Sketch Algorithms for Estimating Point Queries in NLP

Amit Goyal and Hal Daumé III
University of Maryland
{amit,hal}@umiacs.umd.edu

Graham Cormode
AT&T Labs–Research
graham@research.att.com

Abstract

Many NLP tasks rely on accurate statistics from large corpora. Tracking complete statistics is memory intensive, so recent work has proposed using compact approximate "sketches" of frequency distributions. We describe 10 sketch methods, including existing and novel variants. We compare and study the errors (over-estimation and under-estimation) made by the sketches. We evaluate several sketches on three important NLP problems. Our experiments show that *one* sketch performs best for all the three tasks.

1 Introduction

Since the emergence of the World Wide Web, social media and mobile devices, we have ever larger and richer examples of text data. Such vast corpora have led to leaps in the performance of many language-based tasks: the concept is that simple models trained on big data can outperform more complex models with fewer examples. However, this new view comes with its own challenges: principally, how to effectively represent such large data sets so that model parameters can be efficiently extracted? One answer is to adopt compact summaries of corpora in the form of probabilistic "sketches".

In recent years, the field of Natural Language Processing (NLP) has seen tremendous growth and interest in the use of approximation, randomization, and streaming techniques for large-scale problems (Brants et al., 2007; Turney, 2008). Much of this work relies on tracking very many statistics. For example, storing approximate counts (Talbot and Osborne, 2007; Van Durme and Lall, 2009a; Goyal and Daumé III, 2011a), computing approximate association scores like Pointwise Mutual Information (Li et al., 2008; Van Durme and Lall, 2009b; Goyal and Daumé III, 2011a), finding frequent items (like n-grams) (Goyal et al., 2009), building streaming language models (Talbot and Brants, 2008; Levenberg and Osborne, 2009), and distributional similarity (Ravichandran et al., 2005; Van Durme and Lall, 2010). All these problems ultimately depend on approximate counts of items (such as n-grams, word pairs and word-context pairs). Thus we focus on solving this central problem in the context of NLP applications.

Sketch algorithms (Charikar et al., 2004; Cormode, 2011) are a memory- and time-efficient solution to answering point queries. Recently in NLP, we (Goyal and Daumé III, 2011a) demonstrated that a version of the Count-Min sketch (Cormode and Muthukrishnan, 2004) accurately solves three large-scale NLP problems using small bounded memory footprint. However, there are several other sketch algorithms, and it is not clear why this instance should be preferred amongst these. In this work, we conduct a systematic study and compare many sketch techniques which answer point queries with focus on large-scale NLP tasks. While sketches have been evaluated within the database community for finding frequent items (Cormode and Hadjieleftheriou, 2008) and join-size estimation (Rusu and Dobra, 2007), this is the *first* comparative study for NLP problems.

Our work includes three contributions: (1) We propose novel variants of existing sketches by extending the idea of *conservative update* to them. We propose Count sketch (Charikar et al., 2004) with conservative update (COUNT-CU) and Count-mean-min sketch with conservative update

(CMM-CU). The motivation behind proposing new sketches is inspired by the success of Count-Min sketch with conservative update in our earlier work (Goyal and Daumé III, 2011a). (2) We empirically compare and study the errors in approximate counts for several sketches. Errors can be over-estimation, under-estimation, or a combination of the two. We also evaluate their performance via Pointwise Mutual Information and LogLikelihood Ratio. (3) We use sketches to solve three important NLP problems. Our experiments show that sketches can be very effective for these tasks, and that the best results are obtained using the "conservative update" technique. Across all the three tasks, *one* sketch (CM-CU) performs best.

2 Sketches

In this section, we review existing sketch algorithms from the literature, and propose novel variants based on the idea of conservative update (Estan and Varghese, 2002). The term 'sketch' refers to a class of algorithm that represents a large data set with a compact summary, typically much smaller than the full size of the input. Given an input of N items $(\mathrm{x}_1, \mathrm{x}_2 \ldots \mathrm{x}_N)$, each item x (where x is drawn from some domain U) is mapped via hash functions into a small sketch vector that records frequency information. Thus, the sketch does not store the items explicitly, but only information about the frequency distribution. Sketches support fundamental queries on their input such as point, range and inner product queries to be quickly answered approximately. In this paper, we focus on point queries, which ask for the (approximate) count of a given item.

The algorithms we consider are *randomized* and *approximate*. They have two user-chosen parameters ϵ and δ. ϵ controls the amount of tolerable error in the returned count and δ controls the probability with which the error exceeds the bound ϵ. These values of ϵ and δ determine respectively the width w and depth d of a two-dimensional array sk$[\cdot,\cdot]$ of count information. The depth d denotes the number of hash functions employed by the sketch algorithms.

Sketch Operations. Every sketch has two operations: UPDATE and QUERY to update and estimate the count of an item. They all guarantee essentially *constant time* operation (technically, this grows as $O(\log(\frac{1}{\delta}))$ but in practice this is set to a constant) per UPDATE and QUERY. Moreover, sketches can be combined: given two sketches s_1 and s_2 computed (using the same parameters w and d, and same set of d hash functions) over different inputs, a sketch of the combined input is obtained by adding the individual sketches, entry-wise. The time to perform the COMBINE operation on sketches is $O(d \times w)$, independent of the data size. This property enables sketches to be implemented in distributed setting, where each machine computes the sketch over a small portion of the corpus and makes it *scalable* to large datasets.

2.1 Existing sketch algorithms

This section describes sketches from the literature:
Count-Min sketch (CM): The CM (Cormode and Muthukrishnan, 2004) sketch has been used effectively for many large scale problems across several areas, such as Security (Schechter et al., 2010), Machine Learning (Shi et al., 2009; Aggarwal and Yu, 2010), Privacy (Dwork et al., 2010), and NLP (Goyal and Daumé III, 2011a). The sketch stores an array of size $d \times w$ counters, along with d hash functions (drawn from a pairwise-independent family), one for each row of the array. Given an input of N items $(\mathrm{x}_1, \mathrm{x}_2 \ldots \mathrm{x}_N)$, each of the hash functions $h_k{:}U \rightarrow \{1 \ldots w\}, \forall 1 \leq k \leq d$, takes an item from the input and maps it into a counter indexed by the corresponding hash function.
UPDATE: For each new item "x" with count c, the sketch is updated as:

$$\mathrm{sk}[k, h_k(\mathrm{x})] \leftarrow \mathrm{sk}[k, h_k(\mathrm{x})] + c, \ \ \forall 1 \leq k \leq d.$$

QUERY: Since multiple items are hashed to the same index for each array row, the stored frequency in each row is guaranteed to *overestimate* the true count, making it a biased estimator. Therefore, to answer the point query (QUERY (x)), CM returns the minimum over all the d positions x is stored.

$$\hat{c}(\mathrm{x}) = \min_k \ \mathrm{sk}[k, h_k(\mathrm{x})], \ \ \forall 1 \leq k \leq d.$$

Setting $w{=}\frac{2}{\epsilon}$ and $d{=}\log(\frac{1}{\delta})$ ensures all reported frequencies by CM exceed the true frequencies by at most ϵN with probability of at least $1 - \delta$. This makes the space used by the algorithm $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$.
Spectral Bloom Filters (SBF): Cohen and Matias (2003) proposed SBF, an extension to Bloom Filters (Bloom, 1970) to answer point queries. The

UPDATE and QUERY procedures for SBF are the same as Count-Min (CM) sketch, except that the range of all the hash functions for SBF are the full array: $h_k{:}U \rightarrow \{1 \ldots w \times d\}, \forall 1 \leq k \leq d$. While CM and SBF are very similar, only CM provides guarantees on the query error.

Count-mean-min (CMM): The motivation behind the CMM (Deng and Rafiei, 2007) sketch is to provide an unbiased estimator for Count-Min (CM) sketch. The construction of CMM sketch is identical to the CM sketch, while the QUERY procedure differs. Instead of returning the minimum value over the d counters (indexed by d hash functions), CMM deducts the value of estimated noise from each of the d counters, and return the median of the d residues. The noise is estimated as $(N - \text{sk}[k, h_k(\text{x})])/(w-1)$. Nevertheless, the median estimate $(\hat{f_1})$ over the d residues can overestimate more than the original CM sketch $\min$ estimate $(\hat{f_2})$, so we return $\min\ (\hat{f_1},\hat{f_2})$ as the final estimate for CMM sketch. CMM gives the same theoretical guarantees as Count sketch (below).

Count sketch (COUNT) (Charikar et al., 2004): COUNT (aka Fast-AGMS) keeps two hash functions for each row, h_k maps items onto $[1, w]$, and g_k maps items onto $\{-1,+1\}$. UPDATE: For each new item "x" with count c:

$$\text{sk}[k, h_k(\text{x})] \leftarrow \text{sk}[k, h_k(\text{x})] + c \cdot g_k(\text{x}),\ \ \forall 1 \leq k \leq d.$$

QUERY: the median over the d rows is an unbiased estimator of the point query:

$$\hat{c(\text{x})} = \text{median}_\text{k}\ \text{sk}[k, h_k(\text{x})] \cdot g_k(\text{x}),\ \ \forall 1 \leq k \leq d.$$

Setting $w{=}\frac{2}{\epsilon^2}$ and $d{=}\log(\frac{4}{\delta})$ ensures that all reported frequencies have error at most $\epsilon(\sum_{i=1}^{N} f_i^2)^{1/2} \leq \epsilon N$ with probability at least $1 - \delta$. The space used by the algorithm is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$.

2.2 Conservative Update sketch algorithms

In this section, we propose novel variants of existing sketches (see Section 2) by combining them with the conservative update process (Estan and Varghese, 2002). The idea of conservative update (also known as Minimal Increase (Cohen and Matias, 2003)) is to only increase counts in the sketch by the minimum amount needed to ensure the estimate remains accurate. It can easily be applied to Count-Min (CM) sketch and Spectral Bloom Filters (SBF) to further improve the estimate of a point query. Goyal and Daumé III (2011a) showed that CM sketch with conservative update reduces the amount of over-estimation error by a factor of at least 1.5, and also improves performance on three NLP tasks.

Note that while conservative update for CM and SBF never increases the error, there is no guaranteed improvement. The method relies on seeing multiple updates in sequence. When a large corpus is being summarized in a distributed setting, we can apply conservative update on each sketch independently before combining the sketches together (see "Sketch Operations" in Section 2).

Count-Min sketch with conservative update (CM-CU): The QUERY procedure for CM-CU (Cormode, 2009; Goyal and Daumé III, 2011a) is identical to Count-Min. However, to UPDATE an item "x" with frequency c, we first compute the frequency $\hat{c}(\text{x})$ of this item from the existing data structure ($\forall 1 \leq k \leq d$, $\hat{c}(\text{x}) = \min_k\ \text{sk}[k, h_k(\text{x})]$) and the counts are updated according to:

$$\text{sk}[k, h_k(\text{x})] \leftarrow \max\{\text{sk}[k, h_k(\text{x})], \hat{c}(\text{x}) + \text{c}\} \qquad (*).$$

The intuition is that, since the point query returns the minimum of all the d values, we update a counter only if it is necessary as indicated by $(*)$. This heuristic avoids unnecessarily updating counter values to reduce the over-estimation error.

Spectral Bloom Filters with conservative update (SBF-CU): The QUERY procedure for SBF-CU (Cohen and Matias, 2003) is identical to SBF. SBF-CU UPDATE procedure is similar to CM-CU, with the difference that all d hash functions have the common range $d \times w$.

Count-mean-min with conservative update (CMM-CU): We propose a *new* variant to reduce the over-estimation error for CMM sketch. The construction of CMM-CU is identical to CM-CU. However, due to conservative update, each row of the sketch is not updated for every update, hence the sum of counts over each row ($\sum_i \text{sk}[k, i]$, $\forall 1 \leq k \leq d$) is not equal to input size N. Hence, the estimated noise to be subtracted here is $(\sum_i \text{sk}[k,i] - \text{sk}[k, h_k(\text{x})])\ /\ (w-1)$. CMM-CU deducts the value of estimated noise from each of the d counters, and returns the median of the d residues as the point query.

Count sketch with conservative update (COUNT-CU): We propose a *new* variant to reduce over-estimation error for the COUNT sketch. The QUERY procedure for COUNT-CU is the same as

COUNT. The UPDATE procedure follows the same outline as CM-CU, but uses the current estimate $\hat{c}(\mathrm{x})$ from the COUNT sketch, i.e.

$$\hat{c}(\mathrm{x}) = \mathrm{median}_k \ \mathrm{sk}[k, h_k(\mathrm{x})] \cdot g_k(\mathrm{x}), \ \ \forall 1 \leq k \leq d.$$

Note, this heuristic is not as strong as for CM-CU and SBF-CU because COUNT can have both over-estimate and under-estimate errors.

Lossy counting with conservative update (LCU-WS): LCU-WS (Goyal and Daumé III, 2011b) was proposed to reduce the amount of over-estimation error for CM-CU sketch, without incurring too much under-estimation error. This scheme is inspired by lossy counting (Manku and Motwani, 2002). In this approach, the input sequence is conceptually divided into *windows*, each containing $1/\gamma$ items. The size of each window is equal to size of the sketch i.e. $d \times w$. Note that there are γN windows; let t denote the index of current window. At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if (sk$[i,j] > 0$ and sk$[i,j] \leq t$), then sk$[i,j] \leftarrow$ sk$[i,j] - 1$. The idea is to remove the contribution of small items colliding in the same entry, while not altering the count of frequent items. The current window index is used to draw this distinction. Here, all reported frequencies $\hat{f}$ have both under and over estimation error: $f - \gamma N \leq \hat{f} \leq f + \epsilon N$.

Lossy counting with conservative update II (LCU-SWS): This is a variant of the previous scheme, where the counts of the sketch are decreased more conservatively. Hence, this scheme has worse over-estimation error compared to LCU-WS, with better under-estimation. Here, only those counts are decremented which are at most the square root of current window index, t. At window boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if (sk$[i,j] > 0$ and sk$[i,j] \leq \lceil\sqrt{t}\rceil$), then sk$[i,j] \leftarrow$ sk$[i,j] - 1$. LCU-SWS has similar analytical bounds to LCU-WS.

3 Intrinsic Evaluations

We empirically compare and study the errors in approximate counts for all 10 sketches. Errors can be over-estimation, under-estimation, or a combination of the two. We also study the behavior of approximate Pointwise Mutual Information and Log Likelihood Ratio for the sketches.

3.1 Experimental Setup

DATA: We took 50 million random sentences from Gigaword (Graff, 2003). We split this data in 10 chunks of 5 million sentences each. Since all sketches have probabilistic bounds, we report average results over these 10 chunks. For each chunk, we generate counts of all word pairs within a window size 7. This results in an average stream size of 194 million word pair tokens and 33.5 million word pair types per chunk.

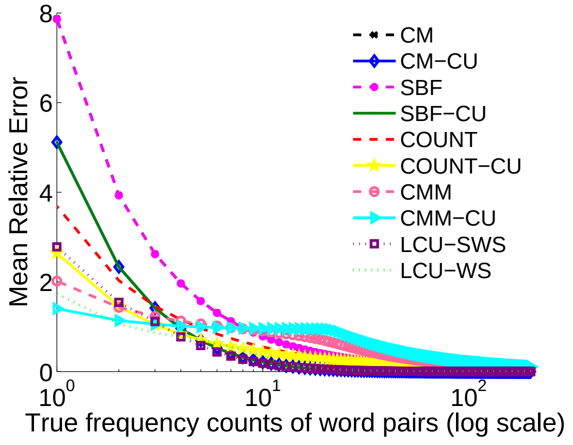
To compare error in various sketch counts, first we compute the exact counts of all the word pairs. Second, we store the counts of all the word pairs in all the sketches. Third, we query sketches to generate *approximate* counts of all the word pairs. Recall, we do not store the word pairs explicitly in sketches but only a compact summary of the associated counts.

We fix the size of each sketch to be $w = \frac{20 \times 10^6}{3}$ and $d = 3$. We keep the size of sketches equal to allow fair comparison among them. Prior work (Deng and Rafiei, 2007; Goyal and Daumé III, 2011a) showed with *fixed sketch size*, a small number of hash functions (d=number of hash functions) with large w (or range) give rise to small error over counts. Next, we group all word pairs with the same true frequency into a single bucket. We then compute the Mean Relative Error (MRE) in each of these buckets. Because different sketches have different accuracy behavior on low, mid, and high frequency counts, making this distinction based on frequency lets us determine the regions in which different sketches perform best. Mean Relative Error (MRE) is defined as the average of absolute difference between the predicted and the exact value divided by the exact value over all the word pairs in each bucket.

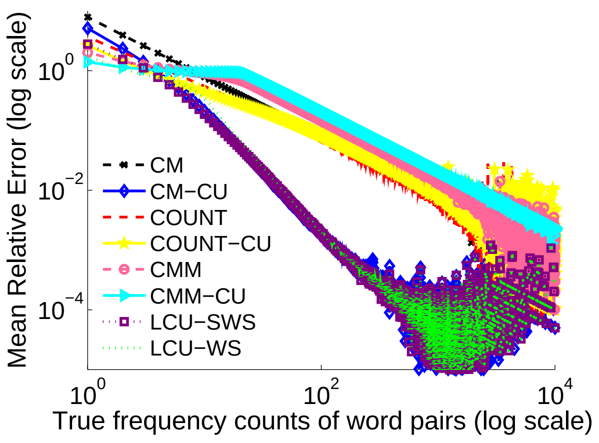
3.2 Studying the Error in Counts

We study the errors produced by all 10 sketches. Since various sketches result in different errors on low, mid, and high frequency counts, we plot the results with a linear error scale (Fig. 1(a)) to highlight the performance for low frequency counts, and with a log error scale (Fig. 1(b)) for mid and high frequency counts.

We make several observations on low frequency counts from Fig. 1(a). (1) Count-Min (CM) and



(a) Focusing on low frequency counts



(b) Focusing on mid and high frequency counts

Figure 1: Comparing several sketches for input size of 75 million word pairs. Size of each sketch: $w = \frac{20 \times 10^6}{3}$ and $d = 3$. All items with same exact count are put in one bucket and we plot Mean Relative Error on the y-axis with exact counts on the x-axis.

Spectral Bloom Filters (SBF) have identical MRE for word pairs. Using conservative update with CM (CM-CU) and SBF (SBF-CU) reduces the MRE by a factor of 1.5. MRE for CM-CU and SBF-CU is also identical. (2) COUNT has better MRE than CM-CU and using conservative update with COUNT (COUNT-CU) further reduces the MRE. (3) CMM has better MRE than COUNT and using conservative update with CMM (CMM-CU) further reduces the MRE. (4) Lossy counting with conservative update variants (LCU-SWS, LCU-WS) have comparable MRE to COUNT-CU and CMM-CU respectively.

In Fig. 1(b), we do not plot the SBF variants as SBF and CM variants had identical MRE in Fig. 1(a). From Figure 1(b), we observe that, CM, COUNT, COUNT-CU, CMM, CMM-CU sketches have worse MRE than CM-CU, LCU-SWS, and LCU-WS for mid and high frequency counts. CM-CU, LCU-SWS, and LCU-WS have zero MRE for all the counts > 1000.

To summarize the above observations, for those NLP problems where we cannot afford to make errors on mid and high frequency counts, we should employ CM-CU, LCU-SWS, and LCU-WS sketches. If we want to reduce the error on low frequency counts, LCU-WS generates least error. For NLP tasks where we can allow error on mid and high frequency counts but not on low frequency counts, CMM-CU sketch is best.

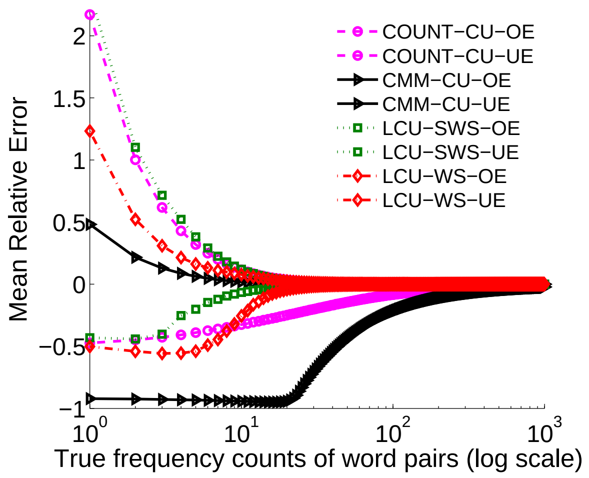
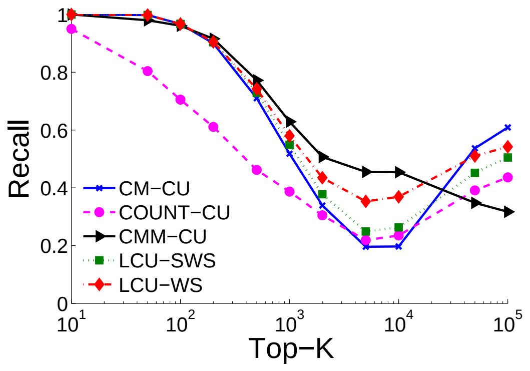


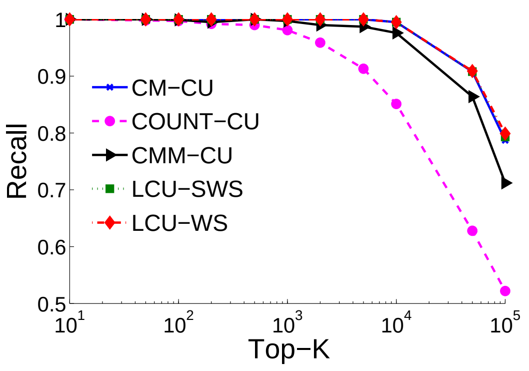
Figure 2: Compare several sketches on over-estimation and under-estimation errors with respect to exact counts.

3.3 Examining OE and UE errors

In many NLP applications, we are willing to tolerate either over-estimation or under-estimation errors. Hence we breakdown the error into over-estimation (OE) and under-estimation (UE) errors for the six best-performing sketches (COUNT, COUNT-CU, CMM, CMM-CU, LCU-SWS, and LCU-WS). To accomplish that, rather than using absolute error values, we divide the values into over-estimation (positive), and under-estimation (negative) error buck-



(a) PMI



(b) LLR

Figure 3: Evaluate the approximate PMI and LLR rankings (obtained using various sketches) with the exact rankings.

ets. Hence, to compute the over-estimation MRE, we take the average of positive values over all the items in each bucket. For under-estimation, we take the average over the negative values. We can make several interesting observations from Figure 2: (1) Comparing COUNT-CU and LCU-SWS, we learn that both have the same over-estimation errors. However, LCU-SWS has less under-estimation error than COUNT-CU. Therefore, LCU-SWS is always better than COUNT-CU. (2) LCU-WS has less over-estimation than LCU-SWS but with more under-estimation error on mid frequency counts. LCU-WS has less under-estimation error than COUNT-CU. (3) CMM-CU has the least over-estimation error and most under-estimation error among all the compared sketches.

From the above experiments, we conclude that tasks sensitive to under-estimation should use the CM-CU sketch, which guarantees over-estimation. However, if we are willing to make some under-estimation error with less over-estimation error, then LCU-WS and LCU-SWS are recommended. Lastly, to have minimal over-estimation error with willingness to accept large under-estimation error, CMM-CU is recommended.

3.4 Evaluating association scores ranking

Last, in many NLP problems, we are interested in association rankings obtained using Pointwise Mutual Information (PMI) and Log Likelihood Ratio (LLR). In this experiment, we compare the word pairs association rankings obtained using PMI and LLR from several sketches and exact word pair counts. We use recall to measure the number of top-K sorted word pairs that are found in both the rankings.

In Figure 3(a), we compute the recall for CM-CU, COUNT-CU, CMM-CU, LCU-SWS, and LCU-WS sketches at several top-K thresholds of word pairs for approximate PMI ranking. We can make several observations from Figure 3(a). COUNT-CU has the worst recall for almost all the top-K settings. For top-K values less than 750, all sketches except COUNT-CU have comparable recall. Meanwhile, for K greater than 750, LCU-WS has the best recall. The is because PMI is sensitive to low frequency counts (Church and Hanks, 1989), over-estimation of the counts of low frequency word pairs can make their approximate PMI scores worse.

In Figure 3(b), we compare the LLR rankings. For top-K values less than 1000, all the sketches have comparable recall. For top-K values greater than 1000, CM-CU, LCU-SWS, and LCU-WS perform better. The reason for such a behavior is due to LLR favoring high frequency word pairs, and COUNT-CU and CMM-CU making under-estimation error on high frequency word pairs.

To summarize, to maintain top-K PMI rankings making over-estimation error is *not* desirable. Hence, LCU-WS is recommended for PMI rankings. For LLR, producing under-estimation error is *not* preferable and therefore, CM-CU, LCU-WS, and LCU-SWS are recommended.

Test Set	Random			Buckets			Neighbor		
Model	CM-CU	CMM-CU	LCU-WS	CM-CU	CMM-CU	LCU-WS	CM-CU	CMM-CU	LCU-WS
50M	87.2	74.3	86.5	83.9	72.9	83.2	71.7	64.7	72.1
100M	90.4	79.0	91.0	86.5	76.9	86.9	73.4	67.2	**74.7**
200M	**93.3**	83.1	92.9	**88.3**	80.1	**88.4**	**75.0**	69.0	**75.4**
500M	**94.4**	86.6	**94.1**	**89.3**	83.4	**89.3**	**75.7**	70.8	**75.5**
1B	**94.4**	88.7	**94.4**	**89.5**	85.1	**89.5**	**75.8**	71.9	**75.8**
Exact	**94.5**			**89.5**			**75.8**		

Table 1: Pseudo-words evaluation on accuracy metric for selectional preferences using several sketches of different sizes against the exact. There is *no* statistically significant difference (at $p < 0.05$ using bootstrap resampling) among bolded numbers.

4 Extrinsic Evaluation

4.1 Experimental Setup

We study three important NLP applications, and compare the three best-performing sketches: Count-Min sketch with conservative update (CM-CU), Count-mean-min with conservative update (CMM-CU), and Lossy counting with conservative update (LCU-WS). The above mentioned 3 sketches are selected from 10 sketches (see Section 2) considering these sketches make errors on different ranges of the counts: low, mid and, high frequency counts as seen in our intrinsic evaluations in Section 3. The goal of this experiment is to show the effectiveness of sketches on large-scale language processing tasks.

These adhere to the premise that simple methods using large data can dominate more complex models. We purposefully select simple methods as they use approximate counts and associations directly to solve these tasks. This allows us to have a fair comparison among different sketches, and to more directly see the impact of different choices of sketch on the task outcome. Of course, sketches are still broadly applicable to many NLP problems where we want to count (many) items or compute associations: e.g. language models, Statistical Machine Translation, paraphrasing, bootstrapping and label propagation for automatically creating a knowledge base and finding interesting patterns in social media.

Data: We use **Gigaword** (Graff, 2003) and a 50% portion of a copy of news web (**GWB50**) crawled by (Ravichandran et al., 2005). The raw size of **Gigaword** (**GW**) and **GWB50** is 9.8 GB and 49 GB with 56.78 million and 462.60 sentences respectively. For both the corpora, we split the text into sentences, tokenize and convert into lower-case.

4.2 Pseudo-Words Evaluation

In NLP, it is difficult and time consuming to create annotated test sets. This problem has motivated the use of pseudo-words to automatically create the test sets without human annotation. The pseudo-words are a common way to evaluate selectional preferences models (Erk, 2007; Bergsma et al., 2008) that measure the strength of association between a predicate and its argument filler, e.g., that the noun "song" is likely to be the object of the verb "sing".

A pseudo-word is the conflation of two words (e.g. song/dance). One word is the original in a sentence, and the second is the confounder. For example, in our task of selectional preferences, the system has to decide for the verb "sing" which is the correct object between "song"/"dance". Recently, Chambers and Jurafsky (2010) proposed a simple baseline based on co-occurrence counts of words, which has state-of-the-art performance on pseudo-words evaluation for selectional preferences.

We use a simple approach (without any typed dependency data) similar to Chambers and Jurafsky (2010), where we count all word pairs (except word pairs involving stop words) that appear within a window of size 3 from **Gigaword** (9.8 GB). That generates 970 million word pair tokens (stream size) and 94 million word pair types. Counts of all the 94 million unique word pairs are stored in CM-CU, CMM-CU, and LCU-WS. For a target verb, we return that noun which has higher co-occurrence count with it, as the correct selectional preference. We evaluate on Chambers and Jurafsky's three test sets[1] (excluding instances involving stop words) that are based on different strategies in selecting confounders: *Random* (4081 instances), *Buckets* (4028

[1] `http://www.usna.edu/Users/cs/nchamber/data/pseudowords/`

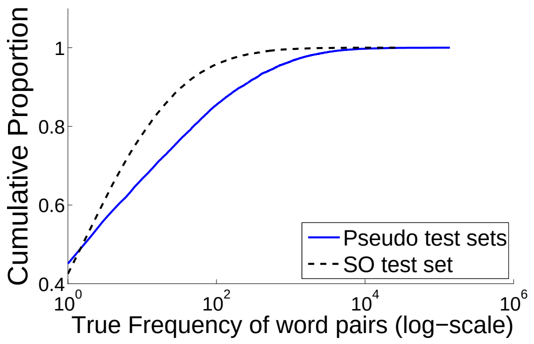


Figure 4: Determining the proportion of low, mid and high frequency test word pairs in **Gigaword** (**GW**).

instances), and *Neighbor* (3881 instances). To evaluate against the exact counts, we compute exact counts for only those word pairs that are present in the test sets. Accuracy is used for evaluation and is defined as the percentage of number of correctly identified pseudo words.

In Fig. 4, we plot the cumulative proportion of true frequency counts of all word pairs (from the three tests) in **Gigaword** (**GW**). To include unseen word pairs from test set in **GW** on log-scale in Fig. 4, we increment the true counts of all the word pairs by 1. This plot demonstrates that 45% of word-pairs are unseen in **GW**, and 67% of word pairs have counts less than 10. Hence, to perform better on this task, it is essential to *accurately maintain counts of rare word pairs*.

In Table 1, we vary the size of all sketches (50 million (M), $100M$, $200M$, $500M$ and 1 billion ($1B$) counters) with 3 hash functions to compare them against the exact counts. It takes 1.8 GB uncompressed space to maintain the exact counts on the disk. Table 1 shows that with sketches of size $> 200M$ on all the three test sets, CM-CU and LCU-WS are comparable to exact. However, the CMM-CU sketch performs less well. We conjecture the reason for such a behavior is due to loss of recall (information about low frequency word pairs) by under-estimation error. For this task CM-CU and LCU-WS scales to storing $94M$ unique word pairs using $200M$ integer (4 bytes each) counters (using 800 MB) $<$ 1.8 GB to maintain exact counts. Moreover, these results are comparable to Chambers and Jurafsky's state-of-the-art framework.

Data	Exact	CM-CU	CMM-CU	LCU-WS
GW	*74.2*	*74.0*	65.3	*72.9*
GWB50	**81.2**	**80.9**	74.9	78.3

Table 2: Evaluating Semantic Orientation on accuracy metric using several sketches of 2 billion counters against exact. Bold and italic numbers denote no statistically significant difference.

4.3 Finding Semantic Orientation of a word

Given a word, the task of finding its Semantic Orientation (SO) (Turney and Littman, 2003) is to determine if the word is more probable to be used in positive or negative connotation. We use Turney and Littman's (2003) state-of-the-art framework to compute the SO of a word. We use same seven positive words (good, nice, excellent, positive, fortunate, correct, and superior) and same seven negative words (bad, nasty, poor, negative, unfortunate, wrong, and inferior) from their framework as seeds. The SO of a given word is computed based on the strength of its association with the seven positive words and the seven negative words. Association scores are computed via Pointwise Mutual Information (PMI). We compute the SO of a word "w" as:

$$\text{SO(w)} = \sum_{\text{p}\in\text{Pos}} PMI(\text{p},\text{w}) - \sum_{\text{n}\in\text{Neg}} PMI(\text{n},\text{w})$$

where, Pos and Neg denote the seven positive and negative seeds respectively. If this score is negative, we predict the word as negative; otherwise, we predict it as positive. We use the General Inquirer lexicon[2] (Stone et al., 1966) as a benchmark to evaluate the semantic orientation similar to Turney and Littman's (2003) work. Our test set consists of 1611 positive and 1987 negative words. Accuracy is used for evaluation and is defined as the percentage of number of correctly identified SO words.

We evaluate SO of words on two different sized corpora (see Section 4.1): **Gigaword** (**GW**) (9.8GB), and **GW** with 50% news web corpus (**GWB50**) (49GB). We fix the size of all sketches to 2 billion ($2B$) counters with 5 hash functions. We store exact counts of all words in a hash table for both **GW** and **GWB50**. We count all word pairs (except word pairs involving stop words) that appear within a window of size 7 from **GW** and **GWB50**. This yields 2.67 billion(B) tokens and $.19B$ types

[2]The General Inquirer lexicon is freely available at `http://www.wjh.harvard.edu/~inquirer/`

	Test Set	**WS-203**			**MC-30**		
	Model	CM-CU	CMM-CU	LCU-WS	CM-CU	CMM-CU	LCU-WS
PMI	10M	**.58**	.25	.28	**.67**	.20	.16
	50M	**.44**	.23	.41	**.61**	.22	.31
	200M	**.53**	**.44**	**.47**	**.57**	.28	**.43**
	Exact		**.52**			**.50**	
LLR	10M	*.47*	.27	.29	*.50*	.29	.10
	50M	*.42*	.31	.34	*.48*	.32	*.35*
	200M	*.41*	*.35*	*.39*	*.40*	*.31*	*.40*
	Exact		*.42*			*.41*	

Table 3: Evaluating distributional similarity using sketches. Scores are evaluated using rank correlation ρ. Bold and italic numbers denote no statistically significant difference.

from **GW** and $13.20B$ tokens and $0.8B$ types from **GWB50**. Next, we compare the sketches against the exact counts over two different size corpora.

Table 2 shows that increasing the amount of data improves the accuracy of identifying the SO of a word. We get an absolute increase of 7 percentage points (with exact counts) in accuracy (The 95% statistical significance boundary for accuracy is about $\pm$ 1.5.), when we add 50% web data (**GWB50**). CM-CU results are equivalent to exact counts for all the corpus sizes. These results are also comparable to Turney's (2003) accuracy of 82.84%. However, CMM-CU results are worse by absolute 8.7 points and 6 points on **GW** and **GWB50** respectively with respect to CM-CU. LCU-WS is better than CMM-CU but worse than CM-CU. Using $2B$ integer (4 bytes each) counters (bounded memory footprint of 8 GB), CM-CU scales to $0.8B$ word pair types (It takes 16 GB uncompressed disk space to store *exact* counts of all the unique word pair types.).

Figure 4 has similar frequency distribution of word pairs[3] in SO test set as pseudo-words evaluation test sets word pairs. Hence, CMM-CU again has substantially worse results than CM-CU due to loss of recall (information about low frequency word pairs) by under-estimation error. We can conclude that for this task CM-CU is best.

4.4 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. For this work, we define context for a given term 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 use PMI and LLR to compute the association score (AS) between the term and each of the context to generate the context vector. We use the cosine similarity measure to find the distributional similarity between the context vectors for each of the terms.

We use two test sets which consist of word pairs, and their corresponding human rankings. We generate the word pair rankings using distributional similarity. We report the Spearman's rank correlation coefficient (ρ) between the human and distributional similarity rankings. We report results on two test sets: **WS-203**: A set of 203 word pairs marked according to similarity (Agirre et al., 2009). **MC-30**: A set of 30 noun pairs (Miller and Charles, 1991).

We evaluate distributional similarity on **Gigaword** (**GW**) (9.8GB) (see Section 4.1). First, we store exact counts of all words and contexts in a hash table from **GW**. Next, we count all the word-context pairs and store them in CM-CU, CMM-CU, and LCU-WS sketches. That generates a stream of size 3.35 billion ($3.35B$) word-context pair tokens and 215 million unique word-context pair types (It takes 4.6 GB uncompressed disk space to store *exact* counts of all these unique word-context pair types.). For every target word in the test set, we maintain top-1000 approximate AS scores contexts using a priority queue, by passing over the corpus a second time. Finally, we use cosine similarity with these approximate top-K context vectors to compute distributional similarity.

In Table 3, we vary the size of all sketches across 10 million (M), $50M$, and $200M$ counters with 3 hash functions. The results using PMI shows that CM-CU has best ρ on both **WS-203** and **MC-30** test sets. The results for LLR in Table 3 show similar trends with CM-CU having best results on small size sketches. Thus, CM-CU scales using $10M$ counters (using fixed memory of 40 MB versus 4.6 GB to store exact counts). These results are compa-

[3]Consider only those pairs in which one word appears in the seed list and the other word appears in the test set.

rable against the state-of-the-art results for distributional similarity (Agirre et al., 2009).

On this task CM-CU is best as it avoids loss of recall (information about low frequency word pairs) due to under-estimation error. For a target word that has low frequency, using CMM-CU will not generate any contexts for it, as it will have large under-estimation error for word-context pairs counts. This phenomenon is demonstrated in Table 3, where CMM-CU and LCU-WS have worse result with small size sketches.

5 Conclusion

In this work, we systematically studied the problem of estimating point queries using different sketch algorithms. As far as we know, this represents the *first* comparative study to demonstrate the relative behavior of sketches in the context of NLP applications. We proposed two novel sketch variants: Count sketch (Charikar et al., 2004) with conservative update (COUNT-CU) and Count-mean-min sketch with conservative update (CMM-CU). We empirically showed that CMM-CU has under-estimation error with small over-estimation error, CM-CU has only over-estimation error, and LCU-WS has more under-estimation error than over-estimation error. Finally, we demonstrated CM-CU has better results on all three tasks: pseudo-words evaluation for selectional preferences, finding semantic orientation task, and distributional similarity. This shows that maintaining information about low frequency items (even with over-estimation error) is better than throwing away information (under-estimation error) about rare items.

Future work is to reduce the bit size of each counter (instead of the number of counters), as has been tried for other summaries (Talbot and Osborne, 2007; Talbot, 2009; Van Durme and Lall, 2009a) in NLP. However, it may be challenging to combine this with conservative update.

Acknowledgments

This work was partially supported by NSF Award IIS-1139909. Thanks to Suresh Venkatasubramanian for useful discussions and the anonymous reviewers for many helpful comments.

References

Charu C. Aggarwal and Philip S. Yu. 2010. On classification of high-cardinality data streams. In *SDM'10*, pages 802–813.

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *NAACL '09: Proceedings of HLT-NAACL.*

Shane Bergsma, Dekang Lin, and Randy Goebel. 2008. Discriminative learning of selectional preference from unlabeled text. In *Proc. EMNLP*, pages 59–68, Honolulu, Hawaii, October.

Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of EMNLP-CoNLL.*

Nathanael Chambers and Dan Jurafsky. 2010. Improving the use of pseudo-words for evaluating selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 445–453. Association for Computational Linguistics.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2004. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312:3–15, January.

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

Saar Cohen and Yossi Matias. 2003. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 241–252. ACM.

Graham Cormode and Marios Hadjieleftheriou. 2008. Finding frequent items in data streams. In *VLDB*.

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

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

Graham Cormode. 2011. Sketch techniques for approximate query processing. Foundations and Trends in Databases. NOW publishers.

Fan Deng and Davood Rafiei. 2007. New estimation algorithms for streaming data: Count-min can do more. `http://webdocs.cs.ualberta.ca/`.

Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. 2010. Pan-private streaming algorithms. In *Proceedings of ICS*.

Katrin Erk. 2007. A simple, similarity-based model for selectional preferences. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, volume 45, pages 216–223. Association for Computational Linguistics.

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

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

Amit Goyal and Hal Daumé III. 2011a. Approximate scalable bounded space sketch for large data NLP. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Amit Goyal and Hal Daumé III. 2011b. Lossy conservative update (LCU) sketch: Succinct approximate count storage. In *Conference on Artificial Intelligence (AAAI)*.

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

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

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

Abby Levenberg and Miles Osborne. 2009. Stream-based randomised language models for SMT. In *EMNLP*, August.

Ping Li, Kenneth Ward Church, and Trevor Hastie. 2008. One sketch for all: Theory and application of conditional random sampling. In *Neural Information Processing Systems*, pages 953–960.

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

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

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of ACL*.

Florin Rusu and Alin Dobra. 2007. Statistical analysis of sketch estimators. In *SIGMOD '07*. ACM.

Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. 2010. Popularity is everything: a new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, HotSec'10, pages 1–8, Berkeley, CA, USA. USENIX Association.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. 2009. Hash kernels for structured data. *Journal Machine Learning Research*, 10:2615–2637, December.

Philip J. Stone, Dexter C. Dunphy, Marshall S. Smith, and Daniel M. Ogilvie. 1966. *The General Inquirer: A Computer Approach to Content Analysis*. MIT Press.

David Talbot and Thorsten Brants. 2008. Randomized language models via perfect hash functions. In *Proceedings of ACL-08: HLT*.

David Talbot and Miles Osborne. 2007. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EM NLP-CoNLL)*.

David Talbot. 2009. Succinct approximate counting of skewed data. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*.

Peter D. Turney and Michael L. Littman. 2003. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Trans. Inf. Syst.*, 21:315–346, October.

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

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

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

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