time. These techniques generally process the input stream in one direction, say from left to right, without re-processing previous input. The main advantage of using these techniques is that they require a storage which is sublinear in size of the input stream. The following surveys comprehensively review the streaming literature: (Rusu and Dobra 2007; Cormode and Hadjieleftheriou 2008).

Recently in NLP community, the streaming algorithm paradigm has become popular. It provides memory and time-efficient framework to deal with terabytes of data. For example, we (Goyal, Daumé III, and Venkatasubramanian 2009); Levenberg and Osborne (2009) build approximate language models and show their effectiveness in Statistical Machine Translation (SMT). In Van Durme and Lall (2009b), a TOMB Counter (Van Durme and Lall 2009a) was used to find the top-K verbs "y" with the highest PMI given verb "x". An efficient online framework to locality sensitive hashing (Van Durme and Lall 2010; 2011) was proposed for large-scale word clustering. Our current work is most similar to our earlier work (Goyal et al. 2010a). In our earlier work, we showed the effectiveness of CU sketch on task of distributional similarity. However, our earlier framework was sensitive to correct frequency cutoff for a corpus. In this current work, we show that using LCU is more robust to frequency cutoff as compared to CU sketch.

CM sketch

The Count-Min sketch (Cormode and Muthukrishnan 2004) with user chosen parameters (ϵ,δ) is represented by a two-dimensional array with width w and depth d. Parameters ϵ and δ control the amount of tolerable error in the returned count (ϵ) and the probability with which the returned count is not within this acceptable error (δ) respectively. These parameters determine the width and depth of the two-dimensional array. We set w=$\frac{2}{\epsilon}$, and d=$\log(\frac{1}{\delta})$. The depth d denotes the number of pairwise-independent hash functions and there exists an one-to-one mapping between the rows and the set of 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, $h_3(x) = 8$ indicates that the item "x" is mapped to the 8^{th} position in the third row of the sketch.

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

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

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$[k, h_k(x)]$.

Both update and query procedures involve evaluating d hash functions. d is generally small; e.g. in our earlier work (Goyal et al. 2010b; 2010a), we used d=3, hence update and query operations take only constant time. Moreover, all reported frequencies are within ϵN of true frequencies with probability of at least 1-δ. The space used by the algorithm is $O(wd)$ that is the size of the 2-D array.

CU sketch

The Count-Min sketch with conservative update (CU sketch) (Cormode 2009; Goyal et al. 2010b) is similar to CM sketch except the update operation. It is based on the idea of conservative update (Estan and Varghese 2002) introduced in the context of networking. It is used with CM sketch to further improve the estimate of a point query. To update an item, x with frequency c, we first compute the frequency $\hat{c}$ of this item from the existing data structure and the counts are updated according to: $\forall 1 \leq k \leq d$, $\hat{c} = \min_k$ sketch$[k, h_k(x)]$

$$\text{sketch[k,}h_k(x)\text{]} \leftarrow \max\{\text{sketch[k,}h_k(x)\text{]}, \hat{c} + \text{c}\}$$

The intuition is that, since the point query returns the minimum of all the d values, we will update a counter only if it is necessary as indicated by the above equation. This heuristic avoids the unnecessary updating of counter values and thus reduces the over-estimation error. In our earlier work (Goyal et al. 2010b), we found CU reduces the average relative error of its approximate counts by a factor of two over CM sketch.

Lossy Counting

We present a brief description of a variant of original algorithm presented by Manku in a presentation of the (Manku 2002) paper. For original lossy counting, refer (Manku and Motwani 2002). The algorithm proceeds by conceptually dividing the stream into *windows*, each containing $1/\gamma$ items [1]. Note that there are γN windows. The algorithm stores (item, count) pairs. For each item in the stream, if it is stored, then the count is incremented; otherwise, it is initialized with a count of 1. At the window boundary, the count of each item is decremented by 1, and items with a count of zero after the upgrade are deleted. At the end, when all items in the stream have been processed, the algorithm returns all items where $count \geq (s - \gamma)N$.[2]

Lossy Conservative Update (LCU) sketch

Prior work using CU sketch (Cormode and Hadjieleftheriou 2008; Goyal et al. 2010b) shows that this method is

[1] γ is a user chosen parameter which controls the size of window

[2] s is support and is another user chosen parameter.

more prone to over-estimation error on low-frequency items. However, in many NLP applications, items frequently follow a Zipfian distribution, and hence making an error on low-frequency items can be problematic. In addition, if counts of low-frequency items are over-estimated, their association scores like PMI becomes high too as PMI is sensitive to low-frequency item counts (Church and Hanks 1989).

To overcome this problem, we combine the idea of lossy counting on top of CU sketch. We refer to this approach as a lossy conservative update (LCU). We conceptually divide the stream into *windows*, each containing $1/\gamma$ items.[3] Note that there are γN windows. We propose four different techniques of decrementing certain counts at window boundaries:

- LCU-ALL: This approach is most similar to the variant of lossy counting (Manku 2002). 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-frequency counts by turning the count of 1's to zero at the window boundary.
- LCU-WS: This approach is similar to original lossy counting (Manku and Motwani 2002). 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: In this approach, 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.[4] 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 $sketch[i,j] = sketch[i,j] - 1$.

All the four above variations are heuristics on top of CU sketch. The motivation behind using these heuristics 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. In future, in another line of work, we will like to do the theoretical analysis of above approaches. We conjecture that all the good properties of CU sketch holds. In addition for all approaches, all reported frequencies $\hat{f}$ have both under and over estimation error: $f - \gamma N \leq \hat{f} \leq f + \epsilon N$. In next section, we experimentally study the Average Relative Error (ARE) over LCU counts compared to their true counts.

[3] 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.

[4] We can also use log function instead of square root. Log function is even more conservative than square root.

Intrinsic Evaluations

We perform intrinsic evaluations to compare the errors incurred in the approximate counts of different sketches to their true counts. We also experiment to find the best (width,depth) settings for LCU sketches. Finally, we show the over-estimation (OE) and under-estimation (UE) errors separately for the LCU sketches.

Data

We used the same pre-processed corpus as used in our earlier work (Goyal et al. 2010a) that is Gigaword corpus (Graff 2003) and a copy of web crawled by (Ravichandran, Pantel, and Hovy 2005). 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.[5] To compare our LCU sketches with CU sketch, we use four different sized corpora: SubSet, Gigaword (GW), GigaWord + 50% of web data (GW-WB1), and GigaWord + 100% of web data (GW-WB2). It is memory, and time intensive to perform many intrinsic evaluations on large data (Brants et al. 2007; Ravichandran, Pantel, and Hovy 2005; Goyal et al. 2010b). Hence, we use a subset of corpus of 2 million sentences (Subset) for it. Corpus Statistics are shown in Table 1.

Corpus	Sub set	GW	GW-WB1	GW-WB2
Size (GB)	.32	9.8	49	90
# of sentences (Million)	2.00	56.78	462.60	866.02
Stream Size (Billion)	.25	7.65	37.93	69.41

Table 1: Corpus Description

Comparing different sketches

To evaluate the amount of error in approximate counts of sketches to their true counts, we first 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 (ARE) 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.

$$\text{ARE} = \frac{1}{M}\sum_{i=1}^{M}\frac{|\text{True}_i - \text{Approximate}_i|}{\text{True}_i}$$

Where True, and Approximate denotes values of exact and approximate sketch counts respectively; M denotes the number of items with same counts in a bucket.

[5] From here on, words, their contexts, and word-context pairs together will be referred as items.

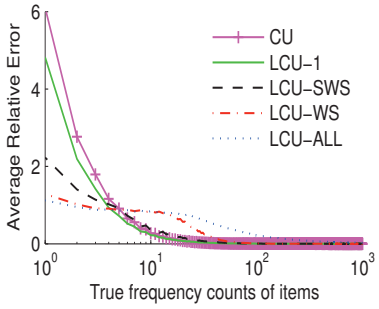

Figure 1: 10 million counter models with different decrement count strategies with fixed depth=3.

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 (Goyal et al. 2010b), 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 more than LCU-WS.

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. Hence, we will compare these sketches with respect to CU sketch on our extrinsic evaluation that is distributional similarity.

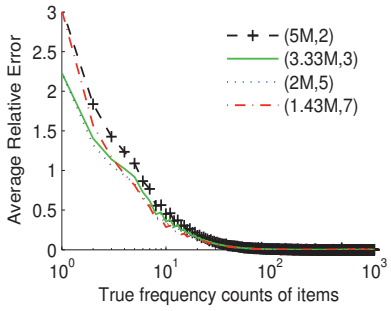

Figure 2: Comparing LCU-SWS 10 million counter models with different (width,depth) settings.

Varying (depth,width) settings

In this experiment, we will find best settings of (width,depth) for LCU-SWS, and LCU-WS. Here again, we fix the size of sketches to 10 million ($10M$) counters in Figure 2 and 3. We try different values of depth (2, 3, 5, and 7) of the sketch and in each case, the width is set to $\frac{10M}{depth}$. In Figure 2, using $depth$ of 5 gets comparatively smaller ARE compared to other settings. However, in Figure 3, using depth of 5, and 7 gets comparatively smaller ARE compared to other settings. Since using depth of 5 is faster than 7 when querying, and updating requires depth # of operations. Hence, for the rest of the paper we fix this parameter to 5.

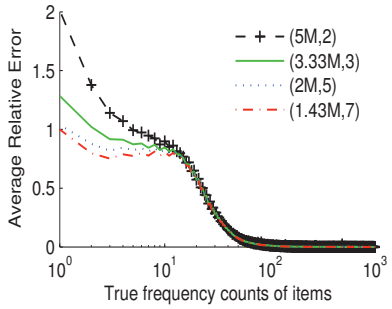

Figure 3: Comparing LCU-WS 10 million counter models with different (width,depth) settings.

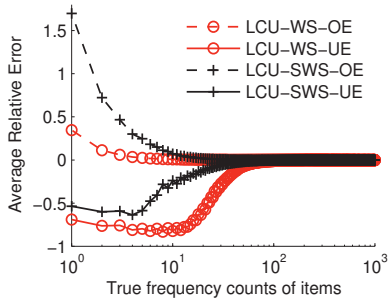

Figure 4: Over-estimation (OE), and under-estimation (UE) error for LCU-WS, and LCU-SWS for 10 million counter models.

Studying Over and Under Estimation Error

Here, we separately study the over-estimation (OE), and under-estimation (UE) error for LCU-SWS, and LCU-WS counts. To accomplish that, rather than using absolute error values, we divide the values into OE (+ve), and UE (-ve) buckets. Hence, to compute the OE ARE, we do the average of positive values over all the items in each bucket For UE, we do the average over the negative values. We fix the size of sketches to 10 million ($10M$) counters with depth 5. We can make following observations from Figure 4:

- The amount of OE ARE in LCU-WS for low-frequency items is low i.e. at most 0.4 compared to LCU-SWS's 1.6.
- However, the amount of UE over average counts for LCU-WS is larger than LCU-SWS.

Extrinsic Evaluations

We use efficient distributional similarity (Goyal et al. 2010a) as an application to show the effectiveness of LCU over CU

Data	GW ($400M$)			GW-WB1 ($2B$)			GW-WB2 ($2B$)		
Model	Frequency cutoff			Frequency cutoff			Frequency cutoff		
	10	20	40	10	20	40	10	20	40
	ρ			ρ			ρ		
WS-353									
Exact	**.36**	.32	.30	**.46**	.45	.44	**.47**	.46	.45
CU	-0.04	**.34**	.29	-0.09	-0.02	.05	-0.03	-0.03	-0.03
LCU-WS	**.34**	.30	.28	.42	**.47**	.45	-0.06	.24	**.47**
LCU-SWS	.01	**.32**	.29	.04	**.49**	.45	-0.04	-0.04	**.46**
WS-203									
Exact	**.49**	.44	.41	**.57**	.57	.56	.56	**.57**	.56
CU	-0.07	**.44**	.38	-0.08	.02	.08	-0.06	-0.06	-0.02
LCU-WS	**.43**	.39	.38	.56	**.57**	.55	-0.07	.40	**.57**
LCU-SWS	.02	**.41**	.38	.10	**.58**	.56	-0.04	-0.03	**.57**
RG-65									
Exact	.14	.09	**.15**	**.45**	.36	.23	**.47**	.44	.31
CU	-0.02	0.0	**.11**	-0.06	-0.02	.03	-0.05	-0.05	.06
LCU-WS	.01	.04	**.11**	**.58**	.38	.22	.01	**.42**	.36
LCU-SWS	-0.02	.03	**.12**	-0.13	**.47**	.23	-0.04	-0.04	**.48**
MC-30									
Exact	**.27**	.19	.18	**.63**	.53	.44	.59	**.60**	.49
CU	-0.26	.23	**.29**	-0.08	.09	.06	-0.27	-0.27	-0.22
LCU-WS	.24	.25	**.26**	.51	**.56**	.39	-0.46	.22	**.49**
LCU-SWS	-0.27	**.27**	**.27**	-0.02	**.58**	.43	-0.48	-0.48	**.53**

Table 2: Evaluating word pairs ranking with Exact and CU counts. Scores are evaluated using ρ metric.

sketch. In first step, we store counts of all items in CU, LCU-SWS, and LCU-WS sketches. In the second step, for a target word "x", we consider all words (excluding numbers, and stop words) as plausible context (since it is faster than traversing the whole corpus.) and query the sketch for such word-context pairs, and compute approximate PMI between them. We maintain only top K PMI contexts using a priority queue for every target word "x" and store them onto the disk. In the third step, we use cosine similarity using these approximate top K context vectors to compute efficient distributional similarity. The efficient distributional similarity using sketches has following advantages. • It can return semantic similarity between any word pairs that are stored in the sketch. • It can return the similarity between word pairs in time O(K). • We do not store items explicitly, so the overall space required is sub-linear in input stream. • Most of the steps can be parallelized and hence this framework can be applied in distributed setting.

Test sets

We use four test sets which consist of word pairs, and their corresponding human rankings. We generate the word pair rankings using efficient distributional similarity. We report the spearman's rank correlation coefficient (ρ) between the human and distributional similarity rankings.[6] The four test sets are: • **WS-353** (Finkelstein et al. 2002) is a set of 353 word pairs. • **WS-203**: A subset of WS-353 containing 203 word pairs marked according to similarity[7] (Agirre et al. 2009). • **RG-65**: (Rubenstein and Goodenough 1965) is set of 65 word pairs. • **MC-30**: A smaller subset of the RG-65 dataset containing 30 word pairs (Miller and Charles 1991).

Results

We compare LCU-SWS, LCU-WS with two baselines: Exact (which uses exact counts), and CU sketch. For all the approaches, we use top 1000 contexts based on PMI values for each word. We use GW, GW-WB1, and GW-WB2 whose corpus statics are shown in Table 1 on page 3 to compute the counts. For each corpus, we use frequency cut-offs of (10, 20, and 40) based on the frequency of word-context pairs. These cut-offs were selected since sketch approaches are sensitive to frequency cut-offs compared to the Exact as found in our earlier work (Goyal et al. 2010a). The results are shown in Table 2.

First, for GW corpus shown in second column of Table 2, we use sketches with size 400 million ($400M$) counters using only 1.6 GB of main memory. Using this data-set, with frequency cutoff of 10, both Exact, and LCU-WS performs well on WS-353, WS-203, and MC-30 test-sets. However, for CU, and LCU-SWS, we need frequency cutoff of 20 to perform well. However, all the approaches on RG-65 perform better with a higher cutoff of 40. For GW-WB1 shown in third column, and GW-WB2 shown in fourth column, we use sketches with size 2 billion ($2B$) counters using 8 GB of main memory which is easily available on conventional desktop machines. For CU sketch, the results are terrible with all the frequency cut-offs, showing that this sketch is highly sensitive to frequency cutoff. For GW-WB1, using LCU-WS, performs well with all frequency cut-offs compared to Exact. However, LCU-SWS is sensitive with respect to frequency cutoff of 10 but performs well with cut-offs of 20, and 40. For GW-WB2, using LCU-WS, and LCU-SWS, they need cutoff of 40 to perform comparable to Ex-

[6] Goal of this work is not to improve distributional similarity but to show the effectiveness of LCU sketches over CU sketch.

[7] http://alfonseca.org/pubs/ws353simrel.tar.gz

act. LCU-WS, also performs reasonable with a cutoff of 20.

From the above discussion, we learn that on distributional similarity task using LCU-WS and LCU-SWS sketches are more robust than CU sketch. Second for LCU-WS, the reduction in over-estimation error for low-frequency items is more beneficial than incurring small error on mid-frequency items. Third for LCU-SWS, the reduction in over-estimation is less compared to LCU-WS but it has negligible error on mid-frequency items, and the results show that this technique is also robust compared to CU. Fourth, since, both LCU-WS, and LCU-SWS has smaller error compared to CU sketches, we can use smaller number of counters for LCU-WS, and LCU-SWS, and can get same performance as CU sketch. This shows that using LCU techniques, we can save time as well as memory.

Conclusion

In this work, we have proposed the framework of LCU sketches by combining the idea of lossy counting on top of CU sketch. In intrinsic evaluations, we show that LCU-SWS is always better than using CU sketch as it reduces over-estimation error significantly without even incurring any error on mid-frequency items. For LCU-WS, the reduction in over-estimation is larger than LCU-SWS, however this decrease comes at the expense of small error on average counts. On the task of distributional similarity, we show that using both LCU-WS, LCU-SWS is more robust, memory, and time efficient compared to CU sketch.

Acknowledgments

The authors gratefully acknowledge the support of NSF grant IIS-0712764 and Google Research Grant for Large-Data NLP. Thanks to Suresh Venkatasubramanian and Jagadeesh Jagarlamudi for useful discussions and the anonymous reviewers for many helpful comments.

References

Agirre, E.; Alfonseca, E.; Hall, K.; Kravalova, J.; Paşca, M.; and Soroa, A. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *NAACL '09: Proceedings of HLT-NAACL*.

Brants, T.; Popat, A. C.; Xu, P.; Och, F. J.; and Dean, J. 2007. Large language models in machine translation. In *Proceedings of EMNLP-CoNLL*.

Church, K., and Hanks, P. 1989. Word Association Norms, Mutual Information and Lexicography. In *Proceedings of ACL*, 76–83.

Cormode, G., and Hadjieleftheriou, M. 2008. Finding frequent items in data streams. In *VLDB*.

Cormode, G., and Muthukrishnan, S. 2004. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*.

Cormode, G. 2009. Encyclopedia entry on 'Count-Min Sketch'. In *Encyclopedia of Database Systems*. Springer. 511–516.

Estan, C., and Varghese, G. 2002. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.* 32(4).

Finkelstein, L.; Gabrilovich, E.; Matias, Y.; Rivlin, E.; Solan, Z.; Wolfman, G.; and Ruppin, E. 2002. Placing search in context: The concept revisited. In *ACM Transactions on Information Systems*.

Firth, J. 1968. A synopsis of linguistic theory 1930-1955. In Palmer, F., ed., *Selected Papers of J. R. Firth*. Longman.

Goyal, A.; Jagarlamudi, J.; Daumé III, H.; and Venkatasubramanian, S. 2010a. Sketch techniques for scaling distributional similarity to the web. In *GEMS workshop at ACL*.

Goyal, A.; Jagarlamudi, J.; Daumé III, H.; and Venkatasubramanian, S. 2010b. Sketching techniques for Large Scale NLP. In *6th WAC Workshop at NAACL-HLT*.

Goyal, A.; Daumé III, H.; and Venkatasubramanian, S. 2009. Streaming for large scale NLP: Language modeling. In *NAACL*.

Graff, D. 2003. English Gigaword. Linguistic Data Consortium, Philadelphia, PA.

Harris, Z. 1954. Distributional structure. *Word 10 (23)* 146–162.

Levenberg, A., and Osborne, M. 2009. Stream-based randomised language models for SMT. In *EMNLP*.

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

Manku, G. S. 2002. Frequency counts over data streams. In *http://www.cse.ust.hk/vldb2002/VLDB2002-proceedings/slides/S10P03slides.pdf*.

Miller, G., and Charles, W. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes* 6(1):1–28.

Pantel, P.; Crestan, E.; Borkovsky, A.; Popescu, A.-M.; and Vyas, V. 2009. Web-scale distributional similarity and entity set expansion. In *Proceedings of EMNLP*.

Ravichandran, D.; Pantel, P.; and Hovy, E. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of ACL*.

Rubenstein, H., and Goodenough, J. 1965. Contextual correlates of synonymy. *Computational Linguistics* 8:627–633.

Rusu, F., and Dobra, A. 2007. Statistical analysis of sketch estimators. In *SIGMOD '07*. ACM.

Turney, P. D. 2008. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of COLING 2008*.

Van Durme, B., and Lall, A. 2009a. Probabilistic counting with randomized storage. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*.

Van Durme, B., and Lall, A. 2009b. Streaming pointwise mutual information. In *Advances in Neural Information Processing Systems 22*.

Van Durme, B., and Lall, A. 2010. Online generation of locality sensitive hash signatures. In *Proceedings of the ACL 2010 Conference Short Papers*, 231–235.

Van Durme, B., and Lall, A. 2011. Efficient online locality sensitive hashing via reservoir counting. In *Proceedings of the ACL 2011 Conference Short Papers*.