ABSTRACT

Title of Dissertation: Streaming and Sketch Algorithms for Large Data NLP

Amit Goyal, Doctor of Philosophy, 2013

Dissertation directed by: Dr. Hal Daumé III
Department Computer Science

The availability of large and rich quantities of text data is due to the emergence of the World Wide Web, social media, and mobile devices. Such vast data sets have led to leaps in the performance of many statistically-based problems. Given a large magnitude of text data available, it is computationally prohibitive to train many complex Natural Language Processing (NLP) models on large data. This motivates the hypothesis that simple models trained on big data can outperform more complex models with small data. My dissertation provides a solution to effectively and efficiently exploit large data on many NLP applications.

Datasets are growing at an exponential rate, much faster than increase in memory. To provide a memory-efficient solution for handling large datasets, this dissertation show limitations of existing streaming and sketch algorithms when applied to canonical NLP problems and proposes several new variants to overcome those shortcomings. Streaming and sketch algorithms process the large data sets in one pass and represent a large data set with a compact summary, much smaller than the full size of the input. These algorithms can easily be implemented in a distributed setting and provide a solution that is both *memory- and time-efficient*. However, the memory and time savings come at the expense of *approximate* solutions. In this dissertation, I demonstrate that approximate solutions achieved on large data are comparable to exact solutions on large data and outperform exact solutions on smaller data.

I focus on many NLP problems that boil down to tracking many statistics, like storing approximate counts, computing approximate association scores like pointwise mutual information

(PMI), finding frequent items (like n-grams), building streaming language models, and measuring distributional similarity. First, I introduce the concept of approximate streaming large-scale language models in NLP. Second, I present a novel variant of the Count-Min sketch that maintains approximate counts of all items. Third, I conduct a systematic study and compare many sketch algorithms that approximate count of items with focus on large-scale NLP tasks. Last, I develop fast large-scale approximate graph (FLAG), a system that quickly constructs a large-scale approximate nearest-neighbor graph from a large corpus.

Streaming and Sketch Algorithms for Large Data NLP

by

Amit Goyal

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2013

Advisory Committee:
Dr. Hal Daumé III, Chair/Advisor
Prof. Min Wu, Dean's Representative
Dr. Suresh Venkatasubramanian
Dr. Amol Deshpande, Depts Representative
Dr. Jordan Boyd-Graber

Dedication

To my parents, siblings, relatives, and friends.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I am eternally grateful to my advisor Hal Daumé III. This thesis would not have been possible without his invaluable advice and support over the past six years. He has always given me invaluable opportunities to work on challenging and extremely interesting problems. He has always been patient and resourceful while I was stuck on a research problem. He has given me right amount of freedom to grow as an independent researcher.

I would also like to thank Min Wu, Suresh Venkatasubramanian, Amol Deshpande, and Jordan Boyd-Graber for serving on my thesis committee. I sincerely appreciate their suggestions and support.

I learned so much about research from Ellen Riloff, Suresh Venkatasubramanian, Graham Cormode, Alex Berg, Tamara Berg, and Srinivas Bangalore. I will always be grateful to them for teaching me to become an effective researcher.

Thanks are due to Marjorie Freedman, Elizabeth Boschee, Ryan Gabbard, Stu Black, Ilana Heintz, Vasin Punyakanok, Ralph Weischedel, and others at Raytheon BBN Technologies for their guidance and support during my internship.

Ideas in this work were greatly influenced by many useful discussions with colleagues in the NLP Research Group at the University of Utah. Special thanks to Piyush Rai, Avishek Saha, Nathan Gilbert, Ruihong Huang, Adam Teichert, and others for their support and many useful discussions.

My collegaues, postdocs, and professors at the Laboratory for Computational Linguistics and Information Processing have broadened my knowledge about natural language processing, machine learning and made my graduate life enjoyable. I would like to especially thank Philip Resnik, Doug Oard, Kristy Hollingshead Seitz, Taesun Moon, Jagadeesh Jagarlamudi, Jiarong Jiang, Abhishek Kumar, Arvind Agarwal, Raul D. Guerra, Gregory Sanders, Vladimir Eidelman, Ke Wu, Viet-An Nguyen, Ferhan Ture, Ke Zhai, Yuening Hu, and others for their support and many useful conversations.

Last but not least, I would like to thank my loving parents, siblings, relatives, and friends for their patience and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NLP	Natural Language Processing
SMT	statistical machine translation
AS	Association Scores
PMI	pointwise mutual information
LLR	log likelihood ratio
SO	semantic orientation
CBC	clustering by committee
TOMB	Talbot Osborne Morris Bloom
ρ	Spearman's rank correlation
ARE	Average Relative Error
MRE	Mean Relative Error
CM	Count-Min
CM-CU	Count-Min with conservative update
COUNT	Count
COUNT-CU	Count with conservative update
SBF	Spectral Bloom Filters
SBF-CU	Spectral Bloom Filters with conservative update
CMM	Count-mean-min
CMM-CU	Count-mean-min with conservative update
LCU-WS	Lossy counting with conservative update I
LCU-SWS	Lossy counting with conservative update II
LCU-1	Lossy counting with conservative update III
LCU-ALL	Lossy counting with conservative update IV
FLAG	System for Fast Large-scale Approximate Graph construction
Lsh	locality sensitive hashing
Pleb	point location in equal balls
Irp	independent random projections
Fast-Pleb	fast point location in equal balls
EP	Europarl data
GW	Gigaword data
nyt	The New York Times Newswire Service (part of Gigaword)
afe	The Agence France Press English Service (part of Gigaword)
apw	The Associated Press Worldstream English Service (part of Gigaword)
xie	The Xinhua News Agency English Service (part of Gigaword)
EAN	Combination of EP + afe + nyt data
GWB50	Gigaword + 50% of web data
GWB100	Gigaword + 100% of web data

Chapter 1

Introduction

Big data is ubiquitous. In everyday life, researchers and engineers are dealing with big data from Web usage, social media and mobile usage in form of multiple modalities (like text, speech, image, and video). This situation demands algorithms that can process massive data in a memory- and time-efficient manner. Streaming and sketch algorithms provide a general framework for effectively and efficiently managing large-scale datasets. In this dissertation, I show limitations of existing streaming and sketch algorithms when applied to canonical NLP problems and propose several new variants to overcome those shortcomings. These new variants effectively and efficiently manage *memory* and *time* to deal with large data for Natural Language Processing (NLP) problems.

1.1 Large Data in NLP

Since the emergence of the World Wide Web, social media and mobile devices, we have ever larger and richer quantities of text data. Such vast corpora have led to leaps in the performance of many language-based tasks. For example, language modeling [10, 103, 115, 116], sentiment analysis [64, 151, 110, 140, 164, 102], measuring distributional similarity [80, 51, 3, 152], entity set expansion [112, 117, 160] and statistical machine translation (SMT) [13, 14, 165, 81]. Figure 1.1 shows some examples of NLP applications where large data has been demonstrated to be beneficial.

However, working on large data problems comes with its own challenges: principally, how to effectively and efficiently manage *memory* and *time* to deal with large datasets. This challenge has motivated the NLP community to use commodity clusters. For example, Brants et al. [10] used 1500 machines for one day to compute the relative frequencies of n-grams from 1.8TB of Web data. Their resulting model contained 300 million unique n-grams. In another line of research, a corpus of roughly 1.6 Terawords was used by Agirre et al. [3] to compute pairwise similarities of words in test sets using the MapReduce infrastructure on $2,000$ cores. Pantel et al. [112] computed similarity between 500 million terms in the MapReduce framework over 200 billion words in 50 hours using 200 quad-core nodes.

However, an average desktop user (who today only has 8 GB of memory) does not have access to these clusters and thus cannot use large datasets. Instead, an average user works with a sub-sample of the large data and does not reap the benefits of large datasets. This problem has motivated this dissertation and other NLP researchers to use streaming, sketch, randomized, and approximate algorithms to handle large amounts of data [56, 81, 157].

In this dissertation, I exploit existing algorithms and propose variants thereof to provide memory-efficient solutions for an average user to exploit large data. At the same time, these algorithms can easily be implemented in the distributed setting and provide memory- and time-efficient solution for commodity cluster users.

However, the *memory and time savings* come at the expense of *approximate* solutions; though in this dissertation I demonstrate that approximate solutions achieved on large data are comparable

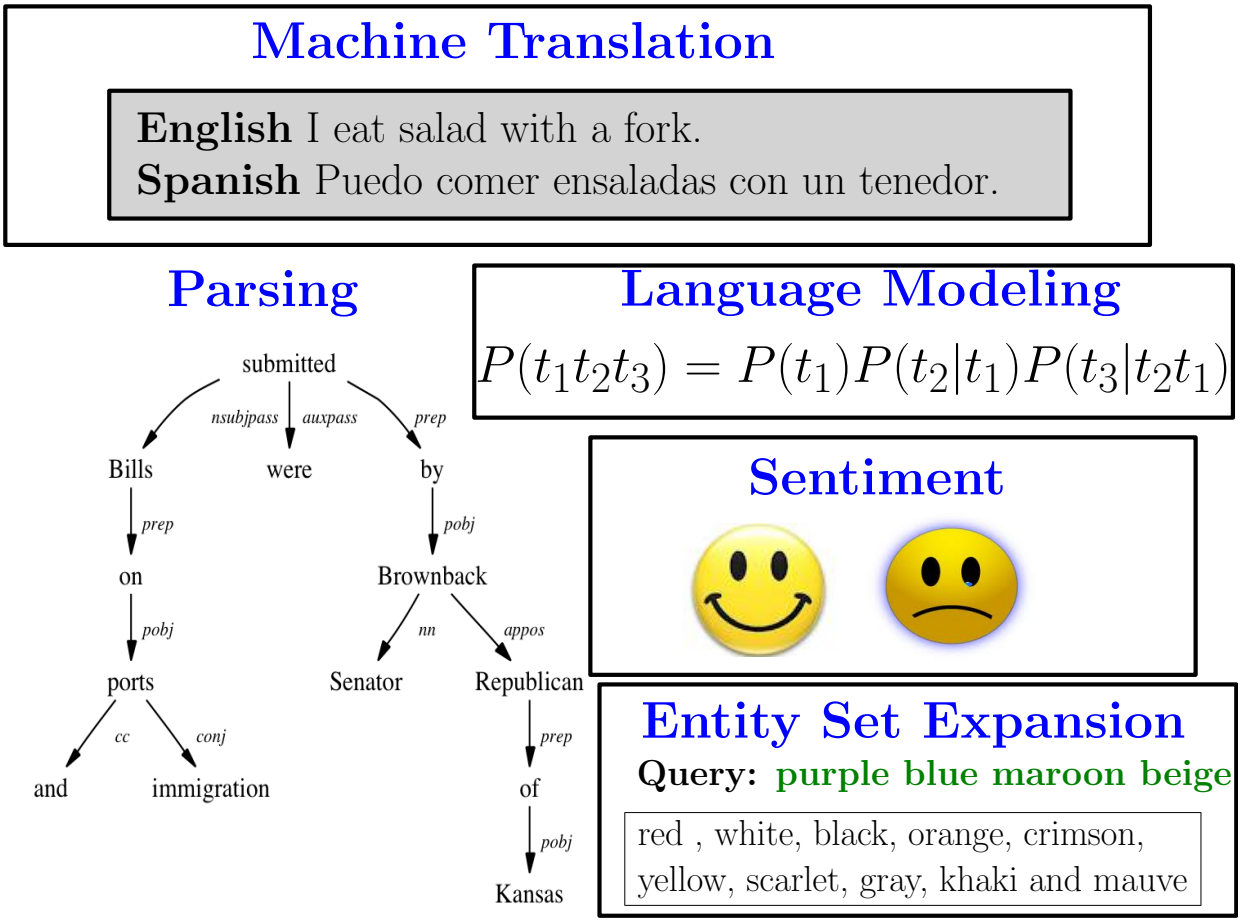


Figure 1.1: Examples of NLP applications where large data is beneficial.

to exact solutions on large data, and outperform exact solutions on smaller data.

1.2 Streaming and Sketch Algorithms

Streaming algorithms process large datasets in only a few passes (typically just one) and represent a large dataset with a compact summary, typically much smaller than the full size of the input. Many of these streaming algorithms are directed at computing statistics of frequency distributions that are too big to be stored in main memory. These constraints mean that an algorithm produces an approximate solution instead of an exact one. However, typically the speed and memory *gains* of these algorithms are substantially larger than the loss in accuracy of the system.

Sketch algorithms refer to a class of streaming algorithm that represents a large dataset 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.

Streaming and sketch algorithms can approximate frequency distributions in many ways. The two schemes that are relevant with respect to this dissertation are : (1) To find the approximate counts of high-frequency items (heavy hitters) in a input stream [27]. (2) To store the approximate

counts of all items with a compact summary using the idea of hashing [26].

1.3 Streaming and Sketch Algorithms for NLP

Given the magnitude of large text data available, it is computationally intensive to train complex NLP models on large data. This motivates the hypothesis that simple models trained on big data can outperform more complex models on small data. Prior work by Banko and Brill [**?**] demonstrated that it is better to direct efforts toward increasing the size of annotated training corpora, while deemphasizing the focus on comparing different learning techniques trained only on small training corpora. In another line of research for large-scale language modeling, Brants et al. [10] used a simple smoothing technique, named "stupid backoff", and showed that, with large amounts of data, stupid backoff outperforms more complex smoothing techniques [76, 20].

In this dissertation I exploit the fact that many NLP tasks rely on tracking many statistics from large corpora. We can compute these statistics approximately in a memory- and time-efficient manner using streaming and sketch algorithms. For example, storing frequency counts [144, 155, 53], computing association scores like pointwise mutual information [86, 156, 53], finding frequent items (like n-grams) [56], building streaming language models [142, 82], and distributional similarity [121, 157].

All these problems ultimately depend on approximate counts of items (such as n-grams, word pairs and word-context pairs). Thus, one of the *paramount* contributions of this dissertation is focused on solving this central problem in the context of NLP applications.

1.4 Contributions and an Overview of This Thesis

The dissertation is presented in three parts. The first part, comprising the next chapter, reviews existing variants of streaming and sketch algorithms and introduces several novel variants. The second part, comprising Chapter 3 through Chapter 6, demonstrates the applicability of streaming and sketch algorithms for various large data NLP tasks. The third and final part of the dissertation concludes and presents preliminary results on extending Fast Large-Scale Approximate Graph (FLAG) construction (Chapter 6) for other NLP applications.

The chapters in this dissertation are organized as:

Part I: Streaming and Sketch Algorithms

Chapter 2: This chapter provides the required background on streaming and sketch algorithms. Here, I review existing as well as introduce novel variants of these algorithms. The algorithms presented in this chapter are exploited in chapters 3, 4, 5 and 6 to solve many large data NLP problems.

Part II: Streaming and Sketch Algorithms for Large Data NLP

Chapter 3: I present language modeling as a canonical example of a large-scale task that requires computing frequency counts of high-order n-grams. There has been prior work on maintaining approximate counts for higher-order language models (LMs). Talbot et al. [143, 142] operates under the model that the goal is to store a compressed representation of a disk-resident table of counts and use this compressed representation to answer count queries approximately. There are two difficulties with scaling the above approaches as the order of the LM increases. First, the

computation time to build the database of counts increases rapidly. Second, the initial disk storage required to maintain these counts, prior to building the compressed representation is enormous.

As a first contribution,[1] I propose a method [56] that solves both of these problems. I do this by making use of the *streaming algorithm* paradigm [105]. Working under the assumption that multiple-GB models are infeasible, my goal is, instead of estimating a large model and then compressing it, directly estimating a small model. I use a deterministic streaming algorithm [93] that computes approximate frequency counts of *frequently occurring n-grams*. This is the *first* work that introduced the concept of approximate streaming large-scale language models in NLP. I use these counts directly as features in a statistical machine translation (SMT) system and propose a direct way to integrate these features into an SMT decoder. Experiments show that, compared to using count-based or entropy-based pruning counts, directly storing approximate counts of frequent 5-grams gives equivalent SMT performance and dramatically reduces memory usage by getting rid of pre-computing a large model.

Chapter 4: In this Chapter, I focus my attention on maintaining approximate counts of *all items* instead of just focusing on storing frequent items. As a second contribution,[2] I explore sketch techniques, especially the Count-Min sketch [28] to build a single sketch model [53] and show its effectiveness on three important NLP tasks: (1) Predicting the semantic orientation of words [151], (2) Distributional approaches for word similarity [3], and (3) Unsupervised dependency parsing [22] with a little linguistic knowledge.

I explore Count-Min (CM) [28] sketch with conservative update [44] (CM-CU) to address the issue of *efficient storage* of large data. Using conservative update with a CM sketch reduces the average relative error of its approximate counts by a factor of 1.5. The CM sketch stores counts of *all* word pairs within a *bounded space*. Storage space is reduced by *approximating* the frequency of word pairs in the corpus without explicitly storing the word pairs themselves. 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 constant-time operations, making the sketch an efficient online storage data structure for large data. Sketches are *scalable* and can easily be implemented in the distributed setting. I use CM sketch to store counts of word pairs within a window of size 7 words over differently sized corpora and store exact counts of words in a hash table (number of unique words pairs >> unique words). Approximate pointwise mutual information (PMI) and log likelihood ratio (LLR) scores are computed using these approximate counts and are applied to solve three NLP tasks. Experiments demonstrate that, on all three tasks, my approach achieves performance comparable to computing exact word pair counts and to the state-of-the-art. My method scales to 49 GB of Web data using a bounded space of 2 billion 32-bit counters (8 GB memory).

Chapter 5: It is unclear from my work discussed in Chapter 4, why CM-CU is preferred over several other sketches [18, 28]. To investigate this question, as a third contribution,[3] I conduct a systematic study to compare many (10) sketch techniques (which stores approximate counts) for large-scale NLP tasks. This is the *first* comparative study of sketches for NLP problems.

The above work includes three contributions:

1. I propose novel variants of existing sketches by extending the idea of *conservative update* to them. I propose Count sketch [18] with conservative update (COUNT-CU), Count-

[1]This work [56] was presented at North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT) 2009 conference.

[2]This work [53] was presented at Empirical Methods in Natural Language Processing (EMNLP) 2011 conference.

[3]The empirical comparison study for sketches [52] was published at Empirical Methods in Natural Language Processing (EMNLP) 2012 conference.

mean-min sketch with conservative update (CMM-CU) and variants of Lossy Counting with conservative update.[4] The motivation behind proposing new sketches is inspired by the success of Count-Min sketch with conservative update (presented in Chapter 4 [53]).

2. I 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. I also evaluate their performance via pointwise mutual information and log likelihood ratio.

3. I use sketches to solve three important NLP problems (finding semantic orientation, measuring distributional similarity, and unsupervised dependency parsing). 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.

Chapter 6: In previous chapters, I demonstrated the effectiveness of streaming and sketch algorithms [105, 26] to provide memory- and time-efficient solutions to many individual NLP tasks. The chapter we are in, I focus on a more general and broad NLP problem: constructing large nearest-neighbor graphs in a memory- and time-efficient manner. Such graphs are necessary for a wide variety of NLP tasks, including constructing polarity lexicons based on lexical graphs from WordNet [120], constructing polarity lexicons from Web data [159], unsupervised part-of-speech tagging using label propagation [30] and detecting visual text [36]. The latter three approaches construct nearest-neighbor graphs between word pairs by computing the nearest neighbors of words in large corpora. These nearest neighbors form the edges of the graph, with weights given by the distributional similarity [152] between terms. Unfortunately, computing the distributional similarity between all words in a large vocabulary is computationally and memory intensive when working with large amounts of data [112]. This bottleneck is typically addressed by means of commodity clusters. For example, Pantel et al. [112] compute distributional similarity between 500 million terms over 200 billion words in 50 hours using 100 quad-core nodes, explicitly storing a sparse similarity matrix between 500 million terms.

To construct large nearest-neighbor graphs, I propose Fast Large-Scale Approximate Graph (FLAG) construction, a system that quickly construct such graphs from large text corpora. To build this system, I exploit recent developments in the area of approximation, randomization and streaming algorithms for large-scale NLP problems [121, 56, 81]. More specifically I exploit work on online-PMI [156]; Locality Sensitive Hashing (Lsh) [19] for computing word-pair similarities from large text collections [121, 157]; random projections [108, 1, 66, 87]; and feature hashing [161, 132, 59] and improve on previous work done on Pleb (Point Location in Equal Balls) [67, 19].

The above part of dissertation includes three main contributions:[5] (1) I present fast large-scale approximate graph (FLAG), a novel system that returns approximate nearest neighbors by exploiting a variant of Count-Min (CM-CU) sketch in a distributed framework. (2) I propose a distributed online-PMI algorithm and our experiment shows that it scales to 110 GB of Web data with 866 million sentences in less than 2 days using 100 quad-core nodes. (3) I propose a novel variant of Point Location in Equal Balls, Fast-Pleb, and demonstrate empirically that the variant is two orders of magnitude faster with comparable recall.

I focus on generating fast large-scale approximate graphs for NLP problems. However, the algorithms proposed will be applicable to many other application areas like computer vision, machine learning, knowledge mining, social media, and information retrieval.

[4]The work [54], which proposed variants of Lossy counting with conservative update, was published at Association for the Advancement of Artificial Intelligence (AAAI) 2011 conference.

[5]This work was published at Empirical Methods in Natural Language Processing (EMNLP) 2012 conference.

Part III: Future Work

Chapter 7: This chapter provides future directions and concludes this dissertation. I describe four extensions of FLAG. First, I present different kinds of contexts that can be used for construction of FLAG. Second, I describe heuristics that can be used to remove noise from approximate nearest neighbor graphs. Third, I describe an approximate version of Clustering By Committee (CBC) [113] as a canonical example for several clustering methods. Last, I describe the algorithmic techniques that can be used to improve the efficiency and effectiveness of FLAG. I then conclude the dissertation by summarizing important contributions and providing future directions.

Chapter 2

Streaming and Sketch Algorithms

This chapter provides the required background on streaming and sketch algorithms. Here, I review existing as well as novel variants of these algorithms proposed by me. These algorithms are exploited in chapters 3, 4, 5, and 6 to solve many large data NLP problems.

2.1 Streaming

Consider an algorithm that reads the input from a read-only *stream* from left to right, with no ability to go back to the input that it has already processed. This algorithm has working storage that it can use to store parts of the input or other intermediate computations. However, (and this is a critical constraint), this working storage space is significantly smaller than the input stream length. For typical algorithms, the storage size is of the order of $\log^k N$, where N is the input size and k is some constant. An example of streaming algorithm is visualized in Figure 2.1.

Stream 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 datasets. A good survey of the model and core challenges can be found in [105]. There has been considerable work on the problem of identifying high-frequency items (items with frequency above a threshold), and a detailed review of these methods is beyond the scope of this article. A new survey by [27] comprehensively reviews the literature.

2.2 Lossy Counting

In this section, we review the lossy counting algorithm proposed in the context of database management [93]. The lossy counting is a streaming algorithm whose goal is is to find the approximate counts of high-frequency items (heavy hitters) in a input stream. Fix parameters $s \in (0, 1)$, and $\gamma \in (0, 1), \gamma \ll s$. Lossy counting's goal is to approximately find all items with frequency at least sN. For an input stream of items of length N, the algorithm outputs a set of items (and frequencies) and guarantees the following:

- All items with frequencies exceeding sN are output (*no false negatives*).
- No item with frequency less than $(s - \gamma)N$ is output (*few false positives*).
- All reported frequencies are less than the true frequencies by at most γN (*close-to-exact frequencies*).
- The space used by the algorithm is $O(\frac{1}{\gamma} \log \gamma N)$.

A simple example illustrates these properties. Let us fix $s = 0.01, \gamma = 0.001$. Then the algorithm guarantees that all items with frequency at least 1% will be returned, no element with

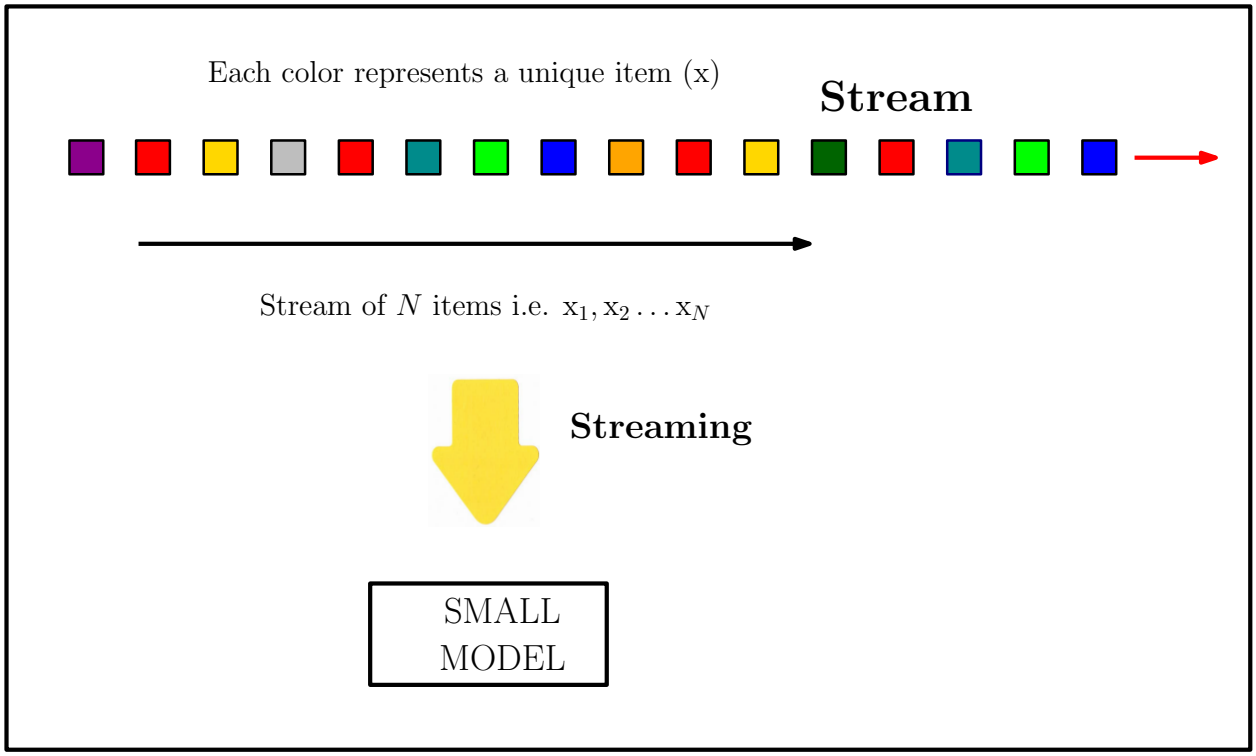


Figure 2.1: Streaming algorithms process the input in one pass and generates a small model (that is summary of the stream).

frequency less than 0.9% will be returned, and all frequencies will be no more than 0.1% away from the true frequencies. The space used by the algorithm is $O(\log N)$, which can be compared to the much larger (close to N) space needed to store the initial frequency counts. In addition, the algorithm runs in linear time by definition, requiring only one pass over the input. Note that there might be $\frac{1}{\gamma}$ elements with frequency at least γN, and so the algorithm uses optimal space (up to a logarithmic factor).

In the following sub-sections, I review the original algorithm [93] proposed by Manku and Motwani and a variant of original algorithm presented by Manku in a presentation [92] of the paper. Both the algorithms provide the guarantees discussed above.

2.2.1 The original Algorithm

We present a high-level overview of the algorithm; for more details, the reader is referred to [93]. The algorithm proceeds by conceptually dividing the stream into *epochs*, each containing $1/\gamma$ elements. Note that there are γN epochs. Each such epoch has an ID, starting from 1. The algorithm maintains a list of tuples[1] of the form (e, f, Δ), where e is an item, f is its reported frequency, and Δ is the maximum error in the frequency estimation. While the algorithm reads items associated with the current epoch, it does one of two things: if the new element e is contained in the list of tuples, it merely increments the frequency count f. If not, it creates a new tuple of the form $(e, 1, T-1)$, where T is the ID of the current epoch.

After each epoch, the algorithm "cleans house" by eliminating tuples whose maximum true

[1]We use hash tables to store tuples; however smarter data structures like suffix trees could also be used.

frequency is small. Formally, if the epoch that just ended has ID T, then the algorithm deletes all tuples satisfying condition $f + \Delta \leq T$. Since $T \leq \gamma N$, this ensures that no low-frequency tuples are retained. When all elements in the stream have been processed, the algorithm returns all tuples (e, f, Δ) where $f \geq (s - \gamma)N$. In practice, however we do not care about s and return all tuples. At a high level, the reason the algorithm works is that if an element has high frequency, it shows up more than once each epoch, and so its frequency gets updated enough to stave off elimination.

2.2.2 The variant of original Algorithm

Here, I review a brief description of a variant of original algorithm presented by Manku in a presentation of the [92] paper. The algorithm proceeds by conceptually dividing the stream into *epochs*, each containing $1/\gamma$ items. Note that there are γN epochs. 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 epoch 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 $count \geq (s - \gamma)N$.

2.3 Sketches

In this chapter, I review existing sketch algorithms from the literature, and propose novel variants based on the idea of conservative update [44]. The term 'sketch' refers to a class of streaming algorithm that represents a large dataset 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, these 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. They can also be applied to solve the finding frequent items problem [27] in a data stream. In this dissertation, I focus on point queries, which ask for the (approximate) count of a given item. For more details, the reader should refer to Cormode's survey [26] on "Sketch Techniques for Approximate Query Processing".

The algorithms I 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 sketch of count information. The depth d denotes the number of hash functions employed by the sketch algorithms and w denotes the range of each hash function. A sketch with parameters (ϵ,δ) is represented with width w and depth d:

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

Sketch Operations: Every sketch has two operations: UPDATE and QUERY to update and estimate the count of an item. They all guarantee *constant time* operation ($O(\log(\frac{1}{\delta}))$ 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

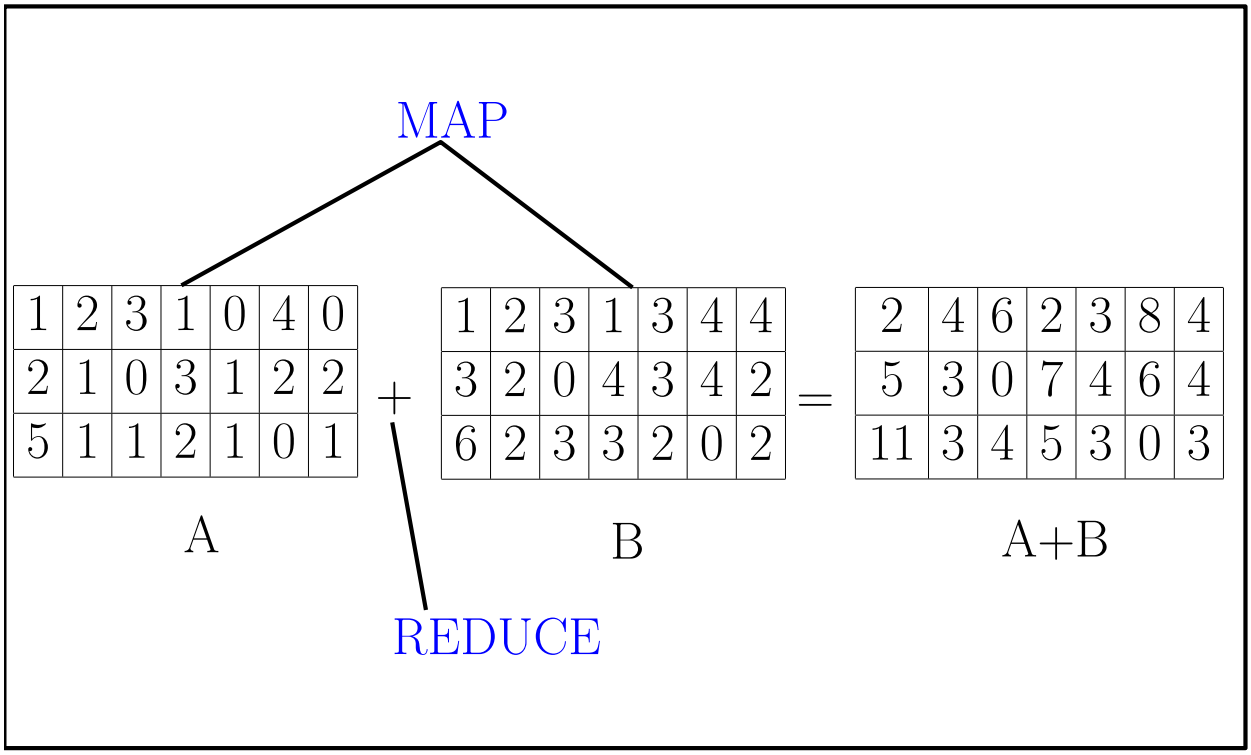


Figure 2.2: Example showing the COMBINE operation of sketches. Sketch A and B are constructed over individual data-sets ("Map" step in MapReduce framework). Sketch over combined data-sets is constructing by element-wise addition of individual sketches ("Reduce" step in MapReduce framework).

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. An example illustrates the COMBINE operation in Figure 2.2.

2.3.1 Existing sketch algorithms

This section describes sketches from the literature:

2.3.1.1 Count-Min sketch (CM)

The CM [28] sketch has been used effectively for many large scale problems across several areas, such as security [128], machine learning [132, 2], privacy [39], and NLP [53]. 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:

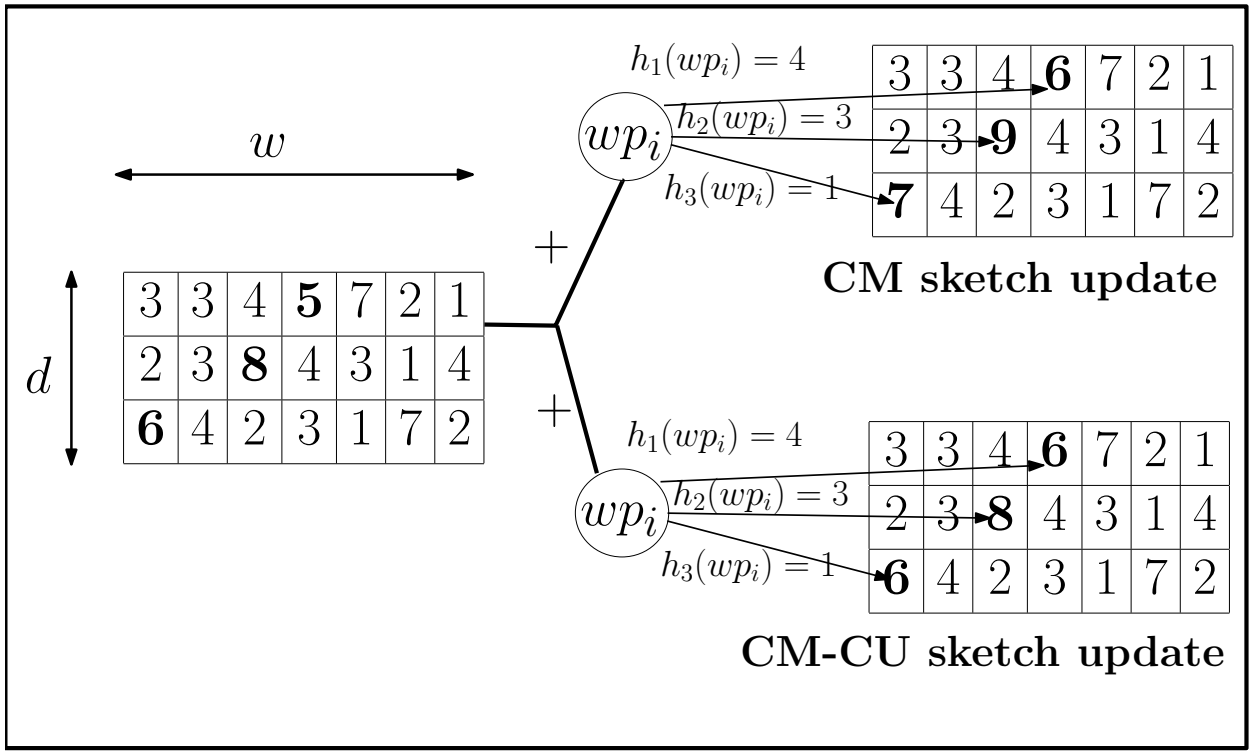


Figure 2.3: Example of UPDATE procedure for Count-Min (CM) sketch and Count-Min sketch with conservative update (CM-CU).

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

Figure 2.3 shows an example of CM sketch UPDATE. An item[2] (wp_i) arrives and gets mapped to three positions, corresponding to the three hash functions. Their counts before update were (5,8,6) and after update they become (6,9,7). Note that, since we are using a hash to map an item into an index, a collision can occur and multiple items may get mapped to the same counter in any given row. Because of this, the values stored by the d counters for a given item tend to differ.

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.

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

Figure 2.4 shows an example of CM sketch QUERY.

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 $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$.

[2]Example assumes count (c) associated with the item is 1.

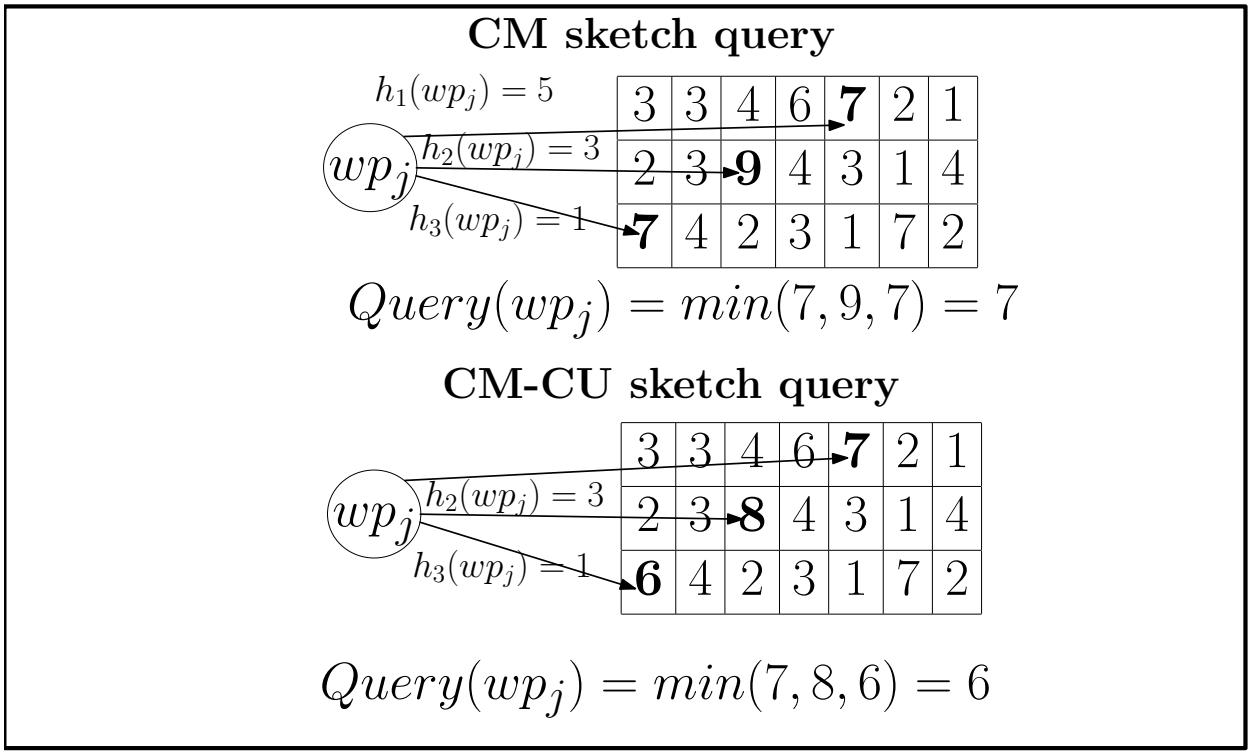


Figure 2.4: Example of QUERY procedure for Count-Min (CM) sketch and Count-Min sketch with conservative update (CM-CU). Example shows that CM-CU sketch has *less* over-estimation error than CM sketch.

2.3.1.2 Spectral Bloom Filters (SBF)

Cohen and Matias [23] proposed SBF, an extension to Bloom Filters [9] 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.

2.3.1.3 Count-mean-min (CMM)

The motivation behind the CMM [32] 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 $(N - \text{sketch}[k, h_k(\text{x})])/(w - 1)$. 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 I 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).

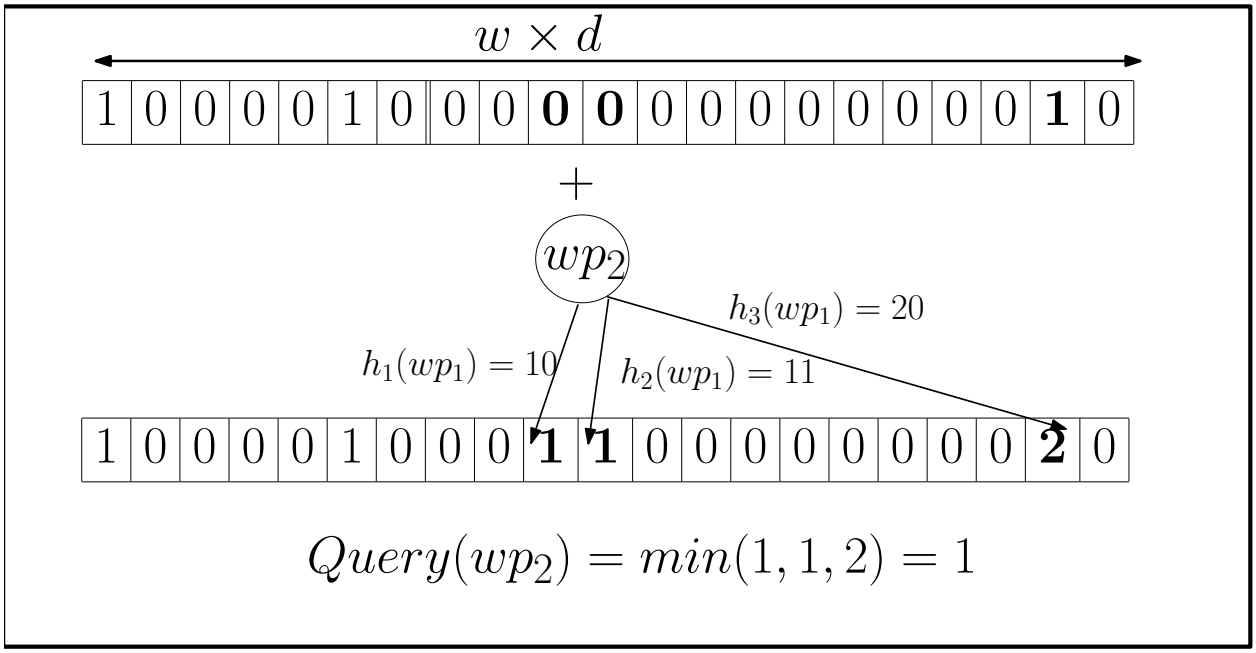


Figure 2.5: Example of UPDATE and QUERY procedure for Spectral Bloom Filters (SBF). This example shows how Spectral Bloom Filters is different from Count-Min sketch (see Figure 2.3 and 2.4).

2.3.1.4 Count sketch (COUNT)

COUNT (aka Fast-AGMS) [18] 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:

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

Figure 2.6 shows an example of the update procedure.

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

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

Figure 2.7 shows an example of the query procedure.

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 $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$.

2.3.2 Conservative Update sketch algorithms

In this section, I propose novel variants of existing sketches (see Section 2.3) by combining them with the conservative update process [44]. The idea of conservative update (also known as Minimal Increase [23]) 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

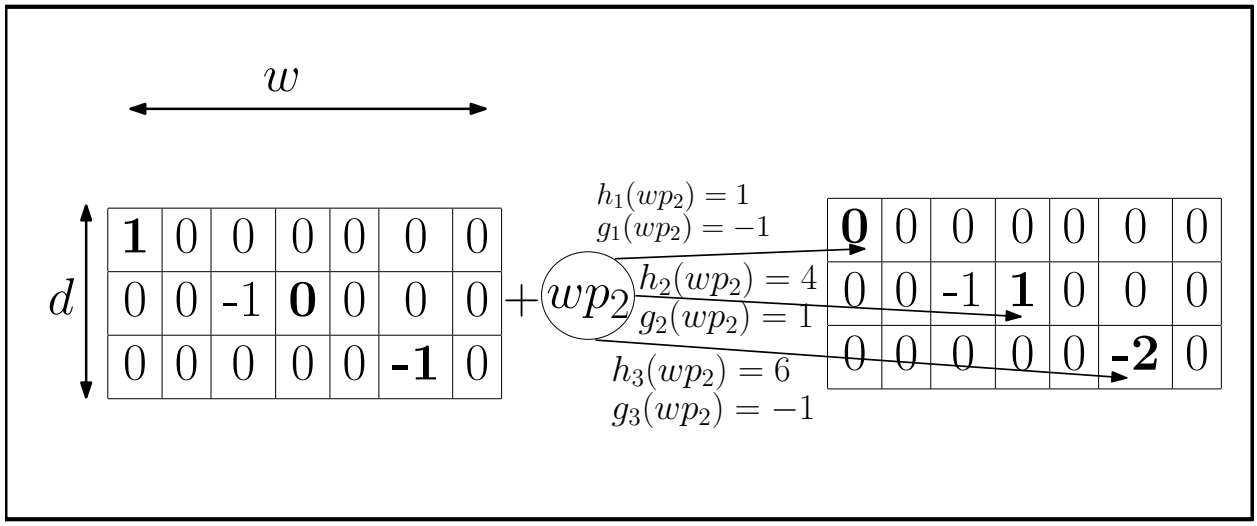


Figure 2.6: Example of UPDATE procedure for Count (COUNT) sketch.

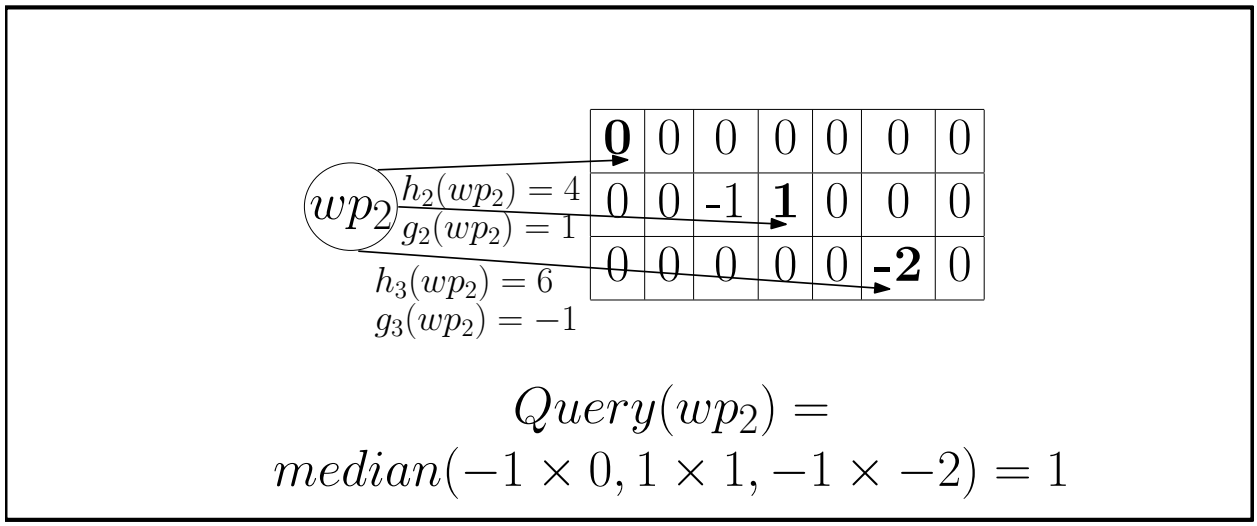


Figure 2.7: Example of QUERY procedure for Count (COUNT) sketch.

Spectral Bloom Filters (SBF) to further improve the estimate of a point query. I [53] 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 2.3).

2.3.2.1 Count-Min sketch with conservative update (CM-CU)

The QUERY procedure for CM-CU [25, 53] is identical to Count-Min. However, to UPDATE an item "x" with frequency c, I first compute the frequency $\hat{c}(\mathrm{x})$ of this item from the existing data structure ($\forall 1 \leq k \leq d$, $\hat{c}(\mathrm{x}) = \min_k \ \mathrm{sketch}[k, h_k(\mathrm{x})]$) and the counts are updated according to:

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

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

Figure 2.3 shows an example of CM-CU sketch update. When an item (wp_i) arrives, it gets mapped into three positions in the sketch data structure. Their counts before update were (5,8,6) and the frequency of the item before update is 5 (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 $\hat{c}(\mathrm{x}) + 1 = 6$. As a result, the values in these counters after the update become (6,8,6). However, if the value in any of the counters is already greater than 6 e.g. 8, 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 second counter we leave the value 8 unchanged.

Figure 2.4 shows an example of CM-CU sketch QUERY. In this example, after updating the count of an item (wp_i) in Figure 2.3, CM-CU sketch is queried for item (wp_j). The count returned by CM-CU for item (wp_j) is less than than the count returned by CM sketch. This shows that CM-CU sketch has *less* over-estimation error than CM sketch.

2.3.2.2 Spectral Bloom Filters with conservative update (SBF-CU)

The QUERY procedure for SBF-CU [23] is identical to SBF. SBF-CU UPDATE procedure is similar to CM-CU, with the difference that all d hash functions having the common range $d \times w$.

2.3.2.3 Count-mean-min with conservative update (CMM-CU)

I 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 ($\sum_i \mathrm{sketch}[k, i]$, $\forall 1 \leq k \leq d$) is not equal to input size N. Hence, the estimated noise to be subtracted here is $(\sum_i \mathrm{sketch}[k, i] - \mathrm{sketch}[k, h_k(\mathrm{x})]) \,/\, (w - 1)$. CMM-CU deducts the value of estimated noise from each of the d counters, and returns the median

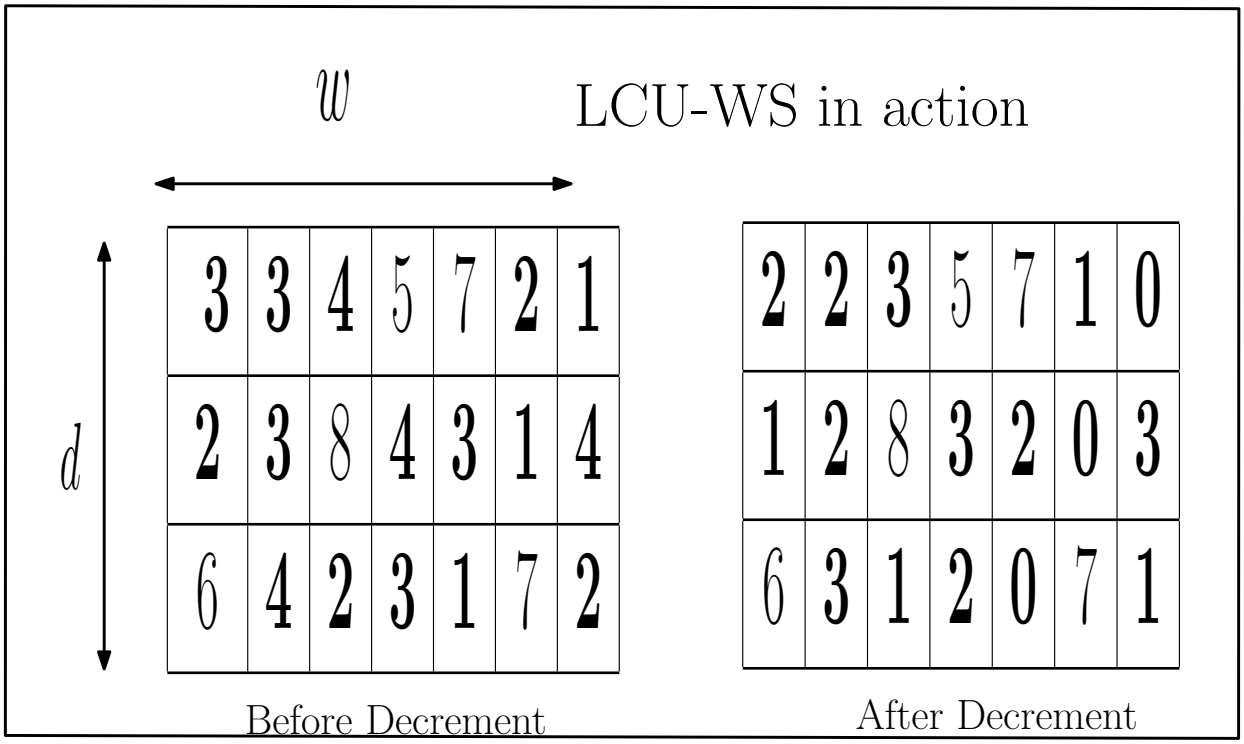


Figure 2.8: LCU-WS in Action at window boundary 4. Counters ≤ 4 are decremented at the window boundary.

of the d residues as the point query.

2.3.2.4 Count sketch with conservative update (COUNT-CU)

I 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.

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

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.

2.3.2.5 Lossy Counting with conservative update I (LCU-WS)

LCU-WS was proposed by me [54] 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 [93]. In this approach, the input sequence is conceptually divided into *epochs*, each containing $1/\gamma$ items. The size of each epoch is equal to size of the sketch i.e. $d \times w$. Note that there are γN epochs; let t denote the index of current epoch.
At epoch boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($\mathrm{sketch}[i,j] > 0$ and $\mathrm{sketch}[i,j] \leq t$), then $\mathrm{sketch}[i,j] \leftarrow \mathrm{sketch}[i,j] - 1$. 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 epoch index is used to

draw this distinction. Here, all reported frequencies $\hat{f}$ have both under and over estimation error: $f - \gamma N \leq \hat{f} \leq f + \epsilon N$.

An example of LCU-WS is shown in Figure 2.8.

2.3.2.6 Lossy Counting with conservative update II (LCU-SWS)

This [54] 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 epoch index, t.
At epoch boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if (sketch$[i,j] > 0$ and sketch$[i,j] \leq \lceil\sqrt{t}\rceil$), then sketch$[i,j] \leftarrow$ sketch$[i,j] - 1$. LCU-SWS has similar analytical bounds to LCU-WS.

2.3.2.7 Lossy Counting with conservative update III (LCU-ALL)

This approach [54] is most similar to the variant of lossy counting [92].
At epoch boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if $(sketch[i,j] > 0)$, then $sketch[i,j] \leftarrow sketch[i,j]-1$. The only difference is LCU-ALL sketch does not have items stored explicitly. Hence, the sketch decrements the whole sketch itself rather than explicitly decrementing the counts of all items by 1. Intuitively, over here sketch is only storing the frequent items into the sketch. LCU-ALL has similar analytical bounds to LCU-WS.

2.3.2.8 Lossy Counting with conservative update IV (LCU-1)

This approach [54] is the most conservative (in decrementing the entries in the sketch) variant of Lossy Counting with conservative update.
At epoch boundaries, $\forall\ 1 \leq i \leq d$, $1 \leq j \leq w$, if ($sketch[i,j] > 0$ and $sketch[i,j] <= 1$), then $sketch[i,j] \leftarrow sketch[i,j] - 1$. Intuitively, in this approach, the sketch reduces some error over low-frequent counts by turning the count of 1's to zero at the epoch boundary. LCU-1 has similar analytical bounds to LCU-WS.

2.4 Contributions

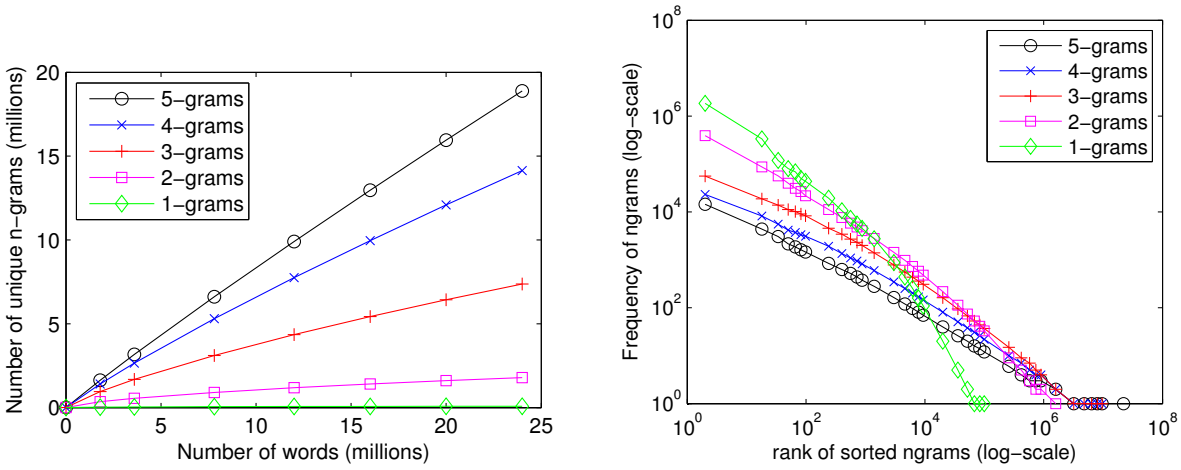
The work [54] which proposed variants of Lossy Counting with conservative update was published at Association for the Advancement of Artificial Intelligence (AAAI) 2011 conference. The work that carried out the empirical comparison study for sketches [52] was published at Empirical Methods in Natural Language Processing (EMNLP) 2012 conference. In the empirical comparison paper, I have proposed Count sketch [18] with conservative update (COUNT-CU) and Count-mean-min sketch with conservative update (CMM-CU).

Chapter 3

Streaming Large Data Language Modeling

In many NLP problems, we are faced with the challenge of dealing with large amounts of data. Many problems boil down to computing relative frequencies of certain items on this data. Items can be words, patterns, associations, n-grams, and others. Language modeling [20], noun-clustering [121], constructing syntactic rules for statistical machine translation (SMT) [49], and finding analogies [150] are examples of some of the problems where we need to compute relative frequencies.

In this chapter [56], I use language modeling as a canonical example of a large-scale task that requires relative frequency estimation.



(a) Token Type Curve

(b) Sorted n-gram Frequency Curve

Figure 3.1: Token Type and Sorted n-gram Frequency Curve using Europarl (Section 3.2.1) data.

Computing relative frequencies seems like an easy problem. However, as corpus sizes grow, it becomes a highly computational expensive task. Another problem with building higher order LMs on huge datasets is that as the order of n increases, the number of unique n-grams also increases. Figure 3.1(a) shows that the fraction of unique n-grams increases with n. For example, a 25 million word corpus generates 20 million unique 5-grams. In another example from SMT literature, Brants et al. [10] used 1500 machines for a day to compute the relative frequencies of n-grams (summed over all orders from 1 to 5) from 1.8TB of web data. Their resulting model contained 300 million unique n-grams. Hence, it is not realistic using conventional computing resources to use all the 300 million n-grams for applications like speech recognition, spelling correction, information extraction,

Cutoff	Size	BLEU	NIST	METEOR
Exact	367.6m	**28.73**	**7.691**	**56.32**
2	229.8m	28.23	7.613	56.03
3	143.6m	28.17	7.571	**56.53**
5	59.4m	28.33	7.636	56.03
10	18.3m	27.91	7.546	55.64
100	1.1m	28.03	7.607	55.91
200	0.5m	27.62	7.550	55.67

Table 3.1: Effect of count-based pruning on SMT performance using EAN corpus. Results are according to BLEU, NIST and METEOR metrics. Bold #s are not statistically significant worse than exact model.

γ	Size	BLEU	NIST	METEOR
Exact	367.6m	**28.73**	**7.691**	**56.32**
1e-10	218.4m	**28.64**	**7.669**	**56.33**
5e-10	171.0m	**28.48**	**7.666**	**56.38**
1e-9	148.0m	**28.56**	7.646	**56.51**
5e-9	91.9m	28.27	7.623	56.16
1e-8	69.4m	28.15	7.609	56.19
5e-7	28.5m	28.08	7.595	55.91

Table 3.2: Effect of entropy-based pruning on SMT performance using EAN corpus. Results are as in Table 3.1.

and statistical machine translation (SMT).

While it is true that the number of unique n-grams increases exponentially, it is also known that n-grams follow a Zipfian distribution (illustrated in Figure 3.1(b)). It can be clearly seen from Figure 3.1(b) that low frequency count n-grams contribute most towards the size of a language model. Moreover, state of the art smoothing technique [76] discount low frequency n-grams to zero. This suggests that throwing away infrequently occurring n-grams might not significantly degrade LM performance.

Hence to reduce the size of language models, researchers have used count-based pruning which discards all n-grams whose count is less than a pre-defined threshold. Although count-based pruning is quite simple, yet it is effective for machine translation. As I do not have a copy of the web, I will use a portion of Gigaword i.e. EAN (see Section 3.2.1) to show the effect of count-based pruning on performance of SMT (see Section 3.3.1). Table 3.1 shows that using a cutoff of 100 produces a model of size 1.1 million n-grams with a Bleu of 28.03. If I compare this with an exact model of size 367.6 million n-grams, I see an increase of 0.8 points in Bleu (95% statistical significance level is ≈ 0.53 Bleu). However, I need 300 times bigger model to get such an increase. Unfortunately, it is not possible to integrate such a big model inside a decoder using normal computation resources.

A better way of reducing the size of n-grams is to use entropy pruning [138]. Table 3.2 shows the results with entropy pruning with different settings of γ. I see that for three settings of γ equal to $1e$-10, $5e$-10 and $1e$-9, I get Bleu scores comparable to the exact model. However, the size of all these models is not at all small. The size of smallest model is 25% of the exact model. Even with this size it is still not feasible to integrate such a big model inside a decoder. If I take a model of size comparable to count cutoff of 100, i.e., with $\gamma =$ 5e-7, I see both count-based pruning as well as entropy pruning performs the same.

There also have been prior work on maintaining approximate counts for higher-order language models (LMs) ([143, 144, 142]) operates under the model that the goal is to store a compressed representation of a disk-resident table of counts and use this compressed representation to answer count queries approximately.

There are two difficulties with scaling all the above approaches as the order of the LM increases. Firstly, the computation time to build the database of counts increases rapidly. Secondly, the initial disk storage required to maintain these counts, prior to building the compressed representation is enormous.

The method I propose solves both of these problems. I do this by making use of the *streaming algorithm* paradigm [105]. Working under the assumption that multiple-GB models are infeasible, my goal is to instead of estimating a large model and then compressing it, I directly estimate a small model (see Figure 3.2). I use a deterministic streaming algorithm named Lossy Counting (Section 2.2) [93] that computes approximate frequency counts of frequently occurring n-grams. This scheme is considerably more accurate in getting the actual counts as compared to other schemes [31, 73] that find the set of frequent items without caring about the accuracy of counts.

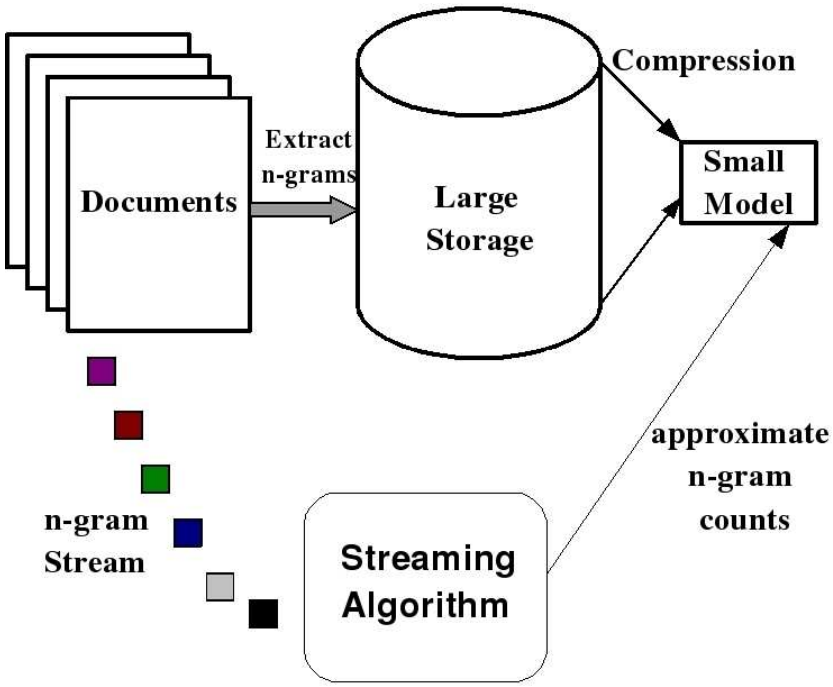


Figure 3.2: Streaming algorithm directly estimates a small model instead of first estimate a large model and then compress it.

I use these counts directly as features in an SMT system, and propose a direct way to integrate these features into an SMT decoder. Experiments show that directly storing approximate counts of frequent 5-grams compared to using count or entropy-based pruning counts gives equivalent SMT performance, while dramatically reducing the memory usage and getting rid of pre-computing a large model. This chapter was published at North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT) 2009 conference [56].

3.1 Background

3.1.1 n-gram Language Models

Language modeling is based on assigning probabilities to sentences. It can either compute the probability of an entire sentence or predict the probability of the next word in a sequence. Let w_1^m denote a sequence of words $(w_1, \ldots, w_m)$. The probability of estimating word w_m depends on previous n-1 words where n denotes the size of n-gram. This assumption that probability of predicting a current word depends on the previous words is called a Markov assumption, typically estimated by relative frequency:

$$P(w_m \mid w_{m-n+1}^{m-1}) = \frac{C(w_{m-n+1}^{m-1} w_m)}{C(w_{m-n+1}^{m-1})} \tag{3.1.1}$$

Equation 3.1.1 estimates the n-gram probability by taking the ratio of observed frequency of a particular sequence and the observed frequency of the prefix. This is precisely the relative frequency estimate I seek.

3.1.2 Large-scale Language modeling

Using higher order LMs to improve the accuracy of SMT is not new. Brants et al. [10] and Emami et al. [41] built 5-gram LMs over web using distributed cluster of machines and queried them via network requests. Since the use of cluster of machines is not always practical, Talbot and Osborne [144, 143] showed a randomized data structure called Bloom filter, that can be used to construct space efficient language models for SMT. Talbot and Brants [142] presented randomized language model based on perfect hashing combined with entropy pruning to achieve further memory reductions. A problem mentioned in Talbot and Brants [142] is that the algorithm that computes the compressed representation might need to retain the entire database in memory; in their paper, they design strategies to work around this problem. Federico and Bertoldi [45] also used single machine and fewer bits to store the LM probability by using efficient prefix trees.

Uszkoreit and Brants [154] used partially class-based LMs together with word-based LMs to improve SMT performance despite the large size of the word-based models used. Schwenk and Koehn [129] and Zhang et al. [166] used higher language models at time of re-ranking rather than integrating directly into the decoder to avoid the overhead of keeping LMs in the main memory since disk lookups are simply too slow. Now using higher order LMs at time of re-ranking looks like a good option. However, the target n-best hypothesis list is not diverse enough. Hence if possible it is always better to integrate LMs directly into the decoder.

3.2 Intrinsic Evaluation

I conduct a set of experiments with approximate n-gram counts (stream counts) produced by the original Lossy Counting algorithm (Section 2.2.1). I define various metrics on which I evaluate the quality of stream counts compared with exact n-gram counts (true counts). To evaluate the quality of stream counts on these metrics, I carry out three experiments.

3.2.1 Experimental Setup

The freely available English side of Europarl (EP) and Gigaword corpus [62] is used for computing n-gram counts. I only use EP along with two sections of the Gigaword corpus: Agence

Corpus	Gzip-MB	M-wrds	Perplexity
EP	63	38	1122.69
afe	417	171	1829.57
apw	1213	540	1872.96
nyt	2104	914	1785.84
xie	320	132	1885.33

Table 3.3: Corpus Statistics and perplexity of LMs made with each of these corpora on development set

France Press English Service (afe) and The New York Times Newswire Service (nyt). The unigram language models built using these corpora yield better perplexity scores on the development set (see Section 3.3.1) compared to The Xinhua News Agency English Service (xie) and Associated Press Worldstream English Service (apw) as shown in Table 3.3. The LMs are build using the SRILM language modelling toolkit [137] with modified Kneser-Ney discounting and interpolation. The evaluation of stream counts is done on EP +afe +nyt (EAN) corpus, consisting of 1.1 billion words.

3.2.2 Description of the metrics

To evaluate the quality of counts produced by the Lossy Counting algorithm four different metrics are used. The accuracy metric measures the quality of top N stream counts by taking the fraction of top N stream counts that are contained in the top N true counts.

$$\text{Accuracy} = \frac{\text{Stream Counts} \cap \text{True Counts}}{\text{True Counts}}$$

Spearman's rank correlation coefficient or Spearman's rho(ρ) computes the difference between the ranks of each observation (i.e. n-gram) on two variables (that are top N stream and true counts). This measure captures how different the stream count ordering is from the true count ordering.

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

d_i is the difference between the ranks of corresponding elements X_i and Y_i; N is the number of elements found in both sets; X_i and Y_i denote the stream and true counts.

Mean square error (MSE) quantifies the amount by which a predicted value differs from the true value. In current setting, it estimates how different the stream counts are from the true counts.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\text{true}_i - \text{predicted}_i)^2$$

true and predicted denotes values of true and stream counts; N denotes the number of stream counts contained in true counts.

3.2.3 Varying γ experiments

In the first experiment, I use accuracy, ρ and MSE metrics for evaluation. Here, I compute 5-gram stream counts with different settings of γ on the EAN corpus. γ controls the number of stream counts produced by the algorithm. The results in Table 3.4 support the theory that

γ	**5-gram produced**	**Accuracy**	ρ	**MSE**
50e-8	245k	0.294	-3.6097	0.4954
20e-8	726k	0.326	-2.6517	0.1155
10e-8	1655k	0.352	-1.9960	0.0368
5e-8	4018k	0.359	-1.7835	0.0114

Table 3.4: Evaluating quality of 5-gram stream counts for different settings of γ on EAN corpus

γ	**7-gram produced**	**Accuracy**	ρ	**MSE**
50e-8	44k	0.509	0.3230	0.0341
20e-8	128k	0.596	0.5459	0.0063
10e-8	246k	0.689	0.7413	0.0018
5e-8	567k	0.810	0.8599	0.0004

Table 3.5: Evaluating quality of 7-gram stream counts for different settings of γ on EP corpus

decreasing the value of γ improves the quality of stream counts. Also, as expected, the algorithm produces more stream counts with smaller values of γ. The evaluation of stream counts obtained with $\gamma = 50e$-8 and $20e$-8 reveal that the stream counts learned with this large value are more susceptible to errors.

If I look closely at the counts for $\gamma = 50e$-8, I see that I get at least 30% of the stream counts from $245k$ true counts. This number is not significantly worse than the 36% of stream counts obtained from $4,018k$ true counts for the smallest value of $\gamma =$ 5e-8. However, if I look at the other two metrics, the ranking correlation ρ of stream counts compared with true counts on $\gamma = 50e$-8 and $20e$-8 is low compared to other γ values. For the MSE, the error with stream counts on these γ values is again high compared to other values. As I decrease the value of γ I continually get better results: decreasing γ pushes the stream counts towards the true counts. However, using a smaller γ increases the memory usage. Looking at the evaluation, it is therefore advisable to use 5-gram stream counts produced with at most $\gamma \leq 10e$-7 for the EAN corpus.

Since it is not possible to compute true 7-grams counts on EAN with available computing resources, I carry out a similar experiment for 7-grams on EP to verify the results for higher order n-grams.[1] The results in Table 3.5 tell a story similar to the results for 7-grams. The size of EP corpus is much smaller than EAN and so I see even better results on each of the metrics with decreasing the value of γ. The overall trend remains the same; here too, setting $\gamma \leq 10e$-8 is the most effective strategy. The fact that these results are consistent across two datasets of different sizes and different n-gram sizes suggests that they will carry over to other tasks.

3.2.4 Varying top K experiments

In the second experiment, I evaluate the quality of the top K (sorted by frequency) 5-gram stream counts. Here again, I use accuracy, ρ and MSE for evaluation. I fix the value of γ to $5e$-8 and compute 5-gram stream counts on the EAN corpus. I vary the value of K between $100k$ and $4,018k$ (that is all the n-gram counts produced by the stream algorithm). The experimental results in Table 3.6 support the theory that stream count algorithm computes the exact count of most of the high frequency n-grams. Looking closer, I see that if I evaluate the algorithm on just the

[1]Similar evaluation scores are observed for 9-gram stream counts with different values of γ on EP corpus.

Top K	**Accuracy**	ρ	**MSE**
100k	0.994	0.9994	0.01266
500k	0.934	0.9795	0.0105
1000k	0.723	0.8847	0.0143
2000k	0.504	0.2868	0.0137
4018k	0.359	-1.7835	0.0114

Table 3.6: Evaluating top K sorted 5-gram stream counts for γ=5e-8 on EAN corpus

Top K	**Accuracy**	ρ	**MSE**
10k	0.996	0.9997	0.0015
20k	0.989	0.9986	0.0016
50k	0.950	0.9876	0.0016
100k	0.876	0.9493	0.0017
246k	0.689	0.7413	0.0018

Table 3.7: Evaluating top K sorted 7-gram stream counts for γ=10e-8 on EP corpus

top $100k$ 5-grams (roughly 5% of all 5-grams produced), I see almost perfect results. Further, if I take the top $1,000k$ 5-grams (approximately 25% of all 5-grams) I again see excellent performance on all metrics. The accuracy of the results decrease slightly, but the ρ and MSE metrics are not decreased that much in comparison. Performance starts to degrade as I get to $2,000k$ (over 50% of all 5-grams), a result that is not too surprising. However, even here I note that the MSE is low, suggesting that the frequencies of stream counts (found in top K true counts) are very close to the true counts. Thus, I conclude that the quality of the 5-gram stream counts produced for this value of γ is quite high (in relation to the true counts).

As before, I corroborate my results with higher order n-grams. I evaluate the quality of top K 7-gram stream counts on EP.[2] Since EP is a smaller corpus, I evaluate the stream counts produced by setting γ to 10e-8. Here I vary the value of K between $10k$ and $246k$ (the total number produced by the stream algorithm). Results are shown in Table 3.7. As I saw earlier with 5-grams, the top $10k$ (i.e. approximately 5% of all 7-grams) are of very high quality. Results, and this remains true even when I increase K to $100k$. There is a drop in the accuracy and a slight drop in ρ, while the MSE remains the same. Taking all counts again shows a significant decrease in both accuracy and ρ scores, but this does not affect MSE scores significantly. Hence, the 7-gram stream counts i.e. $246k$ counts produced by $\gamma = 10e$-8 are quite accurate when compared to the top $246k$ true counts.

3.2.5 Analysis of tradeoff between coverage and space

In the third experiment, I investigate whether a large LM can help MT performance. I evaluate the coverage of stream counts built on the EAN corpus on the test data for SMT experiments (see Section 3.3.1) with different values of γs. I compute the recall of each model against 3071 sentences of test data where recall is the fraction of number of n-grams of a dataset found in stream counts.

$$\text{Recall} = \frac{\text{Number of } n\text{-grams found in stream counts}}{\text{Number of } n\text{-grams in dataset}}$$

[2]Similar evaluation scores are observed for different top K sorted 9-gram stream counts with γ=10e-8 on EP corpus.

N-gram	unigram		bigram		trigram		5-gram		7-gram	
γ	**Gzip MB**	**Recall**	**Gzip MB**	**Recall**	**Gzip MB**	**Recall**	**Gzip MB**	**Recall**	**Gzip MB**	**Recall**
50e-8	.352	.785	2.3	.459	3.3	.167	1.9	.006	.864	5.6e-5
20e-8	.568	.788	4.5	.494	7.6	.207	5.3	.011	2.7	1.3e-4
10e-8	.824	.791	7.6	.518	15	.237	13	.015	9.7	4.1e-4
5e-8	1.3	.794	13	.536	30	.267	31	.020	43	5.9e-4
all	17	.816	228	.596	1200	.406	4800	.072	NA	

Table 3.8: Gzipped space required to store n-gram counts on disk and their coverage on a test set with different γs

I build unigram, bigram, trigram, 5-gram and 7-gram with four different values of γs. Table 3.8 contains the `gzip` size of the count file and the recall of various different stream count n-grams. As expected, the recall with respect to true counts is maximum for unigrams, bigrams, trigrams and 5-grams. However the amount of space required to store all true counts in comparison to stream counts is extremely high: I need $4.8GB$ of compressed space to store all the true counts for 5-grams.

For unigram models, I see that the recall scores are good for all values of γs. If I compare the approximate stream counts produced by largest γ (which is worst) to all true counts, I see that the stream counts compressed size is 50 times smaller than the true counts size, and is only three points worse in recall. Similar trends hold for bigrams, although the loss in recall is higher. As with unigrams, the loss in recall is more than made up for by the memory savings (a factor of nearly 150). For trigrams, I see a 14 point loss in recall for the smallest γ, but a memory savings of 400 times. For 5-grams, the best recall value is .020 ($1.2k$ out of $60k$ 5-gram stream counts are found in the test set). However, compared with the true counts I only loss a recall of 0.05 ($4.3k$ out of $60k$) points but memory savings of 150 times. In extrinsic evaluations, I will show that integrating 5-gram stream counts with an SMT system performs slightly worse than the true counts, while dramatically reducing the memory usage.

For 7-gram I can not compute the true n-gram counts due to limitations of available computational resources. The memory requirements with smallest value of γ are similar to those of 5-gram, but the recall values are quite small. For 7-grams, the best recall value is 5.9e-4 which means that stream counts contains only 32 out of $54k$ 7-grams contained in test set. The small recall value for 7-grams suggests that these counts may not be that useful in SMT. I further substantiate my findings in the extrinsic evaluations. There I show that integrating 7-gram stream counts with an SMT system does not affect its overall performance significantly.

3.3 Extrinsic Evaluation

We use stream counts (instead of estimating conditional probabilities) directly as a feature in SMT to evaluate its performance. We show that using stream counts is as effective as count-pruning and entropy-pruning language models. Also, we demonstrate that the amount of memory usage required for SMT to run on test set with stream counts is significantly less compared to true counts.

3.3.1 Experimental Setup

All the experiments conducted here make use of publicly available resources. Europarl (EP) corpus French-English section is used as parallel data. The publicly available Moses[3] decoder is used for training and decoding [77]. The news corpus released for ACL SMT workshop in 2007 consisting of 1057 sentences[4] is used as the development set. Minimum error rate training (MERT) is used on this set to obtain feature weights to optimize translation quality. The final SMT system performance is evaluated on a uncased test set of 3071 sentences using the BLEU [114], NIST [35] and METEOR [6] scores. The test set is the union of the 2007 news devtest and 2007 news test data from ACL SMT workshop 2007.[5]

3.3.2 Integrating stream counts feature into decoder

The Lossy Counting method only computes high-frequency n-gram counts; it does not estimate conditional probabilities. I can either turn these counts into conditional probabilities (by using SRILM) or use the counts directly. I observed no significant difference in performance between these two approaches. However, using the counts directly consumes significantly less memory at run-time and is therefore preferable. Hence, SRILM results are omitted.

The only remaining open question is: *how should we turn the counts into a feature that can be used in an SMT system?* I considered several alternatives; the most successful was a simple weighted count of n-gram matches of varying size, appropriately backed-off. Specifically, consider an n-gram model. For every sequence of words $w_i, \ldots, w_{i+N-1}$, I obtain a feature score computed recursively according to Equation (3.3.1).

$$\begin{aligned} f(w_i) &= \log\left(\frac{C(w_i)}{Z}\right) \\ f(w_i, \ldots, w_{i+k}) &= \log\left(\frac{C(w_i, \ldots, w_{i+k})}{Z}\right) \\ &+ \frac{1}{2} f(w_{i+1}, \ldots, w_{i+k}) \end{aligned} \tag{3.3.1}$$

Here, $\frac{1}{2}$ is the backoff factor and Z is the largest count in the count set (the presence of Z is simply to ensure that these values remain manageable). In order to efficiently compute these features, I store the counts in a suffix-tree. The computation proceeds by first considering w_{i+N-1} alone and then "expanding" to consider the bigram, then trigram and so on. The advantage to this order of computation is that the recursive calls can cease whenever a zero count is reached. (Extending Moses to include this required only about 100 lines of code.)

3.3.3 Results

Table 3.9 summarizes SMT results. I have 4 baseline LMs that are conventional LMs smoothed using modified Kneser-Ney smoothing. The first two trigram and 5-gram LMs are built on EP corpus and the other two are built on EAN corpus. Table 3.9 show that there is not much significant difference in SMT results of 5-gram and trigram LM on EP. As expected, the trigram built on the

[3] http://www.statmt.org/moses/

[4] http://www.statmt.org/wmt07/

[5] I found that testing on Parliamentary test data was completely insensitive to large n-gram LMs, even when these LMs are exact. This suggests that for SMT performance, more data is better *only if* it comes from the right domain.

n-gram(γ)	BLEU	NIST	METEOR	Memory GB
3 EP (exact)	25.57	7.300	54.48	2.7
5 EP (exact)	25.79	7.286	54.44	2.9
3 EAN (exact)	27.04	7.428	55.07	4.6
5 EAN (exact)	**28.73**	**7.691**	**56.32**	20.5
4(10e-8)	27.36	7.506	**56.19**	2.7
4(5e-8)	27.40	7.507	55.90	2.8
5(10e-8)	27.97	7.605	55.52	2.8
5(5e-8)	27.98	**7.611**	56.07	2.8
7(10e-8)	27.97	7.590	55.88	2.9
7(5e-8)	27.88	7.577	56.01	2.9
9(10e-8)	**28.18**	**7.611**	55.95	2.9
9(5e-8)	27.98	7.608	56.08	2.9

Table 3.9: Evaluating SMT with different LMs on EAN. Results are according to BLEU, NIST and METEOR metrics. Bold #s are not statistically significant worse than exact.

large corpus EAN gets an improvement of 1.5 Bleu. However, unlike the EP corpus, building a 5-gram LM on EAN (huge corpus) gets an improvement of 3.2 Bleu. (The 95% statistical significance boundary is about ± 0.53 Bleu on the test data, 0.077 Nist and 0.16 Meteor according to bootstrap resampling) I see similar gains in Nist and Meteor metrics as shown in Table 3.9.

I use stream counts computed with two values of γ, 5e-8 and 10e-8 on EAN corpus. I use all the stream counts produced by the algorithm. 4, 5, 7 and 9 order n-gram stream counts are computed with these settings of γ. These counts are used along with a trigram LM built on EP to improve SMT performance. The memory usage (Memory) shown in Table 3.9 is the full memory size required to run on the test data (including phrase tables).

Adding 4-gram and 5-gram stream counts as feature helps the most. The performance gain by using 5-gram stream counts is slightly worse than compared to true 5-gram LM on EAN. However, using 5-gram stream counts directly is more memory efficient. Also, the gains for stream counts are exactly the same as I saw for same sized count-based and entropy-based pruning counts in Table 3.1 and 3.2 respectively. Moreover, unlike the pruning methods, the Lossy Counting algorithm directly computes a small model, as opposed to compressing a pre-computed large model.

Adding 7-gram and 9-gram does not help significantly, a fact anticipated by the low recall of 7-gram-based counts that I saw in Section 3.2.5. The results with two different settings of γ are largely the same. This validates the intrinsic evaluation results in Section 3.2.3 that stream counts learned using $\gamma \leq$ 10e-8 are of good quality, and that the quality of the stream counts is high.

3.4 Discussion and Conclusion

I have proposed an efficient, low-memory method to construct high-order approximate n-gram LMs. The proposed method easily scales to billion-word monolingual corpora on conventional (8GB) desktop machines. I have demonstrated that approximate n-gram features could be used as a direct replacement for conventional higher order LMs in SMT with significant reductions in memory usage. In future, I will be looking into building streaming skip n-grams, and other variants (like cluster n-grams).

In NLP community, it has been shown that having more data results in better performance

[121, 10, 150]. At web scale, terabytes of data is available and that can capture broader knowledge. Streaming algorithm paradigm provides a memory and space-efficient platform to deal with terabytes of data. I hope that other NLP applications (where we need to compute relative frequencies) like noun-clustering, constructing syntactic rules for SMT, finding analogies, and others can also benefit from streaming methods. I also believe that stream counts can be applied to other problems involving higher order LMs such as speech recognition, information extraction, spelling correction and text generation.

In this chapter, I have used Lossy Counting algorithm to maintain the top-k frequent n-grams. In future, Space Savings [98] can also be used as an alternative for Lossy Counting. Note, Space Savings overestimates the counts instead of underestimating the counts as done by Lossy Counting. For further details, refer the comparison study of streaming algorithms for finding frequent items problem [27].

This is the *first* research that presents streaming language models. After this research work, there has been more work on streaming language models [82, 155]. These streaming language models [82, 155] have approximated counts of all n-grams and shown that using approximate counts of all n-grams is better than only using approximate counts of frequent n-grams. In following Chapter 4, I will focus my attention on maintaining approximate counts of *all items* instead of just focusing on storing frequent items. Chapter 5 shows that discarding approximate counts of infrequent items is *worse* than maintaining approximate counts of infrequent items.

Chapter 4

Approximate Scalable Bounded Space Sketch for Large Data NLP

Streaming approaches [105] provide a memory and time-efficient framework to deal with terabytes of data. However, these approaches are proposed to solve a singe problem. For example, my earlier work [56] and Levenberg and Osborne [82] build approximate language models and show their effectiveness in statistical machine translation (SMT). Stream-based translation models [81] has been shown effective to handle large parallel streaming data for SMT. In Van Durme and Lall [156], a Talbot Osborne Morris Bloom (TOMB) Counter [155] was used to find the top-K verbs given a query verb using the highest approximate online pointwise mutual information (PMI) values.

In this chapter [53], I explore sketch techniques, especially the Count-Min sketch [28] (Section 2.3.1) to build a *single* model to show its effectiveness on three important NLP tasks:

- Predicting the semantic orientaion of words [151]
- Distributional approaches for word similarity [3, 152]
- Unsupervised dependency parsing [22] with a little linguistics knowledge

In all these tasks, we need to compute association measures like PMI, and log likelihood ratio (LLR) between words. To compute association scores (AS), we need to count the number of times pair of words appear together within a certain window size. However, explicitly storing the counts of all word pairs is both computationally expensive and memory intensive [3, 112]. Moreover, the memory usage keeps increasing with increase in corpus size.

Figure 4.1 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.

In this Chapter, I focus my attention on maintaining approximate counts of *all items* instead of just focusing on storing frequent items (Chapter 3). I explore a variant of Count-Min (CM-CU) sketch by combining CM sketch with the conservative update process [44] (Section 2.3.2) to address the issue of *efficient storage* of such data. The CM-CU sketch stores counts of all word pairs within a *bounded space*. Storage space saving is achieved by *approximating* the frequency of word pairs in the corpus without explicitly storing the word pairs themselves. 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 constant time operations making CM-CU sketch efficient online storage data

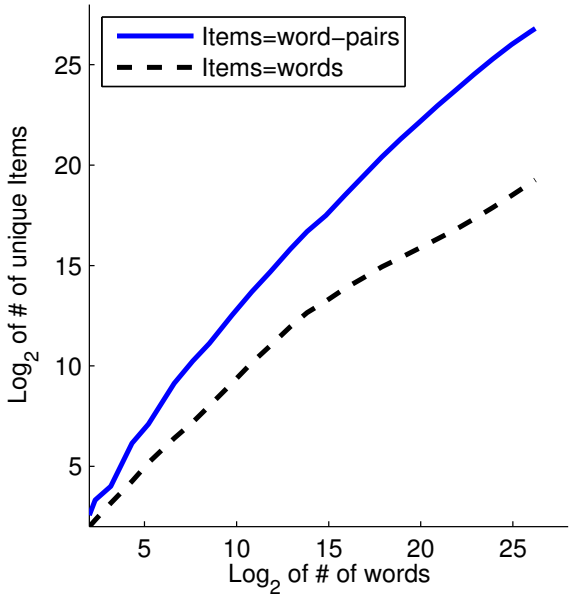


Figure 4.1: Token Type Curve

structure for large datasets. Sketches are *scalable* and can easily be implemented in distributed setting.

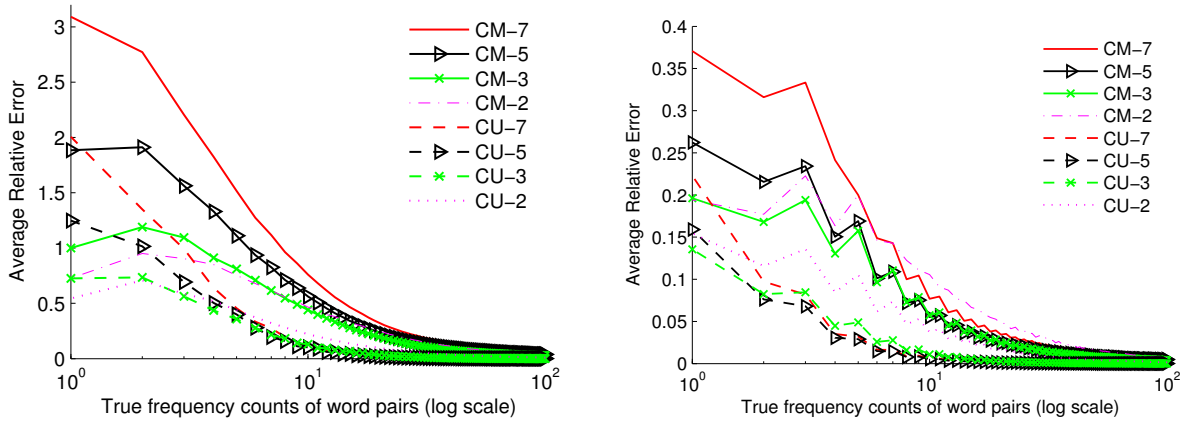
I use CM-CU sketch to store counts of word pairs (except word pairs involving stop words) within a window of size[1] 7 over different size corpora. I store exact counts of words (except stop words) in hash table (since the number of unique words is not large that is quadratically less than the number of unique word pairs). The approximate PMI and LLR scores are computed using these approximate counts and are applied to solve the three NLP tasks. Experiments demonstrate that on all of the three tasks, I get performance comparable to Exact word pair counts setting and state-of-the-art system. My method scales to 49 GB of unzipped web data using bounded space of 2 billion counters (8 GB memory). This chapter was published at Empirical Methods in Natural Language Processing (EMNLP) 2011 conference [53]. My EMNLP paper was an expanded version of my earlier workshop papers [57, 58].

4.1 Intrinsic Evaluations

To show the effectiveness of the CM sketch and CM sketch with conservative update (CM-CU) in the context of NLP, I perform intrinsic evaluations. First, the intrinsic evaluations are designed to measure the error in the approximate counts returned by CM sketch compared to their true counts. Second, I compare the word pairs association rankings obtained using PMI and LLR with sketch and exact counts.

It is memory and time intensive to perform many intrinsic evaluations on large data [121, 10, 56]. Hence, I use a subset of corpus of 2 million sentences (**Subset**) from Gigaword [62] for it. I generate words and word pairs over a window of size 7. I store exact counts of words (except stop words) in a hash table and store approximate counts of word pairs (except word pairs involving stop words) in the sketch.

[1] 7 is chosen from intuition and not tuned.



(a) 20M counters

(b) 50M counters

Figure 4.2: Compare 20 and 50 million counter models with different (width,depth) settings. The notation CM-x represents the CM sketch with a depth of 'x' and CU-x represents the CM-CU sketch with a depth of 'x'.

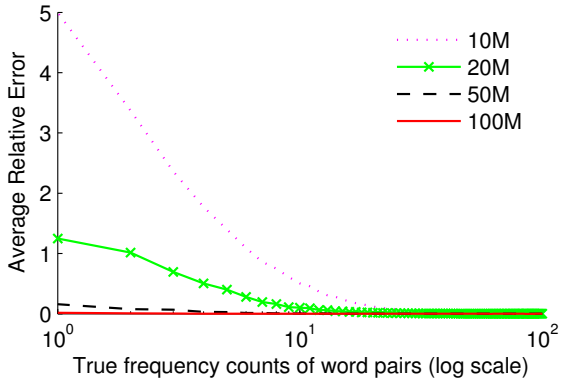


Figure 4.3: Different CM-CU sketch size models with depth 5

4.1.1 Evaluating approximate sketch counts

To evaluate the amount of over-estimation error (see Section 2.3.1) in CM and CM-CU counts compared to the true counts, I first group all word pairs with the same true frequency into a single bucket. I then compute the average relative error in each of these buckets. Since low-frequency word pairs are more prone to errors, making this distinction based on frequency lets us understand the regions in which the algorithm is over-estimating. Moreover, to focus on errors on low frequency counts, I have only plotted word pairs with count at most 100. 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 word pairs 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/CM-CU counts respectively; N denotes the number of word pairs with same counts in a bucket.

In Fig. 4.2(a), I fixed the number of counters to 20 million ($20M$) with four bytes of memory per each counter (thus it only requires 80 MB of main memory). Keeping the total number of counters fixed, I try different values of depth (2, 3, 5 and 7) of the sketch array and in each case the width is set to $\frac{20M}{d}$. The ARE curves in each case are shown in Fig. 4.2(a). I can make three main observations from Figure 4.2(a): First it shows that most of the errors occur on low frequency word pairs. For frequent word pairs, in almost all the different runs the ARE is close to zero. Secondly, it shows that ARE is significantly lower (by a factor of 1.5) for the runs which use conservative update (CU-x run) compared to the runs that use direct CM sketch (CM-x run). The encouraging observation is that, this holds true for almost all different (width,depth) settings. Thirdly, experiments 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, I tried another setting by increasing the number of counters to 50 million. The curves in 4.2(b) follow a pattern which is similar to the previous setting. Low frequency word pairs are more prone to error compared to the frequent ones and employing conservative update reduces the ARE by a factor of 1.5. In this setting, depth 5 does slightly better than depth 3 and gets lowest ARE.

I use CM-CU counts and depth of 5 for the rest of the chapter. As 3 and 5 have lowest ARE in different settings and using 5 hash functions, I get $\delta = 0.01$ ($d = \log(\frac{1}{\delta})$ refer Section 2.3.1) that is probability of failure is 1 in 100, making the algorithm more robust to false positives compared with 3 hash functions, $\delta = 0.1$ with probability of failure 1 in 10.

Fig. 4.3 studies the effect of the number of counters in the sketch (the size of the two-dimensional sketch array) on the ARE with fixed depth 5. As expected, 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 100 million counters, which is comparable to the length of the stream 88 million, I am able to achieve almost zero ARE over all the counts including the rare ones.[2] Note that the space I *save* by not storing the exact counts is almost four times the memory that I use here because on an average each word pair is twelve characters long and requires twelve bytes (thrice the size of an integer) and 4 bytes for storing the integer count. Note, I get even bigger space savings if I work with longer phrases (phrase clustering), phrase pairs (paraphrasing/translation), and varying length n-grams (Information Extraction).

4.1.2 Evaluating word pairs association ranking

In this experiment, I compare the word pairs association rankings obtained using PMI and LLR with CM-CU and exact word pair counts. I use two kinds of measures, namely recall and Spearman's correlation to measure the overlap in the rankings obtained by exact and CM-CU counts.

4.1.2.1 Pointwise Mutual Information

The pointwise mutual information (PMI) [21] between two words w_1 and w_2 is defined as:

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

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

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. It has been observed that PMI has a tendency to rank some of the relatively rare events higher than the more frequent ones. To get around this problem, a variation of PMI described in the Text-NSP toolkit[3] [5] is used that gives a higher weight to the joint occurrence of two words.

4.1.2.2 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 CM-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 (that are word pairs) on two variables (that are top N CM-CU-PMI and exact-PMI values). This measure captures how different the CM-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.

4.1.2.3 Comparing CM-CU PMI ranking

The results with respect to different sized counter (20 million ($20M$), 50 million ($50M$)) models are shown in Table 4.1. If I compare the second and third column of the table using PMI and LLR for $20M$ counters, I get exact rankings for LLR compared to PMI while comparing Top-K word pairs. The explanation for such a behavior is: since I am not throwing away any infrequent word pairs, PMI will rank pairs with low frequency counts higher [21]. Hence, I am evaluating the PMI values for rare word pairs and I need counters linear in size of stream to get almost perfect ranking. This is also evident from the fourth column for $50M$ of the Table 4.1, where CM-CU PMI ranking gets close to the optimal as the number of counters approaches stream size.

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 the extrinsic evaluations, I show that using space less than the length of the stream does not degrade the performance.

[3] `http://ngram.sourceforge.net`

# Cs	20M				50M			
AS	**PMI**		**LLR**		**PMI**		**LLR**	
TopK	R	ρ	R	ρ	R	ρ	R	ρ
50	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	.98	.94	1.0	1.0	1.0	1.0	1.0	1.0
500	.80	.98	1.0	1.0	.98	1.0	1.0	1.0
1000	.56	.99	1.0	1.0	.96	.99	1.0	1.0
5000	.35	.90	1.0	1.0	.85	.99	1.0	1.0
10000	.38	.55	1.0	1.0	.81	.95	1.0	1.0

Table 4.1: Evaluating the PMI and LLR rankings obtained using CM sketch with conservative update (CM-CU) and Exact counts

4.2 Extrinsic Evaluations

4.2.1 Data

Gigaword corpus [62] and a 50% portion of a copy of web crawled by Ravichandran et al. [121] are used to compute counts of words and word pairs. For both the corpora, I split the text into sentences, tokenize and convert into lower-case. I generate words and word pairs over a window of size 7. I use four different sized corpora: SubSet (used for intrinsic evaluations in Section 4.1), Gigaword (GW), Gigaword + 20% of web data (GWB20), and Gigaword + 50% of web data (GWB50). Corpus Statistics are shown below. I store exact counts of words in a hash table and store approximate counts of word pairs in the sketch. Hence, the stream size is the total number of word pairs (word pair tokens) in a corpus.

Corpus	**Subset**	**GW**	**GWB20**	**GWB50**
Unzipped Size (GB)	.32	9.8	22.8	49
# of sentences (Million)	2.00	56.78	191.28	462.60
Stream Size (Billion)	.088	2.67	6.05	13.20

Table 4.2: Corpus Description

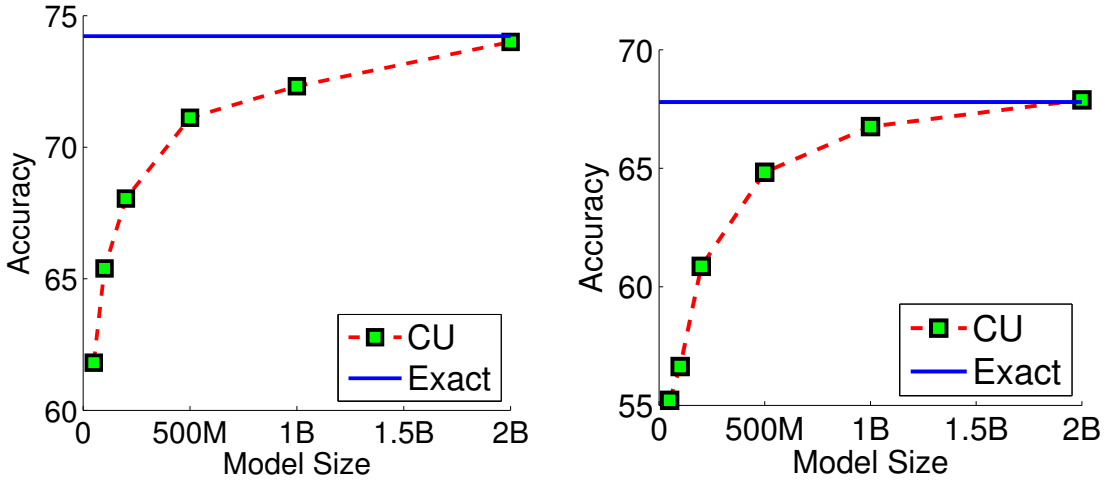
4.2.2 Semantic Orientation

Given a word, the task of finding the semantic orientation (SO) [151] of the word is to identify if the word is more likely to be used in positive or negative sense. I use a similar framework as used by the authors to infer the SO. I take the seven positive words (good, nice, excellent, positive, fortunate, correct, and superior) and the seven negative words (bad, nasty, poor, negative, unfortunate, wrong, and inferior) used in Turney's work [151]. 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. I compute the SO of a word "w" as follows:

$$\text{SO-AS(w)} = \sum_{p \in Pwords} AS(p, w) - \sum_{n \in Nwords} AS(n, w)$$

Where, Pwords and Nwords denote the seven positive and negative prototype words respectively. I use PMI and LLR to compute association scores (AS). If this score is positive, I predict the word as positive. Otherwise, I predict it as negative.

I use the General Inquirer lexicon[4] [139] as a benchmark to evaluate the semantic orientation scores similar to Turney's work [151]. Words with multiple senses have multiple entries in the lexicon, I merge these entries for the experiment. The test set consists of 1597 positive and 1980 negative words. Accuracy is used as an evaluation metric and is defined as the percentage of number of correctly identified SO words.



(a) SO PMI

(b) SO LLR

Figure 4.4: Evaluating semantic orientaion using PMI and LLR with different number of counters of CM-CU sketch built using Gigaword.

4.2.2.1 Varying sketch size

I evaluate SO of words using PMI and LLR on Gigaword (9.8GB). I compare approximate SO computed using varying sizes of CM-CU sketches: 50 million ($50M$), $100M$, $200M$, $500M$, 1 billion ($1B$) and 2 billion ($2B$) counters with Exact SO. To compute these scores, I count the number of individual words w_1 and w_2 and the pair (w_1,w_2) within a window of size 7. Note that computing the exact counts of all word pairs on these corpora is computationally expensive and memory intensive, so I consider only those pairs in which one word appears in the prototype list and the other word appears in the test set.

First, if I look at the Exact SO using PMI and LLR in Figure 4.4(a) and 4.4(b) respectively, it shows that using PMI, I get about 6 points higher accuracy than LLR on this task (The 95% statistical significance boundary for accuracy is about $\pm$ 1.5.). Second, for both PMI and LLR, having more number of counters improve performance.[5] Using $2B$ counters, I get the same accuracy as Exact.

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

[5]I use maximum of $2B$ counters (8GB main memory), as most of the current desktop machines have at most 8GB RAM.

4.2.2.2 Effect of Increasing Corpus Size

I evaluate SO of words on three different sized corpora (see Section 6.3.1): GW (9.8GB), GWB20 (22.8GB), and GWB50 (49GB). First, since for this task using PMI performs better than LLR, so I will use PMI for this experiment. Second, I will fix number of counters to $2B$ (CM-CU-2B) as it performs the best in Section 4.2.2.1. Third, I will compare the CM-CU-2B counter model with the Exact over increasing corpus size.

I can make several observations from the Figure 4.5: • It shows that increasing the amount of data improves the accuracy of identifying the SO of a word. I get an absolute increase of 5.5 points in accuracy, when I add 20% Web data to Gigaword (GW). Adding 30% more Web data (GWB50), gives a small increase of 1.3 points in accuracy which is not even statistically significant. • Second, CM-CU-2B performs as good as exact for all corpus sizes. • Third, the number of $2B$ counters (bounded space) is less than the length of stream for GWB20 ($6.05B$), and GWB50 ($13.2B$). Hence, it shows that using counters less than the stream length does not degrade the performance. • These results are also comparable to Turney's [151] state-of-the-art work where they report an accuracy of 82.84%. Note, they use a billion word corpus which is larger than GWB50.

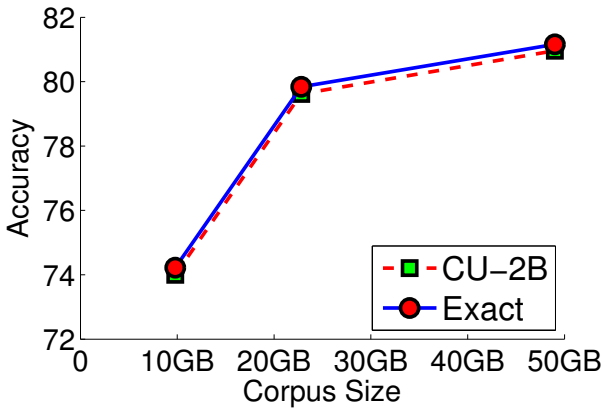


Figure 4.5: Evaluating semantic orientaion of words with Exact and CM-CU counts with increase in corpus size

4.2.3 Distributional Similarity

Distributional similarity is based on the distributional hypothesis that similar terms appear in similar contexts [47, 63]. 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.[6] I use PMI and LLR to compute association score (AS) between the term and each of the context to generate the context vector. Once, I have context vectors for each of the terms, cosine similarity measure returns distributional similarity between terms.

4.2.3.1 Efficient Distributional Similarity

I propose an efficient approach for computing distributional similarity between word pairs using CM-CU sketch. In the first step, I traverse the corpus and store counts of all words (except stop words) in hash table and all word pairs (except word pairs involving stop words) in sketch.

[6]Here, the context for a target word "x" is defined as words appear within a window of size 7.

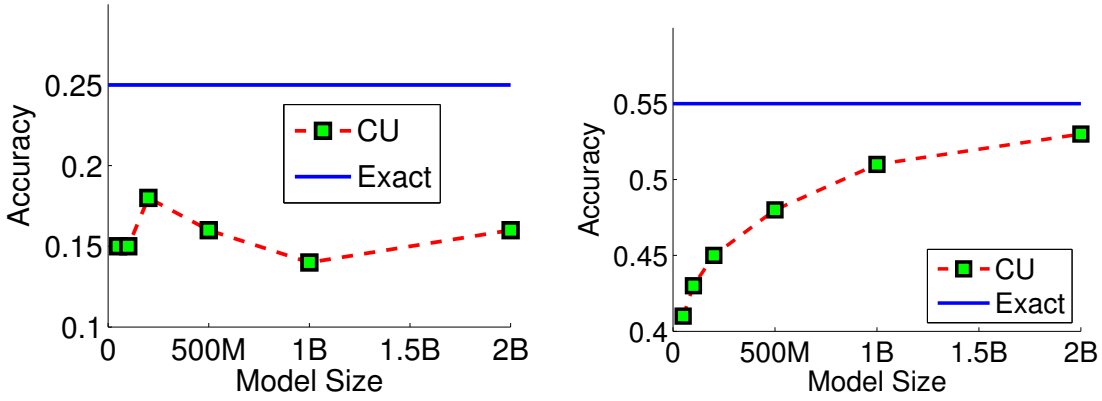
In the second step, for a target word "x", I consider all words (except infrequent contexts which appear less than or equal to 10.) as plausible context (since it is faster than traversing the whole corpus.), and query the sketch for vocabulary number of word pairs, and compute approximate AS between word-context pairs. I maintain only top K AS scores[7] contexts using priority queue for every target word "x" and save them onto the disk. In the third step, I 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 a given word pair in time $O(K)$.
- I do not store word pairs explicitly, and use fixed number of counters, hence the overall space required is bounded.
- The additive property of sketch (Section 2.3.1) enables us to parallelize most of the steps in the algorithm. Thus it can be easily extended to very large amounts of text data.

I use two test sets which consist of word pairs, and their corresponding human rankings. I generate the word pair rankings using efficient distributional similarity. I report the Spearman's rank correlation[8] coefficient (ρ) between the human and distributional similarity rankings. The two test sets are:

1. **WS-353** [46] is a set of 353 word pairs.
2. **RG-65**: [124] is set of 65 word pairs.



(a) Word Similarity PMI

(b) Word Similarity LLR

Figure 4.6: Evaluating Distributional Similarity between word pairs on WS-353 test set using PMI and LLR with different number of counters of CM-CU sketch built using Gigaword data-set.

[7]For this work, I use $K = 1000$ which is not tuned.

[8]To calculate the Spearman correlations values are transformed into ranks (if tied ranks exist, average of ranks is taken), and I calculate the Pearson correlation on them.

Test Set	WS-353			RG-65		
Model	GW	$GWB20$	$GWB50$	GW	$GWB20$	$GWB50$
Agirre			.64			.75
Exact	.55	.55	.62	.65	.72	.74
CM-CU-2B	.53	.58	.62	.66	.72	.74

Table 4.3: Evaluating word pairs ranking with Exact and CM-CU counts. Scores are evaluated using ρ metric.

4.2.3.2 Varying sketch size

I evaluate efficient distributional similarity between word pairs on WS-353 test set using PMI and LLR association scores on Gigaword (9.8GB). I compare different sizes of CM-CU sketch (similar to SO evaluation): 50 million ($50M$), $100M$, $200M$, $500M$, 1 billion ($1B$) and 2 billion ($2B$) counters with the Exact word pair counts. Here again, computing the exact counts of all word-context pairs on these corpora is time, and memory intensive, I generate context vectors for only those words which are present in the test set.

First, if I look at word pair ranking using exact PMI and LLR across Figures 4.6(a) and 4.6(b) respectively, it shows that using LLR, I get better ρ of .55 compared to ρ of .25 using PMI on this task (The 95% statistical significance boundary on ρ for WS-353 is about $\pm$ .08). The explanation for such a behavior is: PMI rank context pairs with low frequency counts higher [21] compared to frequent ones which are favored by LLR. Second, for PMI in Fig. 4.6(a), having more counters does not improve ρ. Third, for LLR in Fig. 4.6(b), having more number of counters improve performance and using $2B$ counters, I get ρ close to the Exact.

4.2.3.3 Effect of Increasing Corpus Size

I evaluate efficient distributional similarity between word pairs using three different sized corpora: GW (9.8GB), GWB20 (22.8GB), and GWB50 (49GB) on two test sets: WS-353, and RG-65. First, since for this task using LLR performs better than PMI, so I will use LLR for this experiment. Second, I will fix number of counters to $2B$ (CM-CU-2B) as it performs the best in Section 4.2.2.1. Third, I will compare the CM-CU-2B counter model with the Exact over increasing corpus size. I also compare the results against the state-of-the-art results (Agirre) for distributional similarity [3]. I report their results of context window of size 7.

I can make several observations from the Table 4.3: • It shows that increasing the amount of data is not substantially improving the accuracy of word pair rankings over both the test sets. • Here again, CM-CU-2B performs as good as exact for all corpus sizes. • CM-CU-2B and Exact performs same as the state-of-the-art system. • The number of $2B$ counters (bounded space) is less than the length of stream for GWB20 ($6.05B$), and GWB50 ($13.2B$). Hence, here again it shows that using counters less than the stream length does not degrade the performance.

4.2.4 Dependency Parsing

Recently, maximum spanning tree (MST) algorithms for dependency parsing [96] have shown great promise, primarily in supervised settings. In the MST framework, words in a sentence form nodes in a graph, and connections between nodes indicate how "related" they are. A maximum spanning tree algorithm constructs a dependency parse by linking together "most similar" words. Typically the weights on edges in the graph are parameterized as a linear function of features, with weight learned by some supervised learning algorithm. In this section, I ask the question: can word

association scores be used to derive syntactic structures in an *unsupervised* manner?

A first pass answer is: clearly not. Metrics like PMI would assign high association scores to rare word pairs (mostly content words) leading to incorrect parses. Metrics like LLR would assign high association scores to frequent words, also leading to incorrect parses. However, with a *small amount* of linguistic side information [38, 107], I see that these issues can be overcome. In particular, I see that large data + a little linguistics > fancy unsupervised learning algorithms.

4.2.4.1 Graph Definition

The approach I propose is conceptually simple. I construct a graph over nodes in the sentence with a unique "root" node. The graph is directed and fully connected, and for any two words in positions i and j, the *weight* from word i to word j is defined as:

$$w_{ij} = \alpha^{\text{asc}}\text{asc}(w_i, w_j) - \alpha^{\text{dist}}\text{dist}(i-j) + \alpha^{\text{ling}}\text{ling}(t_i, t_j)$$

Here, $\text{asc}(w_i, w_j)$ is a association score such as PMI or LLR computed using approximate counts from the sketch. Similarly, $\text{dist}(i-j)$ is a simple parameterized model of distances that favors short dependencies. I use a simple unnormalized (log) Laplacian prior of the form $\text{dist}(i-j) = -|i-j-1|$, centered around 1 (encouraging short links to the right). It is negated because I need to convert distances to similarities.

The final term, $\text{ling}(t_i, t_j)$ asks: according to some simple linguistic knowledge, how likely is if that the (gold standard) part of speech tag associated with word i points at that associated with word j? For this, I use the same linguistic information used by [107], which does *not* encode direction information. These rules are: `root` → { `aux, verb` }; `verb` → { `noun, pronoun, adverb, verb` }; `aux` → { `verb` }; `noun` → { `adj, art, noun, num` }; `prep` → { `noun` }; `adj` → { `adv` }. I simply give an additional weight of 1 to any edge that agrees with one of these linguistic rules.

4.2.4.2 Parameter Setting

The remaining issue is setting the interpolation parameters α associated with each of these scores. This is a difficult problem in purely unsupervised learning. I report results on three settings. First, the OPTIMAL setting is based on grid search for optimal parameters. This is an oracle result based on grid search over two of the three parameters (holding the third fixed at 1). In second approach, BALANCED, I normalize the three components to "compete" equally. In particular, I scale and translate all three components to have zero mean and unit variance, and set the αs to all be equal to one. Finally, third approach, SEMISUP, is based on using a small amount of labeled data to set the parameters. In particular, I use 10 labeled sentences to select parameters based on the same grid search as the OPTIMAL setting. Since this relies heavily on which 10 sentences are used, I repeat this experiment 20 times and report averages.

4.2.4.3 Experiments

My experiments are on a dependency-converted version of section 23 of the Penn Treebank using modified Collins' head finding rules. I measure accuracies as *directed*, unlabeled dependency accuracy. I separately report results of sentences of length at most 10, at most 20 and finally of all length. Note that there is no training or cross-validation: I simply run my MST parser on test data directly.

The results of the parsing experiments are shown in Table 4.4. I compare against the following

	len $\leq$ 10	**len $\leq$ 20**	**all**
Cohen-Dirichlet	45.9	39.4	34.9
Cohen-Best	59.4	45.9	40.5
Oracle	75.1	66.6	63.0
Baseline+Ling	42.4	33.8	29.7
Baseline	33.5	30.4	28.9
CM-CU-2B LLR Optimal	62.4 ± *7.7*	51.1 ± *3.2*	41.1 ± *1.9*
CM-CU-2B PMI Optimal	63.3 ± *7.8*	52.0 ± *3.2*	41.1 ± *2.0*
CM-CU-2B LLR Balanced	49.1 ± *7.6*	43.6 ± *3.3*	37.2 ± *1.9*
CM-CU-2B PMI Balanced	**49.5** ± ***8.0***	**45.0** ± ***3.2***	**38.3** ± ***2.0***
CM-CU-2B LLR Semisup	55.7 ± *0.0*	44.1 ± *0.0*	39.4 ± *0.0*
CM-CU-2B PMI Semisup	**56.5** ± ***0.0***	**45.8** ± ***0.0***	**39.9** ± ***0.0***

Table 4.4: Comparing CM-CU-2B build on GWB50 + a little linguistics v/s fancy unsupervised learning algorithms.

alternative systems. The first, Cohen-Dirichlet and Cohen-Best, are previously reported state-of-the-art results for unsupervised Bayesian dependency parsing [22]. The first is results using a simple Dirichlet prior; the second is the best reported results for any system from that paper.

Next, I compare against an "oracle" system that uses LLR extracted from the training data for the Penn Treebank, where the LLR is based on the probability of observing an edge given two words. This is not a true oracle in the sense that I *might* be able to do better, but it is unlikely. The next two baseline system are simple right branching baseline trees. The Baseline system is a purely right-branching tree. The Baseline+Ling system is one that is right branching *except* that it can only create edges that are compatible with the linguistic rules, provided a relevant rule exists. For short sentences, this is competitive with the Dirichlet prior results.

Finally I report variants of my approach using association scores computed on the GWB50 using CM-CU sketch with 2 billion counters. I experiment with two association scores: LLR and PMI. For each measure, I report results based on the three approaches described earlier for setting the α hyperparameters. Error bars for my approaches are 95% confidence intervals based on bootstrap resampling.

The results show that, for this task, PMI seems slightly better than LLR, across the board. The Optimal performance (based on tuning two hyperparameters) is amazingly strong: clearly beating out all the baselines, and only about 15 points behind the Oracle system. Using the Balanced approach causes a degradation of only 3 points from the Optimal on sentences of all lengths. In general, the balancing approach seems to be slightly worse than the semi-supervised approach, except on very short sentences: for those, it is substantially better. Overall, though, the results for both Balanced and Semisup are competitive with state-of-the-art unsupervised learning algorithms.

4.3 Discussion and Conclusion

In this chapter, I demonstrated that a version of the Count-Min sketch (CM-CU sketch) accurately solves three large-scale NLP problems using small bounded memory footprint. The advantage of using sketch in addition to being memory and time efficient is that it contains counts for all word pairs and hence can be used to compute association scores like PMI and LLR between any word pairs. I show that using sketch counts in the experiments, on the three tasks, I get

performance comparable to Exact word pair counts setting and state-of-the-art system. My method scales to 49 GB of unzipped web data using bounded space of 2 billion counters (8 GB memory). Moreover, the linearity property of the sketch makes it scalable and usable in distributed setting.

The idea of a conservative update with the Count-Min sketch (CM-CU sketch) reduces the average relative error of its approximate counts by a factor of 1.5. Association scores and counts from sketch can be used for more NLP tasks like small-space randomized language models, word sense disambiguation, spelling correction, relation learning, paraphrasing, and machine translation.

In this Chapter, we demonstrated the effectiveness of CM and CM-CU sketch on maintaining approximate counts of *all items* 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 Chapter 5, I conduct a systematic study and compare many sketch techniques which answer point queries with focus on large-scale NLP tasks.

Chapter 5

Comparing Sketch Algorithms for Large Data NLP

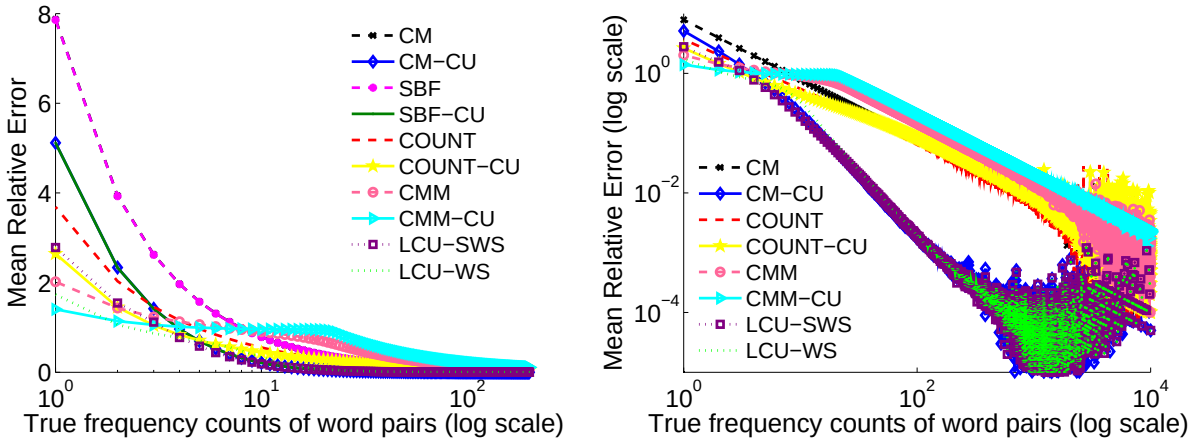
In recent years, the field of Natural Language Processing (NLP) has seen tremendous growth and interest in the use of approximation, randomization, and streaming techniques for large-scale problems [10, 150]. Much of this work relies on tracking very many statistics. For example, storing approximate counts [144, 155, 53], computing approximate association scores like pointwise mutual information [86, 156, 53], finding frequent items (like n-grams) [56], building streaming language models [142, 82], and distributional similarity [121, 157]. All these problems ultimately depend on approximate counts of items (such as n-grams, word pairs and word-context pairs). Thus I focus on solving this central problem in the context of NLP applications.

Sketch algorithms [18, 26] are a memory- and time-efficient solution to answering point queries. In Chapter 4, I demonstrated that a version of the Count-Min sketch [28] 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 chapter, I 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 [27] and join-size estimation [125], this is the *first* comparative study for NLP problems.

This work includes three contributions: (1) I propose novel variants of existing sketches by extending the idea of *conservative update* to them. I propose Count sketch [18] with conservative update (COUNT-CU) and Count-mean-min sketch with conservative update (CMM-CU). See Section 2.3.2 for details. The motivation behind proposing new sketches is inspired by the success of Count-Min sketch with conservative update (presented in Chapter 4 [53]). (2) I 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. I also evaluate their performance via pointwise mutual information and log likelihood ratio. (3) I use sketches to solve three important NLP problems. 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. This chapter was published at Empirical Methods in Natural Language Processing (EMNLP) 2012 conference [52].

5.1 Intrinsic Evaluations

I empirically compare and study the errors in approximate counts for 10 sketches. The 10 sketches are: Count-Min (CM), Spectral Bloom Filters (SBF), Count (COUNT), Count-mean-min (CMM), Count-Min sketch with conservative update (CM-CU), Spectral Bloom Filters with



(a) Focusing on low frequency counts

(b) Focusing on mid and high frequency counts

Figure 5.1: Comparing several sketches for input size of 75 million word pairs. Size of each sketch: $w = \frac{20\times10^6}{3}$ and $d = 3$. All items with same exact count are put in one bucket and I plot Mean Relative Error on the y-axis with exact counts on the x-axis.

conservative update (SBF-CU), Count sketch with conservative update (COUNT-CU), Lossy Counting with conservative update I (LCU-WS) and Lossy Counting with conservative update II (LCU-SWS). Errors can be over-estimation, under-estimation, or a combination of the two. I also study the behavior of approximate pointwise mutual information and log likelihood ratio for the sketches.

5.1.1 Experimental Setup

DATA: I took 50 million random sentences from Gigaword [62]. I split this data in 10 chunks of 5 million sentences each. Since all sketches have probabilistic bounds, I report average results over these 10 chunks. For each chunk, I 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 I compute the exact counts of all the word pairs. Second, I store the counts of all the word pairs in all the sketches. Third, I query sketches to generate *approximate* counts of all the word pairs. Recall, I do not store the word pairs explicitly in sketches but only a compact summary of the associated counts.

I fix the size of each sketch to be $w = \frac{20\times10^6}{3}$ and $d = 3$. I keep the size of sketches equal to allow fair comparison among them. Prior work [32, 53] 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, I group all word pairs with the same true frequency into a single bucket. I 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.

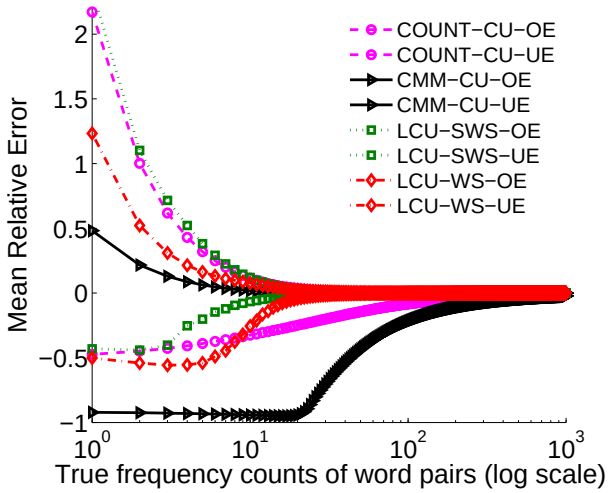


Figure 5.2: Compare several sketches on over-estimation and under-estimation errors with respect to exact counts.

5.1.2 Studying the Error in Counts

I study the errors produced by all 10 sketches. Since various sketches result in different errors on low, mid, and high frequency counts, I plot the results with a linear error scale (Fig. 5.1(a)) to highlight the performance for low frequency counts, and with a log error scale (Fig. 5.1(b)) for mid and high frequency counts.

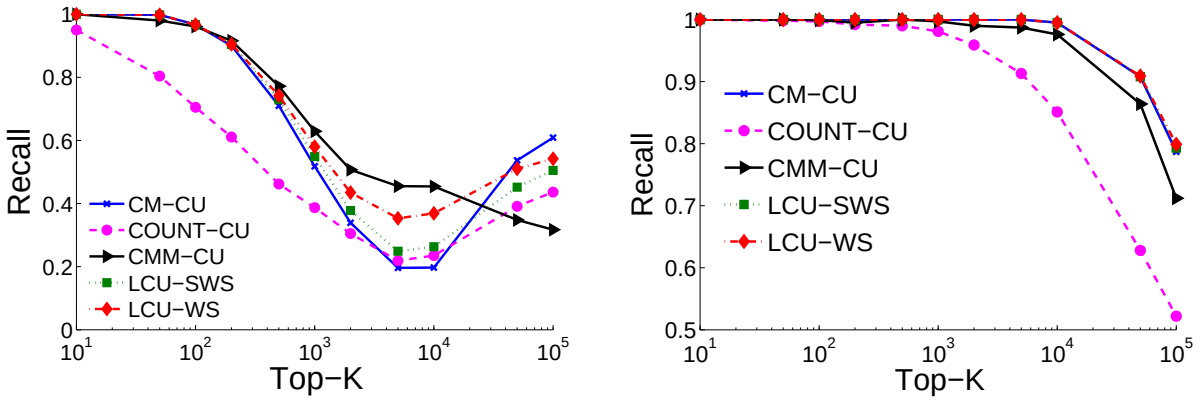
I make several observations on low frequency counts from Fig. 5.1(a). (1) Count-Min (CM) and Spectral Bloom Filters (SBF) have identical MRE for word pairs. Using conservative update with CM (CM-CU) and SBF (SBF-CU) reduces the MRE by a factor of 1.5. MRE for CM-CU and SBF-CU is also identical. (2) COUNT 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. 5.1(b), I do not plot the SBF variants as SBF and CM variants had identical MRE in Fig. 5.1(a). From Figure 5.1(b), I 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.

5.1.3 Examining OE and UE errors

In many NLP applications, we are willing to tolerate either over-estimation or under-estimation errors. Hence I 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, I divide the values



(a) PMI (b) LLR

Figure 5.3: Evaluate the approximate PMI and LLR rankings (obtained using various sketches) with the exact rankings.

into over-estimation (positive), and under-estimation (negative) error buckets. Hence, to compute the over-estimation MRE, I take the average of positive values over all the items in each bucket. For under-estimation, I take the average over the negative values. I can make several interesting observations from Figure 5.2: (1) Comparing COUNT-CU and LCU-SWS, I 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 errors 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 can 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.

5.1.4 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, I compare the word pairs association rankings obtained using PMI and LLR from several sketches and exact word pair counts. I use recall to measure the number of top-K sorted word pairs that are found in both the rankings.

In Figure 5.3(a), I 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. I can make several observations from Figure 5.3(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 [21], over-estimation of the counts of low frequency word pairs can make their approximate PMI scores worse.

In Figure 5.3(b), I 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

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 5.1: 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.

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.

5.2 Extrinsic Evaluation

5.2.1 Experimental Setup

I 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 2.3) considering these sketches make errors on different ranges of the counts: low, mid and, high frequency counts as seen in the intrinsic evaluations in Section 5.1. 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. I 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: I use Gigaword [62] and a 50% portion of a copy of news web (GWB50) crawled by [121]. 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, I split the text into sentences, tokenize and convert into lower-case.

5.2.2 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 [42, 8] that measure the strength of association between a predicate and its argument filler, e.g., that the noun "song"

is likely to be a object of 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 the 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 [17] 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.

I use a simple approach (without any typed dependency data) similar to Chambers and Jurafsky [17], where I 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 a 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. I return that noun as the correct selectional preference for a target verb, which has higher co-occurrence count with it. I 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 instances), and *Neighbor* (3881 instances). To evaluate against the exact counts, I 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. 5.4, I 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. 5.4, I 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 5.1, I 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 5.1 shows that with sketches 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. I 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.

5.2.3 Semantic Orientation

Given a word, the task of finding its semantic orientaion (SO) [151] is to determine if the word is more probable to be used in positive or negative connotation. I use Turney and Littman's [151] state-of-the-art framework to compute the SO of a word. I 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). I compute the SO of a word "w" as:

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

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

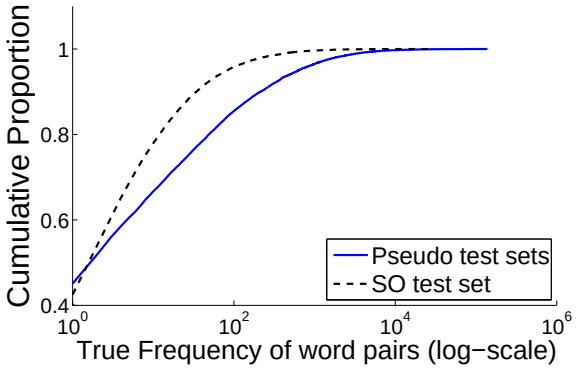


Figure 5.4: Determining the proportion of low, mid and high frequency test word pairs in Gigaword (GW).

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

where, Pos and Neg denote the seven positive and negative seeds respectively. If this score is non-negative, I predict the word as positive. Otherwise, I predict it as negative. I use the General Inquirer lexicon[2] [139] as a benchmark to evaluate the semantic orientation similar to Turney and Littman's [151] work. Words with multiple senses have multiple entries in the lexicon, I merge these entries for current experiment. The 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.

I evaluate SO of words on two different sized corpora (see Section 6.3.1): Gigaword (GW) (9.8GB), and GW with 50% news web corpus (GWB50) (49GB). I fix the size of all sketches to 2 billion ($2B$) counters with 5 hash functions. I store exact counts of all words in a hash table for both GW and GWB50. I 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 from GW and 13.20B tokens and 0.8B types from GWB50. Next, I compare the sketches against the exact counts over two different size corpora.

Table 5.2 shows that increasing the amount of data improves the accuracy of identifying the SO of a word. I get an absolute increase of 7% (with exact counts) points in accuracy (The 95% statistical significance boundary for accuracy is about $\pm$ 1.5.), when I 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 and Littman's [151] 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 5.4 has similar frequency distribution of word pairs[3] in SO test set as pseudo-words

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

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

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

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.

5.2.4 Distributional Similarity

Distributional similarity is based on the distributional hypothesis that similar terms appear in similar contexts [47, 63]. 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, I 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. I use PMI and LLR to compute association score (AS) between the term and each of the context to generate the context vector. I use the cosine similarity measure to find the distributional similarity between the context vectors for each of the terms.

I use two test sets which consist of word pairs, and their corresponding human rankings. I generate the word pair rankings using distributional similarity. I report the Spearman's rank correlation coefficient (ρ) between the human and distributional similarity rankings. I report results on two test sets: **WS-203**: A set of 203 word pairs marked according to similarity [3]. **MC-30**: A set of 30 noun pairs [100].

I evaluate distributional similarity on Gigaword (GW) (9.8GB) (see Section 6.3.1). First, I store exact counts of all words and contexts in a hash table from GW. Next, I 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, I maintain top-1000 approximate AS scores contexts using a priority queue, by passing over the corpus a second time. Finally, I use cosine similarity with these approximate top-K context vectors to compute distributional similarity.

In Table 5.3, I vary the size of all sketches (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 5.3 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 comparable against the state-of-the-art results for distributional similarity [3].

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 phenomena is demonstrated in Table 5.3, where CMM-CU and LCU-WS have worse result with small size sketches.

5.3 Conclusion

In this chapter, I systematically studied the problem of estimating point queries using different sketch algorithms. As far as I know, this represents the *first* comparative study to demonstrate the relative behavior of sketches in the context of NLP applications. I proposed two novel sketch variants: Count sketch [18] with conservative update (COUNT-CU) and Count-mean-min sketch with conservative update (CMM-CU). I 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, I 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 [144, 141, 155] in NLP. However, it may be challenging to combine this with conservative update.

Chapter 6

Fast Large-Scale Approximate Graph Construction for NLP

6.1 Introduction

In this Chapter, I focus on more general and a broad Natural Language Processing (NLP) problem which involves constructing large nearest-neighbor graphs in a memory- and time-efficient manner. Such a graph is necessary for a wide variety of NLP tasks, examples include constructing polarity lexicons based on lexical graphs from WordNet [120], constructing polarity lexicons from web data [159], unsupervised part-of-speech tagging using label propagation [30] and detecting visual text [36]. The later three approaches construct nearest-neighbor graphs between word pairs by computing nearest neighbors between word pairs from large corpora. These nearest neighbors form the edges of the graph, with weights given by the distributional similarity [152] between terms. Unfortunately, computing the distributional similarity between all words in a large vocabulary is computationally and memory intensive when working with large amounts of data [112]. This bottleneck is typically addressed by means of commodity clusters. For example, Pantel et al. [112] compute distributional similarity between 500 million terms over a 200 billion words in 50 hours using 100 quad-core nodes, explicitly storing a similarity matrix between 500 million terms.

In this chapter, I propose Fast Large-Scale Approximate Graph (FLAG) construction, a system that constructs a fast large-scale approximate nearest-neighbor graph from a large text corpus. To build this system, I exploit recent developments in the area of approximation, randomization and streaming for large-scale NLP problems [121, 56, 81]. More specifically I exploit work on Locality Sensitive Hashing (Lsh) [19] for computing word-pair similarities from large text collections [121, 157]. However, Ravichandran et al. [121] approach stored an enormous matrix of all unique words and their contexts in main memory, which is infeasible for very large data sets. A more efficient online framework to locality sensitive hashing [157, 158] computes distributional similarity in a streaming setting. Unfortunately, their approach can handle only additive features like raw-counts, and not non-linear association scores like pointwise mutual information (PMI), which generates better context vectors for distributional similarity [121, 112, 152].

In FLAG, I first propose a novel distributed online-PMI algorithm (Section 6.2.1). It is a streaming method that processes large data sets in one pass while distributing the data over commodity clusters and returns context vectors weighted by pointwise mutual information (PMI) for all the words. The distributed online-PMI algorithm makes use of the Count-Min (CM) sketch algorithm [28] (previously shown effective for computing distributional similarity in my earlier work [53]) to store the counts of all words, contexts and word-context pairs using only $8GB$ of main memory. The main motivation for using the CM sketch comes from its linearity property (see last paragraph of Section 2.3) which makes CM sketch to be implemented in distributed setting for

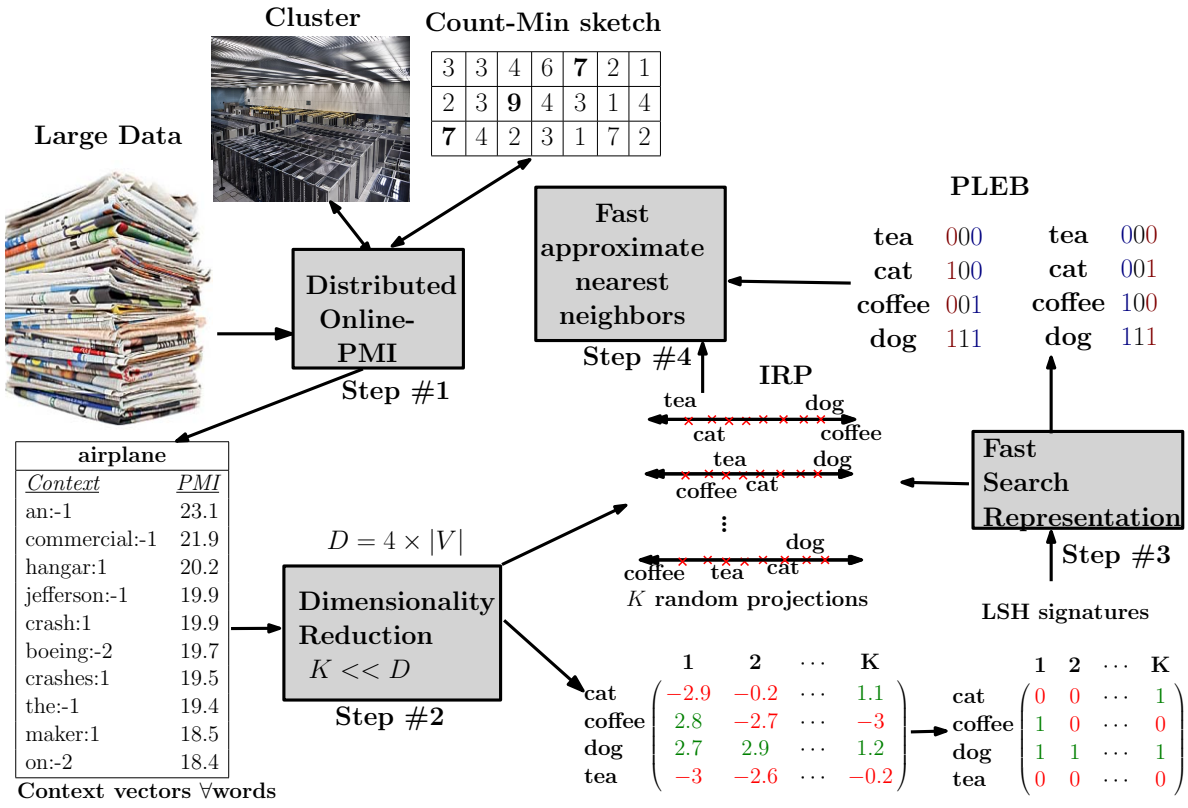


Figure 6.1: Fast Large-Scale Approximate Graph (FLAG) system

large data sets. In my implementation, FLAG scaled up to 110 GB of web data with 866 million sentences in less than 2 days using 100 quad-core nodes. Both intrinsic and extrinsic experiments demonstrate the effectiveness of distributed online-PMI.

After generating context vectors from distributed online-PMI algorithm, the goal is to use them to find fast approximate nearest neighbors for all words. To achieve this goal, I exploit recent developments in the area of existing randomized algorithms for random projections [1, 87], Locality Sensitive Hashing (LSH) [19] and improve on previous work done on PLEB (Point Location in Equal Balls) [67, 19]. I propose novel variants of PLEB to address the issue of reducing the pre-processing time for PLEB. One of the variants of PLEB (FAST-PLEB) with considerably *less* pre-processing time has effectiveness comparable to PLEB. I evaluate these variants of PLEB both quantitatively and qualitatively on large data sets. Finally, I show the applicability of large-scale graphs built from FLAG on two applications: the Google-Sets problem [50], and learning concrete and abstract words [149]. This chapter was published at Empirical Methods in Natural Language Processing (EMNLP) 2012 conference [55].

6.2 FLAG: Fast Large-Scale Approximate Graph Construction

I describe a system, FLAG, for generating a nearest neighbor graph from a large corpus. For every node (word), the system returns top l approximate nearest neighbors, which implicitly defines the graph. Figure 6.1 shows an overview of the system FLAG. The system operates in four steps. First, for every word "z", the system generates a sparse context vector

$(\langle(c_1, v_1); (c_2, v_2) \ldots; (c_d, v_d)\rangle)$ of size d where c_d denotes the context and v_d denotes the PMI (strength of association) between the context c_d and the word "z". The context can be lexical, semantic, syntactic, and/or dependency units that co-occur with the word "z". I compute this efficiently using a new distributed online Pointwise Mutual Information algorithm (Section 6.2.1). Second, I project all the words with context vector size d onto k random vectors and then binarize these random projection vectors (Section 6.2.2). Third, I propose novel variants of PLEB (Section 6.2.3) with less pre-processing time to represent data for fast query retrieval. Fourth, using the output of variants of PLEB, I generate a small set of potential nearest neighbors for every word "z" (Section 6.2.4). From this small set, I can compute the Hamming distance between every word "z" and its potential nearest neighbors to return the l nearest-neighbors for all unique words.

6.2.1 Distributed online-PMI

I propose a *new* distributed online Pointwise Mutual Information (PMI) algorithm motivated by the online-PMI algorithm [156] (page 5). This is a streaming algorithm which processes the input corpus in one pass. After one pass over the data set, it returns the context vectors for all query words. The original online-PMI algorithm was used to find the top-d verbs for a query verb using the highest approximate online-PMI values using a Talbot-Osborne-Morris-Bloom[1] (TOMB) Counter [155]. Unfortunately, this algorithm is prohibitively slow when computing contexts for *all* words, rather than just a small query set. This motivates us to propose a distributed variant that enables us to scale to large data and large vocabularies.[2]

Algorithm 1 Modified online-PMI

Require: Data set D, buffer size B
Ensure: context vectors V, mapping word z to d-best contexts in priority queue $\langle$ y,PMI(z, y)$\rangle$

```
initialize CM-CU sketch to store approximate counts of words, context and word-context pairs
for each buffer B in the data set D do
    initialize S to store ⟨z,y⟩ observed in B
    for ⟨z,y⟩ in B do
        set S (⟨z,y⟩) =1
        insert z, y and pair ⟨z,y⟩ in sketch
    end for
    for x in set S do
        recompute vectors V(x) using current contexts in priority queue and {y|S(⟨z,y⟩)=1}
    end for
end for
return context vectors V
```

I make three modifications to the original online-PMI algorithm and refer to it as the "modified online-PMI algorithm" shown in Algorithm 1. First, I use Count-Min with conservative update (CM-CU) sketch [53] instead of TOMB. I prefer CM-CU sketch because it enables distributed framework due to its linearity property (Section 2.3) and footnote #2. Second, I store the counts of words ("z"), contexts ("y") and word-context pairs all together in the CM-CU sketch (in the original online-PMI algorithm, exact counts of words and contexts were stored in a hash table; only

[1] TOMB is a variant of CM sketch which focuses on reducing the bit size of each counter (in addition to the number of counters) at the cost of incurring more error in the counts.

[2] The serialized online-PMI algorithm took a week to generate context vectors for all the words from GW (Section 6.3.1).

the pairs were stored in the TOMB data structure). Third, in the original algorithm, for each "z" a vector of top-d contexts are modified at the end of each buffer (refer Algorithm 1). However, in my algorithm, I only modify the list of those "z"'s which appeared in the recent buffer rather than modifying for all the "z"'s (Note, if "z" does not appear in the recent buffer, then its top-d contexts cannot be changed. Hence, I only modify those "z"s which appear in the recent buffer).

In distributed online-PMI algorithm, first I split the data into chunks of 10 million sentences. Second, I run the modified online-PMI algorithm on each chunk in distributed setting. This stores counts of all words ("z"), contexts ("y") and word-context pairs in the CM-CU sketch, and store top-d contexts for each word in priority queues. In third step, I merge all the sketches using linearity property to sum the counts of the words, contexts and word-context pairs. Additionally I merge the lists of top-d contexts for each word. In the last step, I use the single merged sketch and merged top-d contexts list to generate the final distributed online-PMI top-d contexts list.

It takes around one day to compute context vectors for all the words from a chunk of 10 million sentences using first step of distributed online-PMI. I generated context vectors for all the 87 chunks (110 GB data with 866 million sentences: see Table 6.1) in one day by running one process per chunk over a cluster. The first step of the algorithm involves traversing the data set and is the most time intensive step. For the second step, the merging of sketches is fast, since sketches are two dimensional array data structures (I used the sketch of size 2 billion counters with 3 hash functions). Merging the lists of top-d contexts for each word is embarrassingly parallel and fast. The last step to generate the final top-d contexts list is again embarrassingly parallel and fast and takes couple of hours to generate the top-d contexts for all the words from all the chunks. If implemented serially the "modified online-PMI algorithm" on 110 GB data with 866 million sentences would take approximately 3 months.

The downside of the distributed online-PMI is that it splits the data into small chunks and loses information about the global best contexts for a word over all the chunks. The algorithm locally computes the best contexts for each chunk, that can be bad if the algorithm misses out globally *good* contexts and that can affect the accuracy of downstream application. I will demonstrate in experiments (Section 6.3.2) by using distributed online-PMI, I do not lose any significant information about global contexts and perform comparable to offline-PMI over an intrinsic and extrinsic evaluation.

$$\begin{array}{c} \\ \mathbf{k_1} \\ \mathbf{k_2} \\ \vdots \\ \mathbf{k_K} \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & Z \end{array} \\ \begin{pmatrix} \langle z_1,\ 26\rangle & \langle z_2, 80\rangle & \cdots & \langle z_Z,\ 3\rangle \\ \langle z_1, -28\rangle & \langle z_2,\ 6\rangle & \cdots & \langle z_Z, 111\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle z_1,\ 78\rangle & \langle z_2, 69\rangle & \cdots & \langle z_Z, 92\rangle \end{pmatrix} \end{array} \overset{\textbf{Sort}}{\Longrightarrow} \begin{array}{c} \begin{array}{cccc} \textbf{Smallest} & \textbf{to} & & \textbf{Largest} \end{array} \\ \begin{pmatrix} \langle z_Z,\ 3\rangle & \langle z_1, 26\rangle & \cdots & \langle z_m, 700\rangle \\ \langle z_r, -50\rangle & \langle z_2,\ 6\rangle & \cdots & \langle z_Z, 111\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle z_1, 78\rangle & \langle z_Z, 92\rangle & \cdots & \langle z_u, 432\rangle \end{pmatrix} \end{array} \Rightarrow \begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & Z \end{array} \\ \begin{pmatrix} z_Z & z_1 & \cdots & z_m \\ z_r & z_2 & \cdots & z_Z \\ \vdots & \vdots & \ddots & \vdots \\ z_1 & z_Z & \cdots & z_u \end{pmatrix} \end{array} \Rightarrow \begin{array}{c} \begin{array}{cccc} z_1 & z_2 & \cdots & z_Z \end{array} \\ \begin{pmatrix} 2 & 60 & \cdots & 1 \\ 55 & 2 & \cdots & Z \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 90 & \cdots & 2 \end{pmatrix} \end{array}$$

(a) Matrix A (b) Matrix A (c) Matrix A (d) Matrix C

Figure 6.2: First matrix pairs the words $1 \cdots Z$ and their random projection values. Second matrix sorts each row by the random projection values from smallest to largest. Third matrix throws away the projection values leaving only the words. Fourth matrix maps the words $1 \cdots Z$ to their sorted position in the third matrix for each k. This allows constant query time for all the words.

6.2.2 Dimensionality Reduction from $\mathbb{R}^D$ to $\mathbb{R}^k$

Given context vectors for Z words, the goal is to use k random projections to project the context vectors from $\mathbb{R}^D$ to $\mathbb{R}^k$. There are total D unique contexts ($D >> k$) for all Z words. Let $(\langle(\mathrm{c}_1, v_1); (\mathrm{c}_2, v_2) \ldots; (\mathrm{c}_d, v_d)\rangle)$ be sparse context vectors of size d for Z words. For each word,

I use hashing to project the context vectors onto k directions. I use k pairwise independent hash functions that maps each of the d context (c_d) dimensions onto $\beta_{d,k} \in \{-1, +1\}$; and compute inner product between $\beta_{d,k}$ and v_d. Next, $\forall k, \sum_d \beta_{d,k}.v_d$ returns the k random projections for each word "z". I store the k random projections for all words (mapped to integers) as a matrix A of size of $k \times Z$.

The mechanism described above generates random projections by implicitly creating a random projection matrix from a set of $\{-1, +1\}$. This idea of creating implicit random projection matrix is motivated by the research on stable random projections [87, 86], Count sketch [18], feature hashing [161] and online Locality Sensitive Hashing (LSH) [157]. The idea of generating random projections from the set $\{-1, +1\}$ was originally proposed by Achlioptas [1].

Next I create a binary matrix B using matrix A by taking sign of each of the entries of the matrix A. If $A(i, j) \geq 0$, then $B(i, j) = 1$; else $B(i, j) = 0$. This binarization creates Locality Sensitive Hash (LSH) function that preserves the cosine similarity between every pair of word vectors. This idea was first proposed by Charikar [19] and used in NLP for large-scale noun clustering [121]. However, in large-scale noun clustering research, they had to store the random projection matrix of size $D \times k$; where D denotes the number of all unique contexts (which is generally large and $D >> Z$) and this research does not explicitly require storing a random projection matrix.

6.2.3 Representation for Fast-Search

I describe three approaches to represent the data (matrix A and B from Section 6.2.2) in such a manner that finding nearest neighbors is fast. These three approaches differ in amount of pre-processing time. First, I propose a naive baseline approach using random projections independently with the best pre-processing time. Second, I describe PLEB (Point Location in Equal Balls) [67, 19] with the worst pre-processing time. Third, I propose a variant of PLEB to reduce its pre-processing time.

6.2.3.1 Independent Random Projections

Here, I describe a naive baseline approach to arrange nearest neighbors next to each other by using Independent Random Projections (IRP). In this approach, I pre-process the matrix A. First for matrix A, I pair the words $z_1 \cdots z_Z$ and their random projection values as shown in Fig. 6.2(a). Second, I sort the elements of each row of matrix A by their random projection values from smallest to largest (shown in Fig. 6.2(b)). The sorting step takes $O(ZlogZ)$ time (We can assume k to be a constant). The sorting operation puts all the nearest neighbor words (for each k independent projections) next to each other. After sorting the matrix A, I throw away the projection values leaving only the words (see Fig. 6.2(c)). To search for a word in matrix A in constant time, I create another matrix C of size $(k \times Z)$ (see Fig. 6.2(d)). Matrix C maps the words $z_1 \cdots z_Z$ to their sorted position in the matrix A (see Fig. 6.2(c)) for each k.

6.2.3.2 Point Location in Equal Balls

Point Location in Equal Balls (PLEB) was first proposed by Indyk and Motwani [67] and further improved by Charikar [19]. The improved PLEB algorithm puts in operation *all* k random projections together. It randomly permutes the ordering of k binary LSH bits (stored in matrix B) for all the words p times. For each permutation it sorts all the words lexicographically based on their permuted LSH representation of size k. The sorting operation puts all the nearest neighbor words (using k projections together) next to each other for all the permutations. In practice p is generally large, Ravichandran et al. [121] used $p = 1000$ in their work.

In my implementation of PLEB, I have a matrix A of size $(p \times Z)$ similar to the first matrix in Fig. 6.2(a). The main difference to the first matrix in Fig. 6.2(a) is that bit vectors of size k are used for sorting rather than using scalar projection values. Similar to Fig. 6.2(c) after sorting, bit vectors are discarded and a matrix C of size $(p \times Z)$ is used to map the words $1 \cdots Z$ to their sorted position in the matrix A. Note, in IRP approach, the size of A and C matrix is $(k \times Z)$. In PLEB generating random permutations and sorting the bit vectors of size k involves worse pre-processing time than using IRP. However, spending more time in pre-processing leads to finding better approximate nearest neighbors.

6.2.3.3 Fast Point Location in Equal Balls

To reduce the pre-processing time for PLEB, I propose Fast Point Location in Equal Balls (FAST-PLEB). In PLEB, while generating random permutations, it uses all the k bits. In this variant, for each random permutation I randomly sample without replacement q ($q << k$) bits out of k. I use q bits to represent each permutation and sort based on these q bits. This makes pre-processing *faster* for PLEB. Section 6.3.3 shows that FAST-PLEB only needs $q = 10$ to perform comparable to PLEB with $q = 3000$ (that makes FAST-PLEB 300 times faster than PLEB). Here, again I store matrices A and C of size $(p \times Z)$.

6.2.4 Finding Approximate Nearest Neighbors

The goal here is to exploit three representations discussed in Section 6.2.3 to find approximate nearest neighbors quickly. For all the three methods (IRP, PLEB, FAST-PLEB), I can use the same fast approximate search which is simple and fast. To search a word "z", first, I can look up matrix C to locate the k positions where "z" is stored in matrix A. This can be done in constant time (Again assuming k (for IRP) and p (for PLEB and FAST-PLEB) to be a constant.). Once, I find "z" in each row, I can select b (beam parameter) neighbors ($b/2$ neighbors on left and $b/2$ neighbors on right of the query word) for all the k or p rows. This can be done in constant time (Assuming k, p and b to be constants). This search procedure produces a set of bk (IRP) or bp (PLEB and FAST-PLEB) potential nearest neighbors for a query word "z". Next, I compute Hamming distance between query word "z" and the set of potential nearest neighbors from matrix B to return l closest nearest neighbors. For computing hamming distance, all the approaches discussed in Section 6.2.3 require all k random projection bits.

6.3 Experiments

I evaluate system FLAG for fast large-scale approximate graph construction. First, I show that using distributed online-PMI algorithm is as effective as offline-PMI. Second, I compare the approximate nearest neighbors lists generated by FLAG against the exact nearest neighbor lists. Finally, I show the quality of the approximate similarity lists generated by FLAG from the web corpus.

6.3.1 Experimental Setup

Data sets: I use two data sets: Gigaword [62] and a copy of news web [121]. For both the corpora, I split the text into sentences, tokenize and convert into lower-case. To evaluate the approximate graph construction, I evaluate on three data sets: Gigaword (GW), Gigaword + 50% of web data (GWB50) and Gigaword + 100% ((GWB100)) of web data. Corpus statistics are

shown in Table 6.1. I define the context for a given word "z" 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.

Corpus	GW	GWB50	GWB100
Unzipped Size (GB)	12	60	110
# of sentences (Million)	57	463	866
# of tokens (Billion)	2.1	10.9	20.0

Table 6.1: Corpus Description

6.3.2 Evaluating Distributed online-PMI

6.3.2.1 Experimental Setup

First I conduct an intrinsic evaluation to quantitatively evaluate the distributed online-PMI vectors against the offline-PMI vectors computed from Gigaword (GW). Offline-PMI computed from the sketches have been shown as effective as exact PMI in my earlier work [53]. To compute offline-PMI vectors, I do two passes over the corpus. In the first pass, I store the counts of words, contexts and word-context pairs computed from GW in the Count-Min with conservative update (CM-CU) sketch. I use the CM-CU sketch of size 2 billion counters (bounded 8 GB memory) with 3 hash functions. In second pass, using the aggregated counts from the sketch, I generate the offline-PMI vectors of size $d = 1000$ for every word. For rest of this paper for distributed online-PMI, I set $d = 1000$ and the size of the buffer=$10,000$ and I split the data sets into small chunks of 10 million sentences.

6.3.2.2 Intrinsic Evaluation

I use four kinds of measures: precision (P), recall (R), f-measure (F1) and Pearson's correlation (ρ) to measure the overlap in the context vectors obtained using online and offline PMI. ρ is computed between contexts that are found in offline and online context vectors. I do this evaluation on 447 words selected from the concatenation of four test-sets mentioned in the next paragraph. On these 447 words, I achieve an average P of .97, average R of .96 and average F1 of .97 and a perfect average ρ of 1. This evaluation show that the vectors obtained using online-PMI are as effective as offline-PMI.

6.3.2.3 Extrinsic Evaluation

I also compare online-PMI effectiveness on four test sets which consist of word pairs, and their corresponding human rankings. I generate the word pair rankings using online-PMI and offline-PMI strategies. I report the Pearson's correlation (ρ) between the human and system generated similarity rankings. The four test sets are: **WS-353** [46] is a set of 353 word pairs. **WS-203**: A subset of WS-353 with 203 word pairs [3]. **RG-65**: [124] has 65 word pairs. **MC-30**: A subset of RG-65 dataset with 30 word pairs [100].

The results in Table 6.2 shows that by using distributed online-PMI (by making a single pass over the corpus) is comparable to offline-PMI (which is computed by making two passes over the

corpus).

For generating context vectors from GW, for both offline-PMI and online-PMI, I use a frequency cutoff of 5 for word-context pairs to throw away the rare terms as they are sensitive to PMI [21]. Next, FLAG generates online-PMI vectors from GWB50 and GWB100 and uses frequency cutoffs of 15 and 25. The higher frequency cutoffs are selected based on the intuition that, with more data, I get more noise, and hence not considering word-context pairs with frequency less than 25 will be better for the system. As FLAG is going to use the context vectors to find nearest neighbors, I also throw away all those words which have ≤ 50 contexts associated with them. This generates context vectors for $57,930$ words from GW; $95,626$ from GWB50 and $106,733$ from GWB100.

Test Set	**WS-353**	**WS-203**	**RG-65**	**MC-30**
Offline-PMI	.41	.55	.40	.52
Online-PMI	.41	.56	.39	.51

Table 6.2: Evaluating word pairs ranking with online and offline PMI. Scores are evaluated using ρ metric.

6.3.3 Evaluating Approximate Nearest Neighbor

6.3.3.1 Experimental Setup

To evaluate approximate nearest neighbor similarity lists generated by FLAG, I conduct three experiments. I evaluate all the three experiments on 447 words (test set) as used in Section 6.3.2. For each word, both exact and approximate methods return $l = 100$ nearest neighbors. The exact similarity lists for 447 test words is computed by calculating cosine similarity between 447 test words with respect to all other words. I also compare the LSH (computed using Hamming distance between all words and test set.) approximate nearest neighbor similarity lists against the exact similarity lists. LSH provides an upper bound on the performance of approximate search representations (IRP, PLEB, and FAST-PLEB) for fast-search from Section 6.2.3) . I set the number of projections $k = 3000$ for all three methods and for PLEB and FAST-PLEB, I set number of permutations $p = 1000$ as used in large-scale noun clustering work [121].

	Irp								**Pleb**								**Fast-Pleb**							
	10		25		50		100		10		25		50		100		10		25		50		100	
	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ
LSH	.55	.57	.52	.56	.49	.54	.46	.52	.55	.57	.52	.56	.49	.54	.46	.52	.55	.57	.52	.56	.49	.54	.46	.52
20	.29	.50	.26	.55	.25	.54	.24	.50	.50	.59	.45	.60	.41	.57	.37	.55	.48	.58	.42	.58	.38	.58	.35	.55
30	.36	.55	.33	.56	.31	.55	.30	.52	.53	.59	.48	.59	.44	.56	.41	.54	.51	.57	.47	.57	.42	.56	.40	.54
40	.40	.53	.38	.51	.35	.54	.34	.51	.54	.58	.50	.59	.46	.56	.43	.54	.53	.58	.49	.56	.45	.55	.42	.53
50	.44	.56	.42	.54	.39	.54	.37	.52	.54	.58	.51	.57	.47	.56	.44	.53	.54	.58	.50	.56	.46	.55	.44	.53
100	.53	.59	.49	.54	.46	.55	.43	.53	.55	.56	.52	.56	.48	.54	.46	.53	.55	.57	.52	.56	.48	.54	.46	.53

Table 6.3: Evaluation results on comparing LSH, IRP, PLEB, and FAST-PLEB with $k = 3000$ and $b = \{20, 30, 40, 50, 100\}$ with exact nearest neighbors over GW data set. For PLEB and FAST-PLEB, I set $p = 1000$ and for FAST-PLEB, I set $q = 10$. I report results on recall (R) and ρ metric. For IRP, I sample first p rows and only use p rows rather than k.

	10		25		50		100	
	R	ρ	R	ρ	R	ρ	R	ρ
IRP	.40	.53	.38	.51	.35	.54	.34	.51
q=1	.24	.62	.20	.63	.18	.59	.17	.54
q=5	.47	.60	.43	.57	.40	.57	.37	.53
q=10	.53	.58	.49	.56	.45	.55	.42	.53
q=100	.53	.60	.50	.59	.46	.56	.43	.53
q=3000	.54	.58	.50	.59	.46	.56	.43	.54

Table 6.4: Varying parameter q for FAST-PLEB with fixed $p = 1000$, $k = 3000$ and $b = 40$. Results reported on recall and ρ.

	GW								**GWB50**								**GWB100**							
	10		25		50		100		10		25		50		100		10		25		50		100	
	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ	R	ρ
LSH	.55	.57	.52	.56	.49	.54	.46	.52	.51	.55	.46	.54	.44	.52	.42	.48	.48	.58	.45	.52	.42	.49	.40	.47
IRP	.40	.53	.37	.53	.35	.54	.34	.51	.29	.50	.27	.51	.25	.51	.24	.47	.26	.57	.24	.49	.23	.48	.22	.45
PLEB	.54	.58	.50	.59	.46	.56	.43	.54	.46	.58	.42	.56	.38	.53	.36	.51	.44	.57	.40	.56	.36	.52	.33	.49
FAST-PLEB	.53	.58	.49	.56	.45	.55	.42	.53	.46	.56	.41	.56	.37	.54	.35	.51	.43	.57	.38	.55	.35	.52	.32	.50

Table 6.5: Evaluation results on comparing LSH, IRP, PLEB, and FAST-PLEB with $k = 3000$, $b = 40$, $p = 1000$ and $q = 10$ with exact nearest neighbors across three different data sets: GW, GWB50, and GWB100. I report results on recall (R) and ρ metric. The gray color row is the system that I use for further evaluations.

6.3.3.2 Evaluation Metric

I use two kinds of measures, recall and Pearson's correlation to measure the overlap in the approximate and exact similarity lists. Intuitively, recall (R) captures the number of nearest neighbors that are found in both the lists and then Pearson's (ρ) correlation captures if the relative order of these lists is preserved in both the similarity lists. I also compute R and ρ at various $l = \{10, 25, 50, 100\}$.

6.3.3.3 Results

For the first experiment, I evaluate IRP, PLEB, and FAST-PLEB against the exact nearest neighbor similarity lists. For IRP, I sample first p rows and only use p rather than k, this ensures that all the three methods (IRP, PLEB, and FAST-PLEB) take the same query time. I vary the approximate nearest neighbor beam parameter $b = \{20, 30, 40, 50, 100\}$ that controls the number of closest neighbors for a word with respect to each independent random projection. Note, with increasing b, my algorithm approaches towards LSH (computing Hamming distance with respect to all the words). For FAST-PLEB, I set $q = 10$ ($q << k$) that is the number of random bits selected out of k to generate p permuted bit vectors of size q. The results are reported in Table 6.3, where the first row compares the LSH approach against the exact similarity list for test set words. Across three columns I compare IRP, PLEB, and FAST-PLEB. For all methods, increasing b means better recall. If I move down the table, with $b = 100$, IRP, PLEB, and FAST-PLEB get results comparable to LSH (reaches an upper bound). However, using large b implies generating a long potential nearest neighbor list close to the size of the unique context vectors. If I focus on the gray color row with $b = 40$ (This will have comparatively *small* potential list and return nearest neighbors in *less* time), IRP has worse recall with best pre-processing time. FAST-PLEB ($q = 10$) is

jazz	yale	soccer	physics	wednesday
reggae	harvard	basketball	chemistry	tuesday
rockabilly	cornell	hockey	mathematics	thursday
rock	fordham	lacrosse	biology	monday
bluegrass	rutgers	handball	biochemistry	friday
indie	dartmouth	badminton	science	saturday
baroque	nyu	softball	microbiology	sunday
ska	ucla	football	geophysics	yesterday
funk	princeton	tennis	economics	tues
banjo	stanford	wrestling	psychology	october
blues	loyola	rugby	neuroscience	week

Table 6.6: Sample Top 10 similarity lists returned by FAST-PLEB with $k = 3000$, $p = 1000$, $b = 40$ and $q = 10$ from GWB100 data set.

comparable to PLEB (using all bits $q = 3000$) with pre-processing time 300 times faster than PLEB. For rest of this work, FLAG will use FAST-PLEB as it has best recall and pre-processing time with fixed $b = 40$.

For the second experiment, I vary parameter $q = \{1, 5, 10, 100, 3000\}$ for FAST-PLEB in Table 6.4. Table 6.4 demonstrates using $q = \{1, 5\}$ result in worse recall, however using $q = 5$ for FAST-PLEB is better than IRP. $q = 10$ has comparable recall to $q = \{100, 3000\}$. For rest of this work, I fix $q = 10$ as it has best recall and pre-processing time.

For the third experiment, I increase the size of the data set across the Table 6.5. With the increase in size of the data set, LSH, IRP, PLEB, and FAST-PLEB ($q = 10$) have worse recall. The reason for such a behavior is that the number of unique context vectors is greater for big data sets. Across all the three data sets, FAST-PLEB has recall comparable to PLEB with best pre-processing time. Hence, for the next evaluation to show the quality of final lists I use FAST-PLEB with $q = 10$ for GWB100 data set.

In Table 6.6, I list the top 10 most similar words for some words found by system FLAG using GWB100 data set. Even though FLAG's approximate nearest neighbor algorithm has less recall with respect to exact but still the quality of these nearest neighbor lists is excellent.

For the final experiment, I demonstrate the pre-processing and query time results comparing LSH, IRP, PLEB, and FAST-PLEB with $k = 3000$, $p = 1000$, $b = 40$ and $q = 10$ parameter settings. For pre-processing timing results, I perform all the experiments (averaged over 5 runs) on GWB100 data set with 106, 733 words. The second pre-processing step of the system FLAG (Section 6.2.2) that is dimensionality reduction from $\mathbb{R}^D$ to $\mathbb{R}^k$ took 8.8 hours. The pre-processing time differences among IRP, PLEB, and FAST-PLEB from third step (Section 6.2.3) are shown in second column of Table 6.7. Experimental results show that the naive baseline IRP is the fastest and FAST-PLEB is 2 orders of magnitude *faster* than PLEB.

For comparing query time among several methods, I evaluate over 447 words (Section 6.3.2). We report average timing results (averaged over 10 runs and 447 words) to find top 100 nearest neighbors for single query word. The results are shown in third column of Table 6.7. Comparing first and second rows show that LSH is 87 times faster than computing exact top-100 (cosine similarity) nearest neighbors. Comparing second, third, fourth and fifth rows of the table demonstrate that IRP, PLEB and FAST-PLEB methods are twice as fast as LSH.

Methods	Preprocessing	Query (seconds)
Exact	n/a	87
LSH	8.8 hours	0.59
IRP	7.5 minutes	0.28
PLEB	1.8 days	0.28
FAST-PLEB	22 minutes	0.26

Table 6.7: Preprocessing and query time results comparing exact, LSH, IRP, PLEB, and FAST-PLEB methods on GWB100 data set.

Language	english	chinese	japanese	spanish	russian
Place	africa	america	washington	london	pacific
Nationality	american	european	french	british	western
Date	january	may	december	october	june
Organization	ford	microsoft	sony	disneyland	google

Table 6.8: Query terms for Google Sets Problem evaluation

6.4 Applications

I use the graph constructed by FLAG from GWB100 data set (110 GB) by applying FAST-PLEB with parameters $k = 3000$, $p = 1000$, $q = 10$ and $b = 40$. The graph has $106,733$ nodes (words), with each node having 100 edges that denote the top $l = 100$ approximate nearest neighbors associated with each node. However, FLAG applied FAST-PLEB (approximate search) to find these neighbors. Therefore many of these edges can be noisy for many applications. Hence for each node, I only consider top 10 edges. In general for graph-based NLP problems; for example, constructing web-derived polarity lexicons [159], top 25 edges were used, and for unsupervised part-of-speech tagging using label propagation [30], top 5 edges were used.

6.4.1 Google Sets Problem

Google Sets problem [50] can be defined as: given a set of query words, return top similar words with respect to query words. To evaluate the quality of the approximate large-scale graph, I return top 25 words which have best aggregated similarity scores with respect to query words. I take 5 classes and their query terms [97] shown in Table 6.8 and the goal is to learn 25 *new* words which are similar with these 5 query words.

I conduct a manual evaluation to directly measure the quality of returned words. I recruited 1 annotator and developed annotation guidelines that instructed each recruiter to judge whether learned values are similar to query words or not. Overall the annotator found almost all the learned words to be similar to the query words. However, the algorithm can not differentiate between different senses of the word. For example, "French" can be a language and a nationality.

Language	german	french	estonian	hungarian	bulgarian
Place	scandinavia	mongolia	mozambique	zambia	namibia
Nationality	german	hungarian	estonian	latvian	lithuanian
Date	september	february	august	july	november
Organization	looksmart	hotbot	lycos	webcrawler	alltheweb

Table 6.9: Learned terms for Google Sets Problem

Concrete seeds	car, house, tree, horse, animal man, table, bottle, woman, computer
Abstract seeds	idea, bravery, deceit, trust, dedication anger, humour, luck, inflation, honesty

Table 6.10: Example seeds for bootstrapping.

Concrete: girl, person, bottles, wife, gentleman, microcomputer, neighbor, boy, foreigner, housewives, texan, granny, bartender, tables, policeman, chubby, mature, trees mainframe, backbone, truck
Abstract: perseverance, tenacity, sincerity, professionalism, generosity, heroism, compassion, commitment, openness, resentment, treachery, deception, notion, jealousy, loathing, hurry, valour

Table 6.11: Learned concrete/abstract words.

Table 6.9 shows the top ranked words with respect to query words.

6.4.2 Learning Concrete and Abstract Words

Here the goal is to automatically learn concrete and abstract words [149]. I apply bootstrapping [122, 24, 123, 78] on the word graphs by manually selecting 10 seeds for concrete and abstract words (see Table 6.10). I use in-degree (sum of weights of incoming edges) to compute the score for each node which has connections with known (seeds) or automatically labeled nodes, previously exploited to learn hyponymy relations from the web [78]. I learn concrete and abstract words together (known as mutual exclusion principle in bootstrapping [147, 97]), and each word is assigned to only one class. Moreover, after each iteration, I harmonically decrease the weight of the in-degree associated with instances learned in later iterations. I add 25 new instances at each iteration and ran 100 iterations of bootstrapping, yielding 2506 concrete nouns and 2498 abstract nouns. To evaluate the learned words, I searched in WordNet whether they had 'abstraction' or 'physical' as their hypernym. Out of 2506 learned concrete nouns, 1655 were found in WordNet. According to WordNet, **74**% of those are *concrete* and 26% are abstract. Out of 2498 learned abstract nouns, 942 were found in WordNet. According to WordNet, 5% of those are concrete and **95**% are *abstract*. Table 6.11 shows the top ranked concrete and abstract words.

6.5 Conclusion

I proposed a system, FLAG which constructs fast large-scale approximate graphs from large data sets. To build this system I proposed a distributed online-PMI algorithm that scaled up to 110 GB of web data with 866 million sentences in less than 2 days using 100 quad-core nodes. Both intrinsic and extrinsic experiments demonstrated that online-PMI algorithm not at all loses globally good contexts and perform comparable to offline-PMI. Next, I proposed Fast-Pleb (a variant of Pleb) and empirically demonstrated that it has recall comparable to Pleb with 2

orders of magnitude faster pre-processing time. Finally, I show the applicability of FLAG on two applications: Google-Sets problem and learning concrete and abstract words.

Chapter 7

Conclusions and Future Directions

In this dissertation, I have:

- Demonstrated that many Natural Language Processing (NLP) tasks can boil down to tracking many statistics from large corpora.
- Demonstrated that large data + streaming and sketch algorithms can efficiently and effectively solve many large-scale NLP problems.
- Presented streaming and sketch algorithms that provide memory-efficient solutions for an average user to exploit large data. At the same time, these algorithms can easily be implemented in distributed setting and provide memory- and time-efficient solution for commodity cluster users.
- Shown that these memory and time savings come at the expense of approximate solutions; however, the approximate solutions achieved on large data are comparable to exact solutions on large data, and outperform exact solutions on smaller data.
- Developed FLAG, a system that quickly constructs a large-scale approximate nearest-neighbor graph from a large text corpus.

The applications described in this dissertation have shown that streaming and sketch algorithms provide a general framework for solving many large-scale NLP problems. Most of the problems considered in this dissertation are canonical NLP problems, however the algorithms proposed in this dissertation will be applicable to many other application areas like computer vision, machine learning, knowledge mining, social media, and information retrieval.

In Chapter 6, I demonstrated the applicability of FLAG on two NLP applications: the Google-Sets problem, and learning concrete and abstract words. In this chapter, I present four extensions to FLAG:

1. The use of multiple kinds of contexts that can be used for construction of nearest neighbor graphs (Section 7.1)
2. Heuristics that can be used to remove noise from approximate nearest neighbor graphs (Section 7.2)
3. An approximate version of clustering by committee (CBC) [113], a canonical example for several clustering methods (Section 7.3)
4. Algorithmic techniques that can be used to improve the efficiency and effectiveness of FLAG (Section 7.4)

I then conclude the dissertation by summarizing the important contributions and providing future directions (Section 7.5).

I make two simplifying modifications to FLAG to conduct large set of experiments.[1] First, instead of using online-PMI, I follow the conventional offline-PMI framework.[2] Second, I return the nearest neighbors found by LSH and skip the FAST-PLEB part (For details, refer to Chapter 6)[3]. I keep the same parameter settings of FLAG as used in Chapter 6. I use version 5 of the Gigaword (GW) [62] corpus. It is a newer version of Gigaword compared to the version of data used in Chapter 3, 4, 5 and 6. The uncompressed size of text data is 24 GB. It contains 181 million sentences, 4.8 billion tokens and 2.6 million types. I split the text into sentences, tokenize, and convert into lower-case.

7.1 Several Kinds of Contexts

In this section, FLAG generates nearest neighbor graphs using several kinds of contexts around a target word. The motivation behind considering these several contexts is to show that a specific kind of context is essential for solving a specific downstream application.[4] The different context settings considered are:

1. **Collocation:** I use collocation by fixing the position of words on either side of the target word. I keep track of the position of the word, e.g., -2, -1, $+1$, $+2$. This kind of context captures syntactic similarity. This context was used in previous chapter by FLAG.[5]

2. **Bag-of-words:** I considered a window of size 7 on the left and right of the target word to generate word pairs (by using the target word and one of the words from the surrounding window). This kind of context captures topical similarity. I selected 7 as our window size in Chapter 4 and showed the effectiveness of storing word pairs within a window size of 7 over three NLP problems.

3. **Direction:** I keep track of direction − the context "left" and "right" of the target word. I used 4 words on the left and 4 words on the right by fixing their direction. This kind of context captures aspects of both syntactic and topical similarity. This setting is similar to the one used by Turney et al. [149] in identifying literal and metaphorical sense through concrete and abstract context.

   Qualitative Analysis: As preliminary results, I list the top 10 most similar words for sample words found by FLAG on these different kinds of contexts. Table 7.1, 7.2 and 7.3 report these results for collocation, bag-of-words and direction respectively. We can make two observations from these tables: (1) The quality of the nearest neighbor lists is good. (2) The different contexts find different top 10 nearest neighbors and capture different notions of similarity like "relatedness" and "similarity" (For more details refer to Agirre et al. [3]).

[1] The experiments and results reported in this chapter were conducted at Raytheon BBN Technologies.

[2] Online-PMI provides a memory-efficient framework and is more viable when there is limited memory.

[3] Chapter 6 showed that using FAST-PLEB on top of LSH finds nearest neighbors two times faster than LSH, with comparable recall to LSH. This design was made to keep the system simple and perform large-scale experiments.

[4] Section 7.2.1 compares several graph constructions based on different contexts on the task of semantic orientation.

[5] Note, we can easily change the window size on the left and right (according to a specific task or domain) for all different kinds of contexts.

airplane	president	law	congress	vote
aircraft	presient	laws	parliament	votes
plane	presdient	statue	legislature	election
airliner	president	legislation	lawmakers	balloting
jetliner	presidents	statues	legislatures	ballot
aeroplane	presiden	provision	governments	elections
jet	counterpart	constitution	dole	polls
seaplane	pres.	amendment	delegates	voted
helicopter	president	rules	mhp	referendum
warplane	president	ordinance	capitol	runoff
chopper	prsident	regulations	conferees	childsex

Table 7.1: Sample Top 10 similarity lists returned by FLAG with $k = 3000$ from Gigaword dataset using **collocation** context.

airplane	president	law	congress	vote
plane	presidential	laws	lawmakers	votes
jetliner	presidency	enacted	legislature	election
airplanes	elected	provision	congressional	voted
aircraft	met	statute	approve	elections
flight	elect	statutes	representatives	referendum
jetliners	presidents	legislation	legislation	voting
jet	w.	constitution	proposal	ballot
planes	clinton	act	senate	polls
boeing	election	regulations	legislators	parliamentary
airliners	sworn	amendment	democrats	approve

Table 7.2: Sample Top 10 similarity lists returned by FLAG with $k = 3000$ from Gigaword dataset using **bag-of-words** context.

airplane	president	law	congress	vote
plane	counterpart	laws	legislature	votes
aircraft	elect	statute	lawmakers	election
jet	presdient	act	parliament	voted
jetliner	presient	statutes	senate	elections
airliner	presidents	constitution	bush	ballot
jetliners	preisdent	legislation	knesset	referendum
turboprops	incumbent	enacted	legislatures	voting
cessna	presidnet	regulations	congressional	polls
planes	prsident	ordinance	assembly	parliamentary
aeroplane	preident	amendment	bundesrat	revote

Table 7.3: Sample Top 10 similarity lists returned by FLAG with $k = 3000$ from Gigaword dataset using **direction** context.

airplane	president	law	congress	vote
aircraft	presient	laws	parliament	votes
plane	presdient	statute	legislature	election
airliner	preisdent	legislation	lawmakers	balloting
jetliner	presidents	statutes	legislatures	ballot
aeroplane	counterpart	provision	governments	elections
jet	pres.	constitution	capitol	polls
seaplane	leader	amendment	senate	voted
warplane	elect	rules	assembly	referendum
turboprop	incumbent	ordinance	washington	referendums
cessna	candidate	regulations	clinton	primaries

Table 7.4: Sample Top 10 similarity lists returned by FLAG using **Intersect-I** graph.

airplane	president	law	congress	vote
aircraft	leader	laws	parliament	votes
jetliner	elect	legislation	legislature	election
jet	incumbent	provision	lawmakers	balloting
helicopter	candidate	constitution	legislatures	ballot
turboprop	premier	amendment	dole	elections
cessna	manager	rules	delegates	polls
planes	governor	regulations	capitol	referendum
airplanes	hussein	rule	coalition	runoff
boat	president	provisions	statehouses	referendums
glider		sharia	spending	primaries

Table 7.5: Sample Top 10 similarity lists returned by FLAG using **Intersect-II** graph.

7.2 Removing noisy nearest neighbors

The nearest neighbor graphs generated by FLAG in the previous section are good but still noisy (Table 7.1, 7.2, 7.3). Ideally, we would like to capture synonym-like similarity. To remove noise from the nearest neighbor graphs, I exploit ideas from researchers [78, 74] who used template patterns like "countries such as Spain and *" to find other instances of "countries" and instances for other semantic classes. The fundamental notion used above is to use a context word like "countries" to restrict the pattern so it finds precise instances of names of "countries". However, these pattern-based approaches are very restrictive and only have reasonable recall with web-sized datasets. As an approximation of this method, I apply two different heuristics to the nearest neighbor graph generation.

First, for each word, I take an intersection between the top 100 bag-of-words neighbors and top 100 collocation neighbors. This generates a more precise graph, capturing both syntactic and topical similarity and removing noise. I refer to the graph generated by this intersection as "Intersect-I". However, this heuristic is highly restrictive. Second, I take an intersection between the context vectors of the bag-of-words (1000 contexts) nearest neighbor graph and the collocation (100 neighbors) nearest neighbor graph. This heuristic captures the same notion as above, however this is less restrictive. I refer to the graph generated by this intersection as "Intersect-II".

Table 7.4 and 7.5 report the top 10 most similar words for sample words from Intersect-I and Intersect-II graphs respectively.

7.2.1 Comparing various graphs for the Semantic Orientation task

I compare various graph constructions (based on different kinds of contexts) on the task of semantic orientation [64, 110, 151]. The goal of this comparison study is to show that the right kind of graph construction is essential for solving a specific task.

Given a word, the task of finding its semantic orientation is to identify if the word is used in positive or negative sense. I apply the graph propagation algorithm of Velikovich et al. [159], a variant of label propagation algorithms [167, 145] that has been shown to be effective for inducing a web-scale semantic orientation lexicon. I apply the graph propagation algorithm on 5 different graphs (discussed in Section 7.1 and 7.2): Collocation, Bag-of-words, Direction, Intersect-I, and Intersect-II.

Seeds: I use 57 negative and 44 positive seeds. The seeds were manually selected by a native English speaker.

Test Set: I use the General Inquirer lexicon[6] [139] as a benchmark to evaluate the semantic orientation scores similar to Turney's work [151]. Words with multiple senses have multiple entries in the lexicon, I merge these entries for the experiment. The test set consists of 1575 positive and 1954 negative words (excluding seed words). Accuracy is used as an evaluation metric.

Results: We can make following observations from Table 7.6:

1. Bag-of-words context does better than Collocation and Direction context. This is due to the reason that Bag-of-words capture topical similarity. Moreover, Turney and Littman's state-of-the-art [151] on semantic orientation used the local bag-of-words context.

2. Intersect-I and Intersect-II have better accuracy than other contexts. This shows that the heuristic for removing noise in graphs is beneficial.

Graph Type	Accuracy
Collocation	64.41
Bag-of-words	71.26
Direction	67.06
Intersect-I	75.38
Intersect-II	77.59

Table 7.6: Semantic Orientation results using label propagation

7.3 Approximate Clustering by Committee

I describe an approximate version of clustering by committee (CBC) [113]. I perform clustering over the Locality Sensitive Hashing (Lsh) representations of words rather than their original vector-space representation. Note that Lsh representations (bit-vectors) can be used for several other clustering methods. I use CBC clustering as a canonical example for several clustering methods. CBC clustering has been shown more effective than several other clustering methods in NLP.

I run the CBC clustering algorithm on nearest neighbor graphs generated in Section 7.1. I consider each word's top 10 neighbors and use the collocation nearest graph as running example to describe approximate CBC. There are 138,292 words in the collocation graph.

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

Committee	Committee Score
purple green scarlet yellow blue red white pink	11346.8
taupe turquoise pink chartreuse fuchsia teal fuschia magenta purple	11206.4
burgundy taupe beige turquoise blue aubergine mauve magenta fuchsia	10833.6
magenta purple teal pink beige orange yellow blue red	10829.3
yellow blue red beige pink camouflage white black gray	10066
pink sequined colored lilac fuchsia chartreuse teal magenta purple	9714.22
orange red blue yellow green pink	9561.33
turquoise taupe teal fuchsia chartreuse magenta	9560
pink beige white blue yellow	9435.2
purple mauve lilac fuchsia turquoise teal yellow blue	9227.75
periwinkle purple lavender mauve fuchsia lilac pink	9100
yellow pink magenta mauve purple lilac	8983.33
purple green blue red gray beige	8927.33
ecru turquoise colored maroon taupe beige khaki pink	8887.25
magenta mauve aubergine burgundy teal	8661.2
colored grey gray white black blue pink	8630.86
red blue white scarlet crimson chartreuse fuschia purple mauve	8522
fuchsia mauve maroon beige taupe khaki blue	8171.71
chartreuse fuchsia teal mauve aubergine	8039.6
teal fuchsia ocher ochre mauve	8028.8
taupe beige magenta teal chartreuse blue colored aquamarine turquoise	7899.56
pink red yellow blue	7607
beige black white	6720
teal pink beige lavender black purple crimson	6676.57
crimson periwinkle blue beige	5787.5
blue beige	5136
blue	3316

Table 7.7: Example of committees that are similar with respect to each other.

In the first phase, CBC applies Hierarchal Agglomerative Clustering (HAC) over each word and its top 10 nearest neighbors. After applying HAC, CBC selects the highest-scoring cluster (over all HAC clusters) for each of the words and its neighbors. The average pairwise similarity between words in clusters is used to select the highest-scoring cluster. Pantel and Lin [113] referred to the highest-scoring clusters as committees. For our running example, CBC applies HAC 138,292 times and produces 138,292 committees. As these committees are generated over a word and its top 10 similar words, there can be many similar committees. Table 7.7 shows a example (system output over collocation graph) of the committees that are similar with respect to each another. This step of CBC is fast in spite of running HAC number of words (138,292) times. The reason for HAC being fast is that, for each iteration, it is applied only over a word and its top 10 neighbors.

In the second phase, CBC removes redundant committees. Removing redundant committees involves first finding the centroid of feature vectors of words in the committee. In order to expedite this process, we compute the centroid over the Locality Sensitive Hashing (Lsh) representation. Next, we sort the committees in descending order of their average pairwise similarity score between centroids. We traverse the sorted list of committees. If a committee's similarity (based on Hamming distance using the Lsh representation) to previously selected committees is below a threshold (τ_1), then we add this committee to the unique committee list. This step involves computing pairwise similarities between committees. Hence, using Lsh makes this step faster.

In the final phase, CBC assigns words to their closest committees if their similarity to the closest committee is above a threshold (τ_2). To compute its closest committees, each word (using its Lsh representation) computes Hamming distance with each of the committees (using its Lsh representation). Here again, using Lsh makes this step faster.

Qualitative Analysis: Next, as preliminary results, I list the sample committees and their word assignments found by approximate CBC in Table 7.8. I set parameters $\tau_1 = 0.55$ and $\tau_2 = 0.55$. These parameter settings producing a final output of 15,402 committees and their word assignments. Table 7.8 shows that the quality of the approximate clusters is reasonable. There is still some noise in the clustering output; however, the speedup gains are more substantial than the small degree of noise in the output. Moreover, some noise in the clustering output should not affect the downstream applications.

7.4 Algorithmic advancements for FLAG

In this section, I first report a negative result. I explored research on feature hashing [132, 161, 130, 136, 59] to make FLAG more amenable to handling dynamic graph updates from time-varying data to improve its training time. The goal here was to replace the online-PMI algorithm with feature hashing. However, qualitative nearest neighbor results using feature hashing were not promising. Table 7.9 reports the top 10 most similar words for sample words using the feature hashing algorithm from Weinberger et al. [161] with $k = 10,000$ low-dimensional representation. The reason behind this was that feature hashing is a streaming method to handle features that arrive one at a time, and it cannot do a good job at dimensionality reduction. Ideally, we want a solution that can handle streaming features and embed them directly into a low-dimensional bit-vector representation. However, feature hashing does not have any of these properties. It handles streaming features by embedding them into a high-dimensional feature representation rather than a low-dimensional representation, and it does not provide a bit-vector representation, making finding nearest neighbors expensive.

FLAG uses Lsh because of its speed. Lsh can quickly construct bit-vector representations by projecting data onto random directions and mapping projected values to {0,1} by thresholding.

Committee	Word Assignments
airliners, jetliners, jet, airplane, airliner, jetliner, plane, planes, aircraft, warplane	widebody warplanes warplane vessels vessel vehicle turboprops turboprop tupolev truck tomcat tanker superjumbos superjumbo subsonic sq000 speedboat spaceship sorties skyhawk ships ship seaplane rocket planes plane pilots pilot passenger motorboat minibuses minibus migs md00 learjet jumbo jets jetliners jetliner jetfighters jet helicopters helicopter gunship gulfstream glider flights flight fleet fighter ferryboat f00s dreamliner dc0 copter concorde choppers chopper cessna canadair c000 boeings boeing blimp biplane beechcraft balloon b000 atr antonov airplanes airplane airlines airliners airliner airline aircrafts aircraft aircaft airbuses airbus aeroplanes aeroplane a000s
october, february, november, april, march, june, july, december, september, august	women: week wednesday washington unadjusted tokyo thursday surpassing sunday statscan smucker september sept saturday recap peaking paving october oct oct. nymex november nov. month monday mid march march00 mar. london l000 june july january janaury jan. friday feburary febuary february feb feb. enraging doha december dec. cotonou chalking bonn barrancabermeja bangkok august aug april apr. addis
tram, streetcar, lrt, transit, eurostar, rail, railroad, train, subway	trolley transrapid transit tramway trams tram trains train ticketing thalys tgv subway streetcar stagecoach southbound skytrain shinkansen riverboat rer railways railway railroad rail parking northbound mrt motorcoach monorail metrolink metroliner mbta maglev lrt livery levitation jitney intermodal intercity greyhound gondola freight ferry eurostar eastbound commuter car caltrain bus amtrak airtrain acela
presiden, pres., presidents, president, counterpart, presidnet, preisdent, presient, president, presdient	prsident proverb prez presient presidnet presidents president presiden presdient pres. premier preisdent preident predecessor pesident nationality minister leader incumbent excellency elect counterpart
malaysian, australian, american, german, french, british, canadian, u.s., chinese, japanese	zimbabwean zealand zambian zairian zairean yugoslavian yugoslav yemeni vietnamese verdean venezuelan vanuatuan uruguayan ukranian ukrainian ugandan u.s. u.k. turkmen turkish tunisian trinidadian tongan thai tanzanian tajik taiwanese syrian swiss swedish swazi surinamese sudanese spanish spainish soviet somalian somali slovenian slovene slovak skorean singaporean seychellois serbian senegalese scottish salvadorian salvadoran russian romanian rok roc rican regional qatari private portuguese portugese polish philippine peruvian paraguayan panamanian palestinian palestinan palauan pakistani omani okinawan norweigan norwegian nordic nigerien nigerian nicaraguan nepali nepalese national namibian myanmese myanma mozambican moroccan montenegrin moldovan moldavian milanese micronesian mexican mauritian mauritanian maltese malian maldivian malaysian malawian madagascan macedonian local lithuanian libyan liberian leonean lebanese latvian laotian lankan kyrgyz kuwaiti korean kong kenyan kazakhstani kazakh kazak jordanian japanse japanese jamaican ivorian ivoirian italian isreali israeli irish iraqi iranian international ingush indonesian indian icelandic hungarian honduran hellenic haitian guyanese guyanan guinean guatemalan grenadian greek global ghanian ghanaian german georgian gambian gabonese french foreign fledging finnish filipino fijian europen european europan eu ethiopian estonian eritrean emirati egyptian ecuadorian ecuadorean ecuadoran dutch dominican domestic djiboutian danish dagestani czechoslovakian czechoslovak cypriot cuban croatian comorian comoran colombian civilian chinese chines chilean chechnyan chechen chadian catalan candian canadian cambodian burundian burmese burkinabe bulgarian bruneian british brazilian bosnian bolivian bhutanese beninese belorussian belizean belgian belarussian belarusian belarusan barbadian bangladeshi bahraini bahamian azerbaijani automobile austrian australian austalian asian aruban armenian argentine arabian arab antiguan angolan andorran american algerian albertsons albanian african afghani afghan abkhazian *chinese

Table 7.8: Examples of system output of Approximate CBC. Table shows committees and their word assignments.

jazz	yale	soccer	physics	wednesday
starzz	emory	football	mpshe	thursday
today's	stanford	baseball	koncius	tuesday
waitstaff	tufts	basketball:	dali	monday
fountainheads	makerere	ncpfs	accelerators	friday
medallists	tulane	chambord	tykes	saturday
khanijoh	tunghai	linesmen	wanniarachchi	sunday
corncob	birzeit	~~~~~~~~~~	d'adet	yesterday
medallist	tamkang	champions	kezirian	midsummer
ferzan	keio	footbal	atenco	tomorrow
medalists	fudan	omiya	hillenbrand	wedneday

Table 7.9: Sample Top 10 similarity lists returned by feature hashing with $k = 10,000$ from Gigaword data set.

However, LSH-generated bit-vector representations do not make use of data to learn this representation. A learned representation is important, as it can generate compact and better representations, and in turn improving the speed and performance of the system. Hence, for future work, LSH could be replaced in FLAG by a sophisticated hashing method like semantic hashing [126], spectral hashing [163], sparse spectral hashing [131], multidimensional spectral hashing [162] and other recent work [4, 68, 106] to learn bit-vector representations.

FLAG can also benefit from the research on multi-probe LSH [91] that is built upon the idea of entropy-based LSH [111]. Multi-probe LSH has been shown more space- and time-efficient than basic LSH. Another issue with LSH is that it has several parameters to tune for a specific data-set. To address this concern, Dong et al. [37] presented automatic parameter tuning with a performance model for Multi-probe LSH.

FLAG can also take advantage of research on Bayesian LSH [127] that focuses on reducing query time for finding nearest neighbors of unseen data points (words) in training data. The proposed method is very effective in pruning away the majority of the false positives by examining only a few hash bits. The number of hashes which are compared for each candidate pair is determined automatically. Hence, each data point is only hashed as many times as required.

The variant of Point Location in Equal Balls (FAST-PLEB) which is part of FLAG can be beneficial for finding approximate nearest neighbors in other fields like information retrieval [148], speech [69], and image processing [65].

Seminal work on minwise hashing [12, 11] has been applied as a general technique for estimating approximate set similarity. It has been applied to a wide range of applications, for example, web spam [153], detecting near-duplicates for web crawling [94], similarity caching for contextual advertising systems [109], and detection of large-scale redundancy in enterprise file systems [48]. There has been recent work in b-bit minwise hashing [88, 90, 89] to improve the space and time efficiency of basic minwise hashing. FLAG applied in the context of document similarity can leverage recent work on b-bit minwise hashing instead of LSH.

FLAG can also take advantage of work on conditional random sampling (CRS) [85, 83, 86]. CRS is a sampling method used to approximate statistics like cosine, resemblance, mutual information, and correlation. CRS has been applied for language applications like estimating associations [84] and discourse modeling [75].

7.5 Conclusions

In this dissertation, I demonstrated that large data + streaming and sketch algorithms can efficiently and effectively solve many large-scale NLP problems. I particularly focused on NLP applications, where many NLP problems boil down to tracking many statistics.

First, I applied a deterministic streaming algorithm to build approximate streaming large-scale language models. The streaming algorithm, named Lossy Counting, computed approximate frequency counts of frequently occurring n-grams. This is the *first* work that introduced the concept of approximate streaming large-scale language models in NLP. Next, I showed the empirical effectiveness of approximate streaming language models on the statistical machine translation task.

Second, I focused my attention on maintaining approximate counts of *all items* instead of just focusing on storing frequent items. I explored sketch algorithms that represent a large dataset with a compact summary, typically much smaller than the full size of the input. Here I demonstrate that a novel variant of the Count-Min sketch accurately solves three large-scale NLP problems using *small bounded memory footprint*.

Third, I conducted a systematic study and compared many sketch algorithms that approximate counts of items with a focus on large-scale NLP tasks. Here, I showed that, across many NLP tasks, *one* sketch (a variant of Count-Min) performs best. This result shows that maintaining information about low frequency items (even with over-estimation error) is better than throwing away information (under-estimation error) about low frequency items.

Last, I focused on the more general and broad NLP problem of constructing large-scale nearest-neighbor graphs in a memory- and time-efficient manner. To achieve this goal, I exploited research on Count-Min sketch, online pointwise mutual information, Locality Sensitive Hashing, random projections and improve on previous work done on Point Location in Equal Balls (Pleb). I proposed a novel variant of Pleb and demonstrated empirically that the variant has 120 times faster pre-processing time with comparable recall.

In future work, FLAG can be extended to finding the meaning of a word according to its given context. For example, in phrases "catch a ball" and "catch a disease", "catch" has two different meanings. In the first phrase, "catch" stands for "holding" and in the second, it stands for "contract". Prior work [33, 34] in NLP has proposed various methods to tackle this problem. However, most of the proposed solutions [43, 146] rely heavily on dependency parse information. However, dependency parsers are not readily available for many languages and perform poorly on noisy English text. Hence, in the future, FLAG will be extended to infer meaning in context (from large raw text) without relying on *any* dependency parse data.

This work can benefit large-scale problems where we have data from multiple modalities (like text, speech, image, and video) such as recent work [36, 101, 7] on learning relationships between visually descriptive text and images. This work can also be leveraged to build automatic large-scale knowledge bases that, in turn, can enable a deeper understanding of human language. For example, this work can indirectly benefit narrative text understanding[15, 16, 40, 60, 61]; identifying metaphors [95, 133, 134, 135] in text; and question answering [71, 104, 70, 72, 79]. The streaming and sketch algorithms discussed in this dissertation can also benefit many social media applications (e.g., disease outbreak prediction [99, 29] and first story detection [119, 118]).

Bibliography

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4), 2003.

[2] Charu C. Aggarwal and Philip S. Yu. On classification of high-cardinality data streams. In *SIAM Conference on Data Mining (SDM)*, 2010.

[3] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 2008.

[5] S. Banerjee and T. Pedersen. The Design, Implementation, and Use of the Ngram Statistics Package. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, 2003.

[6] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

[7] Alexander C. Berg, Tamara L. Berg, Hal Daumé III, Jesse Dodge, Amit Goyal, Xufeng Han, Alyssa Mensch, Margaret Mitchell, Aneesh Sood, Karl Stratos, and Kota Yamaguchi. Understanding and predicting importance in images. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[8] Shane Bergsma, Dekang Lin, and Randy Goebel. Discriminative learning of selectional preference from unlabeled text. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 2008.

[9] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13, 1970.

[10] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 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*, 2007.

[11] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences, IEEE Computer Society*, 1998.

[12] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13), 1997.

[13] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 1990.

[14] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 1993.

[15] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. In *Proceedings of the Association for Computational Linguistics*, 2008.

[16] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Association for Computational Linguistics*, 2009.

[17] Nathanael Chambers and Dan Jurafsky. Improving the use of pseudo-words for evaluating selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[18] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theorical Computer Science*, 312, 2004.

[19] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, 2002.

[20] S. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[21] K. Church and P. Hanks. Word Association Norms, Mutual Information and Lexicography. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 1989.

[22] S. B. Cohen and N. A. Smith. Covariance in unsupervised learning of probabilistic grammars. *Journal of Machine Learning Research (JMLR)*, 11, 2010.

[23] Saar Cohen and Yossi Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003.

[24] M. Collins and Y. Singer. Unsupervised Models for Named Entity Classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-99)*, 1999.

[25] Graham Cormode. Encyclopedia entry on 'Count-Min Sketch'. In *Encyclopedia of Database Systems*. Springer, 2009.

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

[27] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. In *The International Journal on Very Large Data Bases*, 2008.

[28] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 2004.

[29] Aron Culotta. Lightweight methods to estimate influenza rates and alcohol sales volume from Twitter messages. *Language Resources and Evaluation, Special Issue on Analysis of Short Texts on the Web*, 2012.

[30] Dipanjan Das and Slav Petrov. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[31] E.D. Demaine, A. Lopez-Ortiz, and J.I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, 2002.

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

[33] Georgiana Dinu and Mirella Lapata. Measuring distributional similarity in context. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, October 2010.

[34] Georgiana Dinu, Stefan Thater, and Soeren Laue. A comparison of models of word meaning in context. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012.

[35] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, 2002.

[36] Jesse Dodge, Amit Goyal, Xufeng Han, Alyssa Mensch, Margaret Mitchell, Karl Sratos, Kota Yamaguchi, Yejin Choi, Hal Daumé III, Alexander C. Berg, and Tamara L. Berg. Detecting visual text. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012.

[37] Wei Dong and Zhe Wang. Modeling LSH for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)*, 2008.

[38] Gregory Druck, Gideon Mann, and Andrew McCallum. Semi-supervised learning of dependency parsers using generalized expectation criteria. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1*, 2009.

[39] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *Proceedings of Innovations in Computer Science (ICS)*, 2010.

[40] David Elson and Kathleen McKeown. Extending and evaluating a platform for story understanding. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2009 Spring Symposium on Intelligent Narrative Technologies II*, 2009.

[41] Ahmad Emami, Kishore Papineni, and Jeffrey Sorensen. Large-scale distributed language modeling. In *Proceedings of the 2007 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2007.

[42] Katrin Erk. A simple, similarity-based model for selectional preferences. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007.

[43] Katrin Erk and Sebastian Padó. A structured vector space model for word meaning in context. 2008.

[44] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 32 of *SIGCOMM*, 2002.

[45] Marcello Federico and Nicola Bertoldi. How many bits are needed to store probabilities for phrase-based translation? In *Proceedings on the Workshop on Statistical Machine Translation at ACL06*, 2006.

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

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

[48] George Forman, Kave Eshghi, and Jaap Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *Operating Systems Review*, 43(1), 2009.

[49] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2004.

[50] Zoubin Ghahramani and Katherine A. Heller. Bayesian Sets. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2005.

[51] James Gorman and James R. Curran. Scaling distributional similarity to large corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

[52] Amit Goyal, Graham Cormode, and Hal Daumé III. Sketch algorithms for estimating point queries in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and and Computational Natural Language Learning*, 2012.

[53] Amit Goyal and Hal Daumé III. Approximate scalable bounded space sketch for large data NLP. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011.

[54] Amit Goyal and Hal Daumé III. Lossy conservative update (LCU) sketch: Succinct approximate count storage. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, 2011.

[55] Amit Goyal, Hal Daumé III, and Raul Guerra. Fast large-scale approximate graph construction for NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and and Computational Natural Language Learning*, 2012.

[56] Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. Streaming for large scale NLP: Language modeling. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[57] Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. Sketch techniques for scaling distributional similarity to the web. In *Proceedings of the ACL 2010 Workshop on GEometrical Models of Natural Language Semantics*, 2010.

[58] Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. Sketching techniques for Large Scale NLP. In *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*, 2010.

[59] Amit Goyal, Piyush Rai, and Hal Daumé III. Multiple hash functions for learning. In *Big Learning workshop at Neural Information Processing Systems (NIPS)*, 2011.

[60] Amit Goyal, Ellen Riloff, and Hal Daumé III. Automatically producing plot unit representations for narrative text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[61] Amit Goyal, Ellen Riloff, and Hal Daumé III. A computational model for plot units. *Computational Intelligence Journal*, 2013.

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

[63] Z. Harris. Distributional structure. *Word 10 (23)*, 1954.

[64] Vasileios Hatzivassiloglou and Kathy McKeown. Predicting the semantic orientation of adjectives. In *Association for Computational Linguistics*, 1997.

[65] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, 2005.

[66] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53, 2006.

[67] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, 1998.

[68] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008.

[69] Aren Jansen and Benjamin Van Durme. Indexing raw acoustic features for scalable zero resource search. In *International Speech Communication Association (INTERSPEECH)*, 2012.

[70] A. Kalyanpur, B. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. Qiu. Structured Data and Inference in DeepQA. *IBM Journal of Research and Development*, 56(3/4), 2012.

[71] A. Kalyanpur, S. Patwardhan, B. Boguraev, A. Lally, and J. Chu-Carroll. Fact-based Question Decomposition for Candidate Answer Re-ranking. In *CIKM '11: Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011.

[72] A. Kalyanpur, S. Patwardhan, B. Boguraev, A. Lally, and J. Chu-Carroll. Fact-based Question Decomposition in DeepQA. *IBM Journal of Research and Development*, 56(3/4), 2012.

[73] Richard M. Karp, Christos H. Papadimitriou, and Scott Shenker. A simple algorithm for finding frequent elements in streams and bags. 2003.

[74] Su Nam Kim and Preslav Nakov. Large-scale noun compound interpretation using bootstrapping and the web as a corpus. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011.

[75] Brian Kjersten and Benjamin Van Durme. Space efficiencies in discourse modeling via conditional random sampling. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012.

[76] R. Kneser and H. Ney. Improved Backing-off for M-gram Language Modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995.

[77] Philipp Koehn and Hieu Hoang. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[78] Zornitsa Kozareva, Ellen Riloff, and Eduard Hovy. Semantic class learning from the web with hyponym pattern linkage graphs. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

[79] A. Lally, J. Prager, M. McCord, B. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll. Question Analysis: How Watson Reads a Clue. *IBM Journal of Research and Development*, 56(3/4), 2012.

[80] Lillian Lee. Measures of distributional similarity. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

[81] Abby Levenberg, Chris Callison-Burch, and Miles Osborne. Stream-based translation models for statistical machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[82] Abby Levenberg and Miles Osborne. Stream-based randomised language models for SMT. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[83] Ping Li and Kenneth W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics*, 33(3), 2007.

[84] Ping Li and Kenneth Ward Church. Using sketches to estimate associations. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

[85] Ping Li, Kenneth Ward Church, and Trevor Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In *Advances in Neural Information Processing Systems*, 2006.

[86] Ping Li, Kenneth Ward Church, and Trevor Hastie. One sketch for all: Theory and application of conditional random sampling. In *Advances in Neural Information Processing Systems*, 2008.

[87] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, 2006.

[88] Ping Li and Arnd Christian König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 54(8), 2011.

[89] Ping Li, Art Owen, and Cun-Hui Zhang. One permutation hashing. In *Advances in Neural Information Processing Systems*. 2012.

[90] Ping Li, Anshumali Shrivastava, Joshua L. Moore, and Arnd C. Knig. Hashing algorithms for large-scale learning. In *Advances in Neural Information Processing Systems*. 2011.

[91] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases (VLDB)*, 2007.

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

[93] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, 2002.

[94] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. ACM, 2007.

[95] Zachary J. Mason. Cormet: a computational, corpus-based conventional metaphor extraction system. *Computational Linguistics*, 30(1), 2004.

[96] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. HLT '05, 2005.

[97] Tara McIntosh and James R Curran. Weighted mutual exclusion bootstrapping for domain independent lexicon and template acquisition. In *Proceedings of the Australasian Language Technology Association Workshop 2008*, 2008.

[98] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems (TODS)*, 31(3), 2006.

[99] Mark Dredze Michael J. Paul. You Are What You Tweet: Analyzing Twitter for Public Health. In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*, 2011.

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

[101] Margaret Mitchell, Jesse Dodge, Amit Goyal, Kota Yamaguchi, Karl Stratos, Xufeng Han, Alyssa Mensch, Alexander C. Berg, Tamara L. Berg, and Hal Daumé III. Midge: Generating image descriptions from computer vision detections. In *European Chapter of the Association for Computational Linguistics (EACL)*, 2012.

[102] Saif Mohammad, Cody Dunne, and Bonnie Dorr. Generating high-coverage semantic orientation lexicons from overtly marked words and a thesaurus. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[103] Robert C. Moore and Chris Quirk. Less is more: Significance-based N-gram selection for smaller, better language models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[104] A. Moschitti, J. Chu-Carroll, S. Patwardhan, J. Fan, and G. Riccardi. Using Syntactic and Semantic Structural Kernels for Classifying Definition Questions in Jeopardy! In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

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

[106] Vinod Nair, Dhruv Mahajan, and Sundararajan Sellamanickam. A unified approach to learning task-specific bit vector representations for fast nearest neighbor search. In *Proceedings of the 21st international conference on World Wide Web (WWW)*. ACM, 2012.

[107] Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. Using universal linguistic knowledge to guide grammar induction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, 2010.

[108] Nisan. Pseudorandom generators for space-bounded computation. *COMBINAT: Combinatorica*, 12, 1992.

[109] Sandeep Pandey, Andrei Z. Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-neighbor caching for content-match applications. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, 2009.

[110] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

[111] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA)*. ACM, 2006.

[112] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[113] Patrick Pantel and Dekang Lin. Discovering word senses from text. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[114] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation.

[115] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[116] Adam Pauls and Dan Klein. Large-scale syntactic language modeling with treelets. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2012.

[117] Marco Pennacchiotti and Patrick Pantel. Entity extraction via ensemble semantics. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[118] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Using paraphrases for improving first story detection in news and twitter. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012.

[119] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[120] Delip Rao and Deepak Ravichandran. Semi-supervised polarity lexicon induction. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, 2009.

[121] Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. Randomized algorithms and NLP: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, 2005.

[122] E. Riloff and R. Jones. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In *American Association for Artificial Intelligence (AAAI)*, 1999.

[123] E. Riloff, J. Wiebe, and T. Wilson. Learning Subjective Nouns using Extraction Pattern Bootstrapping. In *Conference on Computational Natural Language Learning (CONLL)*, 2003.

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

[125] Florin Rusu and Alin Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD 2007)*, 2007.

[126] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. volume 50, 2009.

[127] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment (PVLDB)*, 5(5), 2012.

[128] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. 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, 2010.

[129] Holger Schwenk and Philipp Koehn. Large and diverse language models for statistical machine translation. In *Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNP)*, 2008.

[130] Hajime Senuma. K-means clustering with feature hashing. In *Proceedings of the ACL 2011 Student Session*. Association for Computational Linguistics, 2011.

[131] Jian Shao, Fei Wu, Chuanfei Ouyang, and Xiao Zhang. Sparse spectral hashing. *Pattern Recognition Letters*, 33(3), 2012.

[132] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *Journal Machine Learning Research (JMLR)*, 10, 2009.

[133] Ekaterina Shutova. Automatic metaphor interpretation as a paraphrasing task. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[134] Ekaterina Shutova. Models of metaphor in nlp. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[135] Ekaterina Shutova, Lin Sun, and Anna Korhonen. Metaphor identification using verb and noun clustering. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, 2010.

[136] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Richang Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011.

[137] A. Stolcke. SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing*, pages 901–904, Denver, CO, September 2002.

[138] Andreas Stolcke. Entropy-based pruning of backoff language models. In *Proceedings of DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

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

[140] Hiroya Takamura, Takashi Inui, and Manabu Okumura. Extracting semantic orientations of words using spin model. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.

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

[142] David Talbot and Thorsten Brants. Randomized language models via perfect hash functions. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

[143] David Talbot and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007.

[144] David Talbot and Miles Osborne. 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*, 2007.

[145] Partha Pratim Talukdar and Fernando Pereira. Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2010.

[146] Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. Word meaning in context: A simple and effective vector model. In *Proceedings of 5th International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing, 2011.

[147] M. Thelen and E. Riloff. A Bootstrapping Method for Learning Semantic Lexicons Using Extraction Pattern Contexts. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

[148] Ferhan Ture, Tamer Elsayed, and Jimmy Lin. No free lunch: brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 2011.

[149] Peter Turney, Yair Neuman, Dan Assaf, and Yohai Cohen. Literal and metaphorical sense identification through concrete and abstract context. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011.

[150] Peter D. Turney. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, 2008.

[151] Peter D. Turney and Michael L. Littman. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Transactions on Information Systems (TOIS)*, 21, 2003.

[152] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *journal of artificial intelligence research (JAIR)*, 37, 2010.

[153] Tanguy Urvoy, Emmanuel Chauveau, Pascal Filoche, and Thomas Lavergne. Tracking web spam with HTML style similarities. *ACM Transactions on the Web (TWEB)*, 2(1), 2008.

[154] Jakob Uszkoreit and Thorsten Brants. Distributed word clustering for large scale class-based language modeling in machine translation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

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

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

[157] Benjamin Van Durme and Ashwin Lall. Online generation of locality sensitive hash signatures. In *Proceedings of the ACL 2010 Conference Short Papers*, 2010.

[158] Benjamin Van Durme and Ashwin Lall. Efficient online locality sensitive hashing via reservoir counting. In *Proceedings of the ACL 2011 Conference Short Papers*, 2011.

[159] Leonid Velikovich, Sasha Blair-Goldensohn, Kerry Hannan, and Ryan McDonald. The viability of web-derived polarity lexicons. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[160] Vishnu Vyas, Patrick Pantel, and Eric Crestan. Helping editors choose better seed sets for entity expansion. In *Proceedings of ACM Conference on Information and Knowledge Management (CIKM-09)*, 2009.

[161] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09. ACM, 2009.

[162] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *12th European Conference on Computer Vision (ECCV)*, 2012.

[163] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[164] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005a.

[165] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2001.

[166] Ying Zhang, Almut Silja Hildebrand, and Stephan Vogel. Distributed language modeling for n-best list re-ranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, 2006.

[167] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. In *Technical Report CMU-CALD-02-107*. Carnegie Mellon University, 2002.