Sketching Techniques for Large Scale NLP

Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian
University of Utah, School of Computing
{amitg,jags,hal,suresh}@cs.utah.edu

Abstract

In this paper, we address the challenges posed by large amounts of text data by exploiting the power of hashing in the context of streaming data. We explore sketch techniques, especially the Count-Min Sketch, which approximates the frequency of a word pair in the corpus without explicitly storing the word pairs themselves. We use the idea of a conservative update with the Count-Min Sketch to reduce the average relative error of its approximate counts by a factor of two. We show that it is possible to store all words and word pairs counts computed from 37 GB of web data in just 2 billion counters (8 GB RAM). The number of these counters is up to 30 times less than the stream size which is a big memory and space gain. In Semantic Orientation experiments, the PMI scores computed from 2 billion counters are as effective as exact PMI scores.

1 Introduction

Approaches to solve NLP problems (Brants et al., 2007; Turney, 2008; Ravichandran et al., 2005) always benefited from having large amounts of data. In some cases (Turney and Littman, 2002; Patwardhan and Riloff, 2006), researchers attempted to use the evidence gathered from web via search engines to solve the problems. But the commercial search engines limit the number of automatic requests on a daily basis for various reasons such as to avoid fraud and computational overhead. Though we can crawl the data and save it on disk, most of the current approaches employ data structures that reside in main memory and thus do not scale well to huge corpora.

Fig. 1 helps us understand the seriousness of the situation. It plots the number of unique words/word pairs versus the total number of words in a corpus of size 577 MB. Note that the plot is in log-log scale. This 78 million word corpus generates 63 thousand unique words and 118 million unique word pairs. As expected, the rapid increase in number of unique word pairs is much larger than the increase in number of words. Hence, it shows that it is computationally infeasible to compute counts of all word pairs with a giant corpora using conventional main memory of 8 GB.

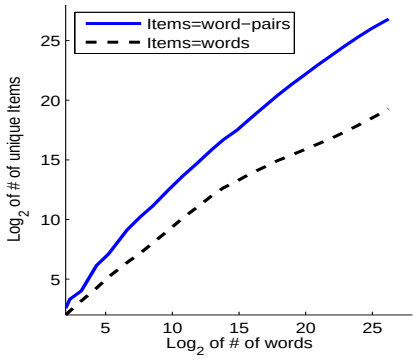


Figure 1: Token Type Curve

Storing only the 118 million unique word pairs in this corpus require 1.9 GB of disk space. This space can be saved by avoiding storing the word pair itself. As a trade-off we are willing to tolerate a small amount of error in the frequency of each word pair. In this paper, we explore sketch techniques, especially the Count-Min Sketch, which approximates the frequency of a word pair in the corpus without explicitly storing the word pairs themselves. It turns out that, in this technique, both updating (adding a new word pair or increasing the frequency of existing word pair) and querying (finding the frequency of a given word pair) are very efficient and can be done in constant time[1].

Counts stored in the CM Sketch can be used to compute various word-association measures like

[1] depend only on one of the user chosen parameters

Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop, pages 17–25,
Los Angeles, California, June 2010.

Pointwise Mutual Information (PMI), and Log-Likelihood ratio. These association scores are useful for other NLP applications like word sense disambiguation, speech and character recognition, and computing semantic orientation of a word. In our work, we use computing semantic orientation of a word using PMI as a canonical task to show the effectiveness of CM Sketch for computing association scores.

In our attempt to advocate the Count-Min sketch to store the frequency of keys (words or word pairs) for NLP applications, we perform both intrinsic and extrinsic evaluations. In our intrinsic evaluation, first we show that low-frequent items are more prone to errors. Second, we show that computing approximate PMI scores from these counts can give the same ranking as Exact PMI. However, we need counters linear in size of stream to achieve that. We use these approximate PMI scores in our extrinsic evaluation of computing semantic orientation. Here, we show that we do not need counters linear in size of stream to perform as good as Exact PMI. In our experiments, by using only 2 billion counters (8GB RAM) we get the same accuracy as for exact PMI scores. The number of these counters is up to 30 times less than the stream size which is a big memory and space gain without any loss of accuracy.

2 Background

2.1 Large Scale NLP problems

Use of large data in the NLP community is not new. A corpus of roughly 1.6 Terawords was used by Agirre et al. (2009) to compute pairwise similarities of the words in the test sets using the MapReduce infrastructure on $2,000$ cores. Pantel et al. (2009) computed similarity between 500 million terms in the MapReduce framework over a 200 billion words in 50 hours using 200 quad-core nodes. The inaccessibility of clusters for every one has attracted the NLP community to use streaming, randomized, approximate and sampling algorithms to handle large amounts of data.

A randomized data structure called Bloom filter was used to construct space efficient language models (Talbot and Osborne, 2007) for Statistical Machine Translation (SMT). Recently, the *streaming algorithm* paradigm has been used to provide memory and space-efficient platform to deal with terabytes of data. For example, We (Goyal et al., 2009) pose language modeling as a problem of finding frequent items in a stream of data and show its effectiveness in SMT. Subsequently, (Levenberg and Osborne, 2009) proposed a randomized language model to efficiently deal with unbounded text streams. In (Van Durme and Lall, 2009b), authors extend Talbot Osborne Morris Bloom (TOMB) (Van Durme and Lall, 2009a) Counter to find the highly ranked k PMI response words given a cue word. The idea of TOMB is similar to CM Sketch. TOMB can also be used to store word pairs and further compute PMI scores. However, we advocate CM Sketch as it is a very simple algorithm with strong guarantees and good properties (see Section 3).

2.2 Sketch Techniques

A sketch is a summary data structure that is used to store streaming data in a memory efficient manner. These techniques generally work on an input stream, i.e. they process the input in one direction, say from left to right, without going backwards. The main advantage of these techniques is that they require storage which is significantly smaller than the input stream length. For typical algorithms, the working storage is sublinear in N, i.e. of the order of $\log^k N$, where N is the input size and k is some constant which is not explicitly chosen by the algorithm but it is an artifact of it.. Sketch based methods use hashing to map items in the streaming data onto a small-space sketch vector that can be easily updated and queried. It turns out that both updating and querying on this sketch vector requires only a constant time per operation.

Streaming algorithms were first developed in the early 80s, but gained in popularity in the late 90s as researchers first realized the challenges of dealing with massive data sets. A good survey of the model and core challenges can be found in (Muthukrishnan, 2005). There has been considerable work on coming up with different sketch techniques (Charikar et al., 2002; Cormode and Muthukrishnan, 2004; Li and Church, 2007). A survey by (Rusu and Dobra, 2007; Cormode and Hadjieleftheriou, 2008) comprehensively reviews the literature.

3 Count-Min Sketch

The Count-Min Sketch (Cormode and Muthukrishnan, 2004) is a compact summary data structure used to store the frequencies of all items in the input stream. The sketch allows fundamental queries

on the data stream such as point, range and inner product queries to be approximately answered very quickly. It can also be applied to solve the finding frequent items problem (Manku and Motwani, 2002) in a data stream. In this paper, we are only interested in point queries. The aim of a point query is to estimate the count of an item in the input stream. For other details, the reader is referred to (Cormode and Muthukrishnan, 2004).

Given an input stream of word pairs of length N and user chosen parameters δ and ϵ, the algorithm stores the frequencies of all the word pairs with the following guarantees:

- All reported frequencies are within the true frequencies by at most ϵN with a probability of at least δ.
- The space used by the algorithm is $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$.
- Constant time of O($\log(\frac{1}{\delta})$) per each update and query operation.

3.1 CM Data Structure

A Count-Min Sketch with parameters (ϵ,δ) is represented by a two-dimensional array with width w and depth d :

$$\begin{bmatrix} \text{sketch[1,1]} & \cdots & \text{sketch[1,w]} \\ \vdots & \ddots & \vdots \\ \text{sketch[d,1]} & \cdots & \text{sketch[d,w]} \end{bmatrix}$$

Among the user chosen parameters, ϵ controls the amount of tolerable error in the returned count and δ controls the probability with which the returned count is not within the accepted error. These values of ϵ and δ determine the width and depth of the two-dimensional array respectively. To achieve the guarantees mentioned in the previous section, we set w=$\frac{2}{\epsilon}$ and d=$\log(\frac{1}{\delta})$. The depth d denotes the number of pairwise-independent hash functions employed by the algorithm and there exists an one-to-one correspondence between the rows and the set of hash functions. Each of these hash functions h_k:$\{1\ldots N\} \rightarrow \{1\ldots w\}$ $(1 \leq k \leq d)$ takes an item from the input stream and maps it into a counter indexed by the corresponding hash function. For example, $h_2(w) = 10$ indicates that the word pair w is mapped to the 10^{th} position in the second row of the sketch array. These d hash functions are chosen uniformly at random from a pairwise-independent family.

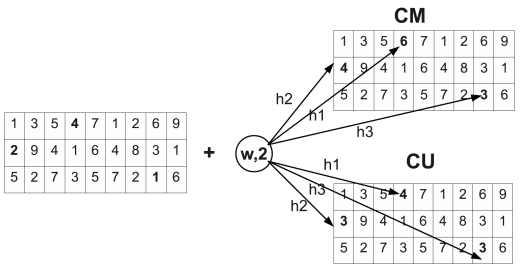


Figure 2: Update Procedure for CM sketch and conservative update (CU)

Initially the entire sketch array is initialized with zeros.

Update Procedure: When a new item (w,c) arrives, where w is a word pair and c is its count[2], one counter in each row, as decided by its corresponding hash function, is updated by c. Formally, $\forall 1 \leq k \leq d$

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

This process is illustrated in Fig. 2 CM. The item (w,2) arrives and gets mapped to three positions, corresponding to the three hash functions. Their counts before update were (4,2,1) and after update they become (6,4,3). Note that, since we are using a hash to map a word into an index, a collision can occur and multiple word pairs may get mapped to the same counter in any given row. Because of this, the values stored by the d counters for a given word pair tend to differ.

Query Procedure: The querying involves finding the frequency of a given item in the input stream. Since multiple word pairs can get mapped into same counter and the observation that the counts of items are positive, the frequency stored by each counter is an overestimate of the true count. So in answering the point query, we consider all the positions indexed by the hash functions for the given word pair and return the minimum of all these values. The answer to Query(w) is:

$$\hat{c} = min_k \text{ sketch[k,}h_k(w)\text{]}$$

Note that, instead of positive counts if we had negative counts as well then the algorithm returns the median of all the counts and the bounds we discussed in Sec. 3 vary. In Fig. 2 CM, for the word pair w it takes the minimum over (6,4,3) and returns 3 as the count of word pair w.

[2]In our setting, c is always 1. However, in other NLP problem, word pairs can be weighted according to recency.

Both update and query procedures involve evaluating d hash functions and a linear scan of all the values in those indices and hence both these procedures are linear in the number of hash functions. Hence both these steps require $O(\log(\frac{1}{\delta}))$ time. In our experiments (see Section 4.2), we found that a small number of hash functions are sufficient and we use d=3. Hence, the update and query operations take only a constant time. The space used by the algorithm is the size of the array i.e. wd counters, where w is the width of each row.

3.2 Properties

Apart from the advantages of being space efficient, and having constant update and constant querying time, the Count-Min sketch has also other advantages that makes it an attractive choice for NLP applications.

- Linearity: given two sketches s_1 and s_2 computed (using the same parameters w and d) over different input streams, the sketch of the combined data stream can be easily obtained by adding the individual sketches in O($\frac{1}{\epsilon} \log \frac{1}{\delta}$) time which is independent of the stream size.

- The linearity is especially attractive because, it allows the individual sketches to be computed independent of each other. Which means that it is easy to implement it in distributed setting, where each machine computes the sketch over a sub set of corpus.

- This technique also extends to allow the deletion of items. In this case, to answer a point query, we should return the median of all the values instead of the minimum value.

3.3 Conservative Update

Estan and Varghese introduce the idea of conservative update (Estan and Varghese, 2002) in the context of networking. This can easily be used with CM Sketch to further improve the estimate of a point query. To update an item, word pair, w 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: $\forall 1 \leq k \leq d$

sketch[k,$h_k(w)$] $\leftarrow \max\{$sketch[k,$h_k(w)$], $\hat{c} +$ 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. Though this is a heuristic, it avoids the unnecessary updates of counter values and thus reduces the error.

The process is also illustrated in Fig. 2CU. When an item "w" with a frequency of 2 arrives in the stream, it gets mapped into three positions in the sketch data structure. Their counts before update were (4,2,1) and the frequency of the item is 1 (the minimum of all the three values). In this particular case, the update rule says that increase the counter value only if its updated value is less than $\hat{c} + 2 = 3$. As a result, the values in these counters after the update become (4,3,3).

However, if the value in any of the counters is already greater than 3 e.g. 4, we cannot attempt to correct it by decreasing, as it could contain the count for other items hashed at that position. Therefore, in this case, for the first counter we leave the value 4 unchanged. The query procedure remains the same as in the previous case. In our experiments, we found that employing the conservative update reduces the Average Relative Error (ARE) of these counts approximately by a factor of 2. (see Section 4.2). But unfortunately, this update prevents deletions and items with negative updates cannot be processed[3].

4 Intrinsic Evaluations

To show the effectiveness of the Count-Min sketch in the context of NLP, we perform intrinsic evaluations. The intrinsic evaluations are designed to measure the error in the approximate counts returned by CMS compared to their true counts. By keeping the total size of the data structure fixed, we study the error by varying the width and the depth of the data structure to find the best setting of the parameters for textual data sets. We show that using conservative update (CU) further improves the quality of counts over CM sketch.

4.1 Corpus Statistics

Gigaword corpus (Graff, 2003) and a copy of web crawled by (Ravichandran et al., 2005) are used to compute counts of words and word pairs. For both the corpora, we split the text into sentences, tokenize and convert into lower-case. We generate words and word pairs (items) over a sliding window of size 14. Unlike previous work (Van Durme

[3]Here, we are only interested in the insertion case.

Corpus	Sub set	Giga word	50% Web	100% Web
Size GB	.15	6.2	15	31
# of sentences (Million)	2.03	60.30	342.68	686.63
# of words (Million)	19.25	858.92	2122.47	4325.03
Stream Size 10 (Billion)	0.25	19.25	18.63	39.05
Stream Size 14 (Billion)	0.23	25.94	18.79	40.00

Table 1: Corpus Description

and Lall, 2009b) which assumes exact frequencies for words, we store frequencies of both the words and word pairs in the CM sketch[4]. Hence, the stream size in our case is the total number of words and word pairs in a corpus. Table 1 gives the characteristics of the corpora.

Since, it is not possible to compute exact frequencies of all word pairs using conventional main memory of 8 GB from a large corpus, we use a subset of 2 million sentences (Subset) from Gigaword corpus for our intrinsic evaluation. We store the counts of all words and word pairs (occurring in a sliding window of length 14) from Subset using the sketch and also the exact counts.

4.2 Comparing CM and CU counts and tradeoff between width and depth

To evaluate the amount of over-estimation in CM and CU counts compared to the true counts, we first group all items (words and word pairs) with same true frequency into a single bucket. We then compute the average relative error in each of these buckets. Since low-frequent items are more prone to errors, making this distinction based on frequency lets us understand the regions in which the algorithm is over-estimating. Average Relative error (ARE) is defined as the average of absolute difference between the predicted and the exact value divided by the exact value over all the items in each bucket.

$$\text{ARE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\text{Exact}_i - \text{Predicted}_i|}{\text{Exact}_i}$$

Where Exact and Predicted denotes values of exact and CM/CU counts respectively; N denotes the number of items with same counts in a bucket.

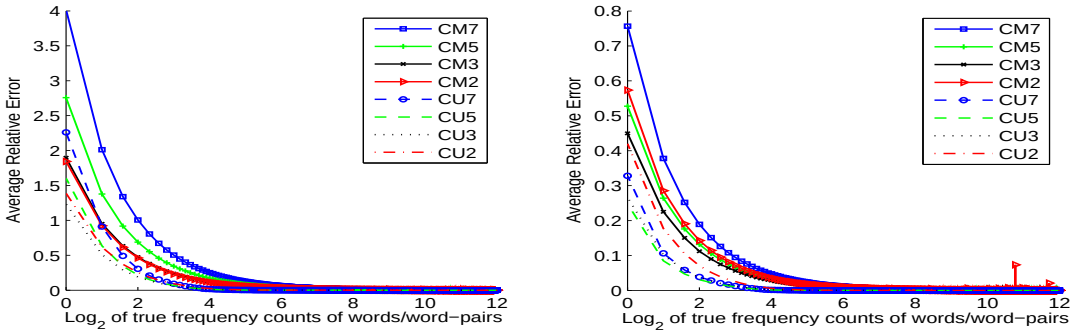
In Fig. 3(a), we fixed the number of counters to 50 million with four bytes of memory per each counter (thus it only requires 200 MB of main memory). Keeping the total number of counters fixed, we try different values of depth (2, 3, 5 and 7) of the sketch array and in each case the width is set to $\frac{50M}{d}$. The ARE curves in each case are shown in Fig. 3(a). There are three main observations: First it shows that most of the errors occur on low frequency items. For frequent items, in almost all the different runs the ARE is close to zero. Secondly, it shows that ARE is significantly lower (by a factor of two) for the runs which use conservative update (CUx run) compared to the runs that use direct CM sketch (CMx run). The encouraging observation is that, this holds true for almost all different (width,depth) settings. Thirdly, in our experiments, it shows that using depth of 3 gets comparatively less ARE compared to other settings.

To be more certain about this behavior with respect to different settings of width and depth, we tried another setting by increasing the number of counters to 100 million. The curves in 3(b) follow a pattern which is similar to the previous setting. Low frequency items are more prone to error compared to the frequent ones and employing conservative update reduces the ARE by a factor of two. In this setting, depth 3 and 5 do almost the same and get lowest ARE. In both the experiments, setting the depth to three did well and thus in the rest of the paper we fix this parameter to three.

Fig. 4 studies the effect of the number of counters in the sketch (the size of the two-dimensional sketch array) on the ARE. Using more number of counters decreases the ARE in the counts. This is intuitive because, as the length of each row in the sketch increases, the probability of collision decreases and hence the array is more likely to contain true counts. By using 200 million counters, which is comparable to the length of the stream 230 million (Table. 1), we are able to achieve almost zero ARE over all the counts including the rare ones[5]. Note that the actual space required to represent the exact counts is almost two times more than the memory that we use here because there are 230 million word pairs and on an average each word is eight characters long and requires eight bytes (double the size of an integer). The summary of this Figure is that, if we want to preserve the counts of low-frequent items accurately, then we need counters linear in size of stream.

[4]Though a minor point, it allows to process more text.

[5]Even with other datasets we found that using counters linear in the size of the stream leads to ARE close to zero $\forall$ counts.



(a) 50M counters (b) 100M counters

Figure 3: Comparing 50 and 100 million counter models with different (width,depth) settings. The notation CMx represents the Count-Min Sketch with a depth of 'x' and CUx represents the CM sketch along with conservative update and depth 'x'.

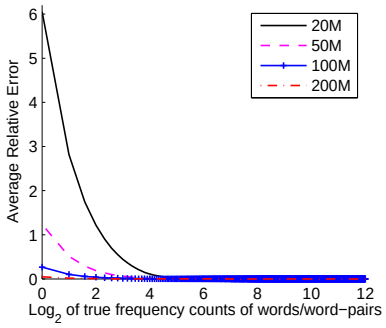


Figure 4: Comparing different size models with depth 3

4.3 Evaluating the CU PMI ranking

In this experiment, we compare the word pairs association rankings obtained using PMI with CU and exact counts. We use two kinds of measures, namely accuracy and Spearman's correlation, to measure the overlap in the rankings obtained by both these approaches.

4.3.1 PointWise Mutual Information

The Pointwise Mutual Information (PMI) (Church and Hanks, 1989) between two words w_1 and w_2 is defined as:

$$PMI(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

Here, $P(w_1, w_2)$ is the likelihood that w_1 and w_2 occur together, and $P(w_1)$ and $P(w_2)$ are their independent likelihoods respectively. The ratio between these probabilities measures the degree of statistical dependence between w_1 and w_2.

4.3.2 Description of the metrics

Accuracy is defined as fraction of word pairs that are found in both rankings to the size of top ranked word pairs.

$$\text{Accuracy} = \frac{|\text{CP-WPs} \cap \text{EP-WPs}|}{|\text{EP-WPs}|}$$

Where CP-WPs represent the set of top ranked K word pairs under the counts stored using the CU sketch and EP-WPs represent the set of top ranked word pairs with the exact counts.

Spearman's rank correlation coefficient (ρ) computes the correlation between the ranks of each observation (i.e. word pairs) on two variables (that are top N CU-PMI and exact-PMI values). This measure captures how different the CU-PMI ranking is from the Exact-PMI ranking.

$$\rho = 1 - \frac{6 \sum d_i^2}{F(F^2 - 1)}$$

Where d_i is the difference between the ranks of a word pair in both rankings and F is the number of items found in both sets.

Intuitively, accuracy captures the number of word pairs that are found in both the sets and then Spearman's correlation captures if the relative order of these common items is preserved in both the rankings. In our experimental setup, both these measures are complimentary to each other and measure different aspects. If the rankings match exactly, then we get an accuracy of 100% and a correlation of 1.

4.3.3 Comparing CU PMI ranking

The results with respect to different sized counter (50, 100 and 200 million) models are shown in Table 2. Table 2 shows that having counters linear

Counters	50M		100M		200M	
Top K	*Acc*	ρ	*Acc*	ρ	*Acc*	ρ
50	.20	-0.13	.68	.95	.92	1.00
100	.18	.31	.77	.80	.96	.95
200	.21	.68	.73	.86	.97	.99
500	.24	.31	.71	.97	.95	.99
1000	.33	.17	.74	.87	.95	.98
5000	.49	.38	.82	.82	.96	.97

Table 2: Evaluating the PMI rankings obtained using CM Sketch with conservative update (CU) and Exact counts

in size of stream ($230M$) results in better ranking (i.e. close to the exact ranking). For example, with $200M$ counters, among the top 50 word pairs produced using the CU counts, we found 46 pairs in the set returned by using exact counts. The ρ score on those word pairs is 1 means that the ranking of these 46 items is exactly the same on both CU and exact counts. We see the same phenomena for $200M$ counters with other Top K values. While both accuracy and the ranking are decent with $100M$ counters, if we reduce the number of counters to say $50M$, the performance degrades.

Since, we are not throwing away any infrequent items, PMI will rank pairs with low frequency counts higher (Church and Hanks, 1989). Hence, we are evaluating the PMI values for rare word pairs and we need counters linear in size of stream to get almost perfect ranking. Also, using counters equal to half the length of the stream is decent. However, in some NLP problems, we are not interested in low-frequency items. In such cases, even using space less than linear in number of counters would suffice. In our extrinsic evaluations, we show that using space less than the length of the stream does not degrades the performance.

5 Extrinsic Evaluations

5.1 Experimental Setup

To evaluate the effectiveness of CU-PMI word association scores, we infer semantic orientation (S0) of a word from CU-PMI and Exact-PMI scores. Given a word, the task of finding the SO (Turney and Littman, 2002) of the word is to identify if the word is more likely to be used in positive or negative sense. We use a similar framework as used by the authors[6] to infer the SO. We take the seven positive words (good, nice, excellent, positive, fortunate, correct, and superior) and the negative words (bad, nasty, poor, negative, unfortunate, wrong, and inferior) used in (Turney and Littman, 2002) work. The SO of a given word is calculated based on the strength of its association with the seven positive words, and the strength of its association with the seven negative words. We compute the SO of a word "w" as follows:

$$\text{SO-PMI(w)} = PMI(+,w) - PMI(-,w)$$

$$\text{PMI(+,w)} = \sum_{p \in Pwords} \log \frac{\text{hits}(p,w)}{\text{hits}(p) \cdot \text{hits}(w)}$$

$$\text{PMI(-,w)} = \sum_{n \in Nwords} \log \frac{\text{hits}(n,w)}{\text{hits}(n) \cdot \text{hits}(w)}$$

Where, Pwords and Nwords denote the seven positive and negative prototype words respectively.

We compute SO score from different sized corpora (Section 4.1). We use the General Inquirer lexicon[7] (Stone et al., 1966) as a benchmark to evaluate the semantic orientation scores similar to (Turney and Littman, 2002) work. Words with multiple senses have multiple entries in the lexicon, we merge these entries for our experiment. Our test set consists of 1619 positive and 1989 negative words. Accuracy is used as an evaluation metric and is defined as the fraction of number of correctly identified SO words.

$$\text{Accuracy} = \frac{\text{Correctly Identified SO Words} * 100}{\text{Total SO words}}$$

5.2 Results

We evaluate SO of words on three different sized corpora: Gigaword (GW) 6.2GB, GigaWord + 50% of web data (GW+WB1) 21.2GB and GigaWord + 100% of web data (GW+WB2) 31GB. Note that computing the exact counts of all word pairs on these corpora is not possible using main memory, so we consider only those pairs in which one word appears in the prototype list and the other word appears in the test set.

We compute the exact PMI (denoted using Exact) scores for pairs of test-set words w_1 and prototype words w_2 using the above data-sets. To compute PMI, we count the number of hits of individual words w_1 and w_2 and the pair (w_1,w_2) within a sliding window of sizes 10 and 14 over these data-sets. After computing the PMI scores, we compute SO score for a word using SO-PMI equation from Section 5.1. If this score is positive, we predict the word as positive. Otherwise, we predict it as

[6] We compute this score slightly differently. However, our main focus is to show that CU-PMI scores are useful.

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

Model		Accuracy window 10			Accuracy window 14		
#of counters	*Mem. Usage*	*GW*	*GW+WB1*	*GW+WB2*	*GW*	*GW+WB1*	*GW+WB2*
Exact	n/a	**64.77**	**75.67**	**77.11**	**64.86**	**74.25**	**75.30**
500M	2GB	62.98	71.09	72.31	63.21	69.21	70.35
1B	4GB	62.95	73.93	75.03	63.95	72.42	72.73
2B	**8GB**	**64.69**	**75.86**	**76.96**	**65.28**	**73.94**	**74.96**

Table 3: Evaluating Semantic Orientation of words with different # of counters of CU sketch with increasing amount of data on window size of 10 and 14. Scores are evaluated using Accuracy metric.

negative. The results on inferring correct SO for a word w with exact PMI (Exact) are summarized in Table 3. It (the second row) shows that increasing the amount of data improves the accuracy of identifying the SO of a word with both the window sizes. The gain is more prominent when we add 50% of web data in addition to Gigaword as we get an increase of more than 10% in accuracy. However, when we add the remaining 50% of web data, we only see an slight increase of 1% in accuracy[8]. Using words within a window of 10 gives better accuracy than window of 14.

Now, we use our CU Sketches of 500 million ($500M$), 1 billion ($1B$) and 2 billion ($2B$) counters to compute CU-PMI. These sketches contain the number of hits of all words/word pairs (not just the pairs of test-set and prototype words) within a window size of 10 and 14 over the whole dataset. The results in Table 3 show that even with CU-PMI scores, the accuracy improves by adding more data. Again we see a significant increase in accuracy by adding 50% of web data to Gigaword over both window sizes. The increase in accuracy by adding the rest of the web data is only 1%.

By using 500M counters, accuracy with CU-PMI are around 4% worse than the Exact. However, increasing the size to $1B$ results in only 2 % worse accuracy compared to the Exact. Going to $2B$ counters (8 GB of RAM), results in accuracy almost identical to the Exact. These results hold almost the same for all the data-sets and for both the window sizes. The increase in accuracy comes at expense of more memory Usage. However, 8GB main memory is not large as most of the conventional desktop machines have this much RAM. The number of $2B$ counters is less than the length of stream for all the data-sets. For GW, GW+WB1 and GW+WB2, $2B$ counters are 10, 20 and 30 times smaller than the stream size. This shows that using counters less than the stream length does not degrade the performance.

[8]These results are similar to the results reported in (Turney and Littman, 2002) work.

The advantage of using Sketch is that it contains counts for all words and word pairs. Suppose we are given a new word to label it as positive or negative. We can find its exact PMI in two ways: First, we can go over the whole corpus and compute counts of this word with positive and negative prototype words. This procedure will return PMI in time needed to traverse the whole corpus. If the corpus is huge, this could be too slow. Second option is to consider storing counts of all word pairs but this is not feasible as their number increases rapidly with increase in data (see Fig. 1). Therefore, using a CM sketch is a very good alternative which returns the PMI in constant time by using only 8GB of memory. Additionally, this Sketch can easily be used for other NLP applications where we need word-association scores.

6 Conclusion

We have explored the idea of the CM Sketch, which approximates the frequency of a word pair in the corpus without explicitly storing the word pairs themselves. We used the idea of a conservative update with the CM Sketch to reduce the average relative error of its approximate counts by a factor of 2. It is an efficient, small-footprint method that scales to at least 37 GB of web data in just 2 billion counters (8 GB main memory). In our extrinsic evaluations, we found that CU Sketch is as effective as exact PMI scores.

Word-association scores from CU Sketch can be used for other NLP tasks like word sense disambiguation, speech and character recognition. The counts stored in CU Sketch can be used to construct small-space randomized language models. In general, this sketch can be used for any application where we want to query a count of an item.

Acknowledgments

We thank the anonymous reviewers for helpful comments. This work is partially funded by NSF grant IIS-0712764 and Google Research Grant Grant for Large-Data NLP.

References

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 Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Moses Charikar, Kevin Chen, and Martin Farach-colton. 2002. Finding frequent items in data streams.

K. Church and P. Hanks. 1989. Word Association Norms, Mutual Information and Lexicography. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, Canada, June.

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*.

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

Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. 2009. Streaming for large scale NLP: Language modeling. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

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

Abby Levenberg and Miles Osborne. 2009. Stream-based randomised language models for SMT. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 756–764, Singapore, August. Association for Computational Linguistics.

Ping Li and Kenneth W. Church. 2007. A sketch algorithm for estimating two-way and multi-way associations. *Comput. Linguist.*, 33(3).

G. S. Manku and R. Motwani. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*.

S. Muthukrishnan. 2005. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2).

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. 2009. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 938–947, Singapore, August. Association for Computational Linguistics.

S. Patwardhan and E. Riloff. 2006. Learning Domain-Specific Information Extraction Patterns from the Web. In *Proceedings of the ACL 2006 Workshop on Information Extraction Beyond the Document*.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

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

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 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)*.

Peter D. Turney and Michael L. Littman. 2002. Unsupervised learning of semantic orientation from a hundred-billion-word corpus. *CoRR*, cs.LG/0212012.

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*, pages 1574–1579.

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