Analysis on Complementary Count Min Sketch

Neeraj Sharma, Kanchi Masalia
Department of Computer Science Engineering
University of Massachusetts Amherst
{`neerajsharma, kmasalia`}`@umass.edu`

Abstract

Count Sketch and Count Min Sketch are commonly used for frequency estimation in a streaming setting. This data structures can also be used for approximating data by accumulating it. One interesting application of this data structure is observed in field of Machine Learning for feature selection. In this paper, we present analysis of our proposed data structure called Complementary Count Min Sketch, which is aimed to use less space and running time without much loss in accuracy compared to count sketch. We analyzed it for feature selection as well as frequency estimation wherein the elements can be removed as well from the stream. We analyzed the performance over RCV1 - a high dimensional dataset as well as a synthetic dataset which gave us flexibility to define dimensions of the dataset, nature of dataset and tweak it's sparsity.

1 Introduction

There has been an immense rise in generated data which has led to the era of Big Data. These have heavily influence how we think, build, and maintain applications. For streams with data arriving at high rate, algorithms are needed which use as little processing time and space in order to analyze and provide query response in real time. Count Min Sketch and Count Sketch data structures are used in such scenarios for frequency estimation and finding most frequently occurring items of the stream.

In machine learning and statistics, feature selection is a process in which one chooses features which contribute most to the prediction variable or output. At times feature selection is confused with dimensionality reduction. It is true that both of these help in reducing the features in a dataset, but the difference lies in how they approach this problem. Dimensionality reduction reduces the number of features by creating new features as combinations of existing ones. So all the features are still present in a way, but the total number of features is reduced. But in feature selection, we either retain a feature or remove it completely from the dataset. When data is present in low dimensions, there are many different algorithms available for feature selection. However, when data is present in high dimensions, the training time for the model increases with the dimensions exponentially. Feature selection of high dimensional data requires large amount of memory and time. Usually high dimensional vectors are sparse i.e very few features actually have non-zero values. It is not difficult to load such sparse high dimensional data in memory as we can ignore the zero elements and use data structure like dictionary to have the position of feature and its value. The problem arises with storing and performing operations on dense high dimensional vector. We propose a structures for feature selection of sparse high dimensional vectors using Complementary Count Min Sketches

(CCMS) along with maintaining heap of the most important features to preserve the interpretability of features. Previously [1] have used Count Sketch (CS) for feature selection. In this work, we are analyzing the complementary count min sketch.

2 Related Work

Count Sketch was introduced by [2] to find the most frequently occurring item in the data stream. Count Min Sketch was introduced by [3] to find the approximate count of items occurring in the stream of data. Both the data structures have a similar design. However, they have different error guarantees. Count Sketch uses pairwise independent hash functions and sign hash functions for hashing the features into the sketch (Array like data structure). Count Min Sketch uses only hash functions. So the total number of hash functions computed while adding an element to count sketch is twice as compared when the element is added to count min sketch. The estimate of frequency of element provided by count min sketch is an upper bound of the actual frequency of the element whereas the estimate of frequency provided by count sketch could be lower or higher than the actual occurrence of the element. The error estimate for Count Min Sketch is L1 norm of the frequency vector (approximation sketch) and for Count Median Sketch is L2 norm of frequency vector. By Cauchy-Schwarz inequality we know that L1 norm is bounded by $\sqrt{n} * \ L2norm$ where n is the length of the frequency vector (approximation sketch). However, Count-Min sketch algorithm gives better average error than the Count Sketches when using constant space. Comparative analysis of count sketch vs count min sketch motivated us to go ahead and propose variants of sketch for frequency estimation and its other applications.

We found an interesting application of this sketch for feature selection technique wherein the sketch is used for compression of feature weights.

Feature selection for high dimensional data is very important as the training time increases exponentially with the dimensions. Due to curse of dimensionality, high dimensional data can easily overfit regression model and thus requires careful hyperparameter tuning. One of the solution to this problem is feature hashing [4] which makes working with high dimensional data computationally feasible, but at the cost of losing the interpretability of features. Consider a 3-gram string "abc". With feature hashing, one uses a lossy, random hash function $h : strings \rightarrow \{1, 2, ...R\}$ to map "abc" to a feature number $h(abc)$ in the range $\{1, 2, ...R\}$. This is extremely convenient because it enables one to avoid creating a large look-up dictionary. Furthermore, this serves as a dimensionality reduction technique. Unfortunately, this convenience comes at a cost, we lose the identity of the original features. This is not a viable option if one cares about both feature selection and interpretability.

Another popular approach by [5] is to use greedy thresholding methods combined with stochastic gradient descent to prevent the feature vector from becoming too dense and blowing up in memory. In these methods, the intermediate iterates are regularized at each step, and a full gradient update is never stored nor computed (since this is memory and computation intensive). However, it is well known that greedy thresholding can be myopic and can result in poor convergence.

[6] have introduced a new sub-linear space sketch: the Weight-Median Sketch. This is for learning compressed linear classifiers over data streams while supporting the efficient recovery of large-magnitude weights in the model. This enables memory-limited execution of several statistical analyses over streams, including online feature selection, streaming data explanation.

[2] has developed a data structure to capture the features that are most discriminative of one stream (or class). The Weight-Median Sketch is built on top of the data structure Count-Sketch, but, instead of sketching counts, it captures sketched gradient updates to the model parameters. The core idea for performing feature selection is figuring out the most discriminative features from the set of features. This memory efficient data structure can be used to accumulate the gradients of the high dimensional feature vector when the model is learning without much loss in approximation. This can be viewed as dimensionality reduction via random projection. The issue with this approach is

we loose the interpretability of the features. Once we accumulate the gradients in count sketch for high dimensional data, it is not possible to decipher the discriminating feature as different features would hash at same index in the sketch and will loose the interpretability.

[1] have implemented a method to maintain the interpretability of the features using the Weight-Median sketch and maintaining a heap of most discriminating features. This method accurately and efficiently performs feature selection on real-world, large-scale datasets with billions of dimensions.

We will be leveraging this methodology to maintain the interpretability of features. We propose using our variants of sketch to maintain weights which would use less space compared to Weight median sketch built on count sketch.

3 Methodology

Here we are presenting the methods and algorithms which we used for our study.

3.1 Zipfian Distribution and Power Law Distribution

Many real data distributions such as sizes of cities, word frequencies, citations of papers, web page access frequencies, and file transfer size and duration are often characterized by the Zipfian, Pareto, or Power-law distributions which only differ by the choice of parameters. The zipfian distribution with parameter $\alpha > 0$ is a discrete distribution stating that the k^{th} largest frequency f_k has a frequency proportional to $k^{-\alpha}$. We can see, $\alpha = 0$ generates a uniform distribution whereas the larger α the more skewed the distribution gets. Zipfian law is cumulative form of Power law distribution. Zipfian and Power law distributions are especially interesting with regards to the heavy hitters problem since this problem looks for frequencies which are significantly larger than the rest of the data. For increasingly skewed zipfian distributions the elements with such frequencies become more frequent, since only a few of the overall frequencies account for more of the total frequency.

3.2 Count Min Sketch

The Count Min Sketch (CMS) is a randomized method closely related to bloom filters. The count min sketch data structure can only be used when Δ is positive. Count Min Sketch has d random pairwise independent hash functions $h_j j \in \{1, 2, ..., d\}$ to map the vector's components to bins w. $h_j : \{1, 2, ..., p\} \rightarrow \{1, 2, 3, ..., w\}$. Every component i is hashed into bin $S(j, h_j(i))$. The count-min-sketch supports two operations: **UPDATE**(item i, increment Δ). The update operation updates the sketch with any observed increment. For an increment Δ to an item i, the sketch is updated by adding Δ to the cell $S(j, h_j(i)) \forall j \in \{1, 2, ..., d\}$. The **QUERY** operation returns the estimate for component i, the min of all the d different associated counters.

Algorithm 1: Count Min Sketch

v universal hash functions h_j
Initialize count-min-sketch matrix $S \in R^{v,w} = 0$

Update(item i, increment: Δ)
Update component i with update Δ
$S(j, h_j(i)) = S(j, h_j(i)) + \Delta \ \ \forall j \in \{1...d\}$

Query (item i):
Query Sketch for an estimate for item i
return Min($S_{j,h_j(i)}$)

for any item i, when we query its value from Count Min Sketch, it will always give an overestimate of its value as other item also can map to the same position in count min sketch.

$$S_j[h_j(i)] = f_i(x) + \sum_{y \neq i : h_j(y) = h_j(i)} f(y)$$

Where $f_i(x)$ is actual value of item i and $\sum_{y \neq i : h_j(y) = h_j(i)} f(y)$ is error in estimating value for item i . Expected Error:

$$E[\sum_{y \neq i : h_j(y) = h_j(i)} f(y)] = \sum_{y \neq i} Pr(h_j(i) = h_j(y)) f(y)$$

$$\implies \sum_{y \neq i} \frac{1}{w} f(y) \leq \frac{N}{w}$$

where N is all items in stream. Using Markov inequality we can say:

$$f(i) \leq S_j[h_j(i)] \leq f(i) + \frac{\epsilon * N}{K}$$

where K is Heavy Hitters Count and ϵ is error rate. So in this way we can say that count min sketch can be used to approximate frequency of every item i in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(\frac{1}{\delta})\frac{k}{\epsilon})$ space and time $O(\log \delta^{-1})$ where δ is failure probability.

For frequency estimation of heavy hitters under stream settings, we can use $\Delta = 1$ in algorithm 1.

3.3 Count Median Sketch (Count Sketch)

The algorithm uses d random hash functions similar to count-min sketch but it also uses d random sign functions as well to map the components of vectors randomly to $\{+1, -1\}$ i.e. $s_i : \{1, 2, ..., n\} \rightarrow \{+1, -1\}$. The count-sketch (CS) also similarly supports two operations, **UPDATE**(item i, update δ) and **QUERY**(item i). The **UPDATE** operation updates the sketch with any observed update. It may be increment as well as decrement. For an update Δ to an item i, the sketch is updated by adding $s_j(i)\Delta$ to the cell $S(j, h_j(i)) \forall j \in \{1, 2, ..., d\}$. The **QUERY** operation returns an estimate for component i, the median of all the d different associated counters.

Algorithm 2: Count Median Sketch

v universal hash functions h_j
v random sign functions s_j
Initialize count-sketch tensor $S \in R^{v,w} = 0$

Update(item i, update: Δ)
Update component i with update Δ
$S(j, h_j(i)) = S(j, h_j(i))$+s$_j(i)\Delta \;\; \forall j \in \{1...d\}$

Query(item i):
Query Sketch for an estimate for item i
return Median($S_{j,h_j(i)} s_j(i)$))

Let us analyze the Count Median Sketch: Let us assume the possible different items coming into the stream will be M. Let us define an indicator function

$$Y_j = \begin{cases} 1, & \text{if } h(j) = h(j^*) \\ 0, & \text{otherwise} \end{cases}$$

Approximate value of item j:

$$\hat{f}_{j^*} = s(j) * S[h(j)]$$

$$\hat{f}_{j^*} = s(j^*) \sum_{j=1}^{M} f_j s(j) Y_j$$

$$\implies s(j^*)^2 f_j y_j^* + \sum_{j \neq j^*} f_j s(j^*) s(j) Y_j$$

$$\implies f_j + \sum_{j \neq j^*} f_j s(j^*) s(j) Y_j$$

Since $s(j*)^2 = 1$ and $Y_{j^*} = 1$, By Linearity of Expectation:

$$E(\hat{f}_{j^*}) = f_{j^*} + \sum_{j \neq j^*} f_j \, E[s(j^*) s(j) Y_j]$$

Since s is pairwise independent and is independent of Y_j which is solely function of h, for $j \neq j^*$, we have:

$$E[s(j^*) s(j) Y_j] = 0$$

$$\implies E[\hat{f}_{j^*}] = f_{j^*}$$

As $\hat{f}_{j^*}$ is an unbiased estimator of f_{j^*}, let us compute its variance:

$$Var(\hat{f}_{j^*}) = E[\hat{f}_{j^*} - f_{j^*}]^2$$

$$\implies E[\sum_{i \neq j^*} \sum_{j \neq j^*} f_i f_j s(i) s(j) Y_i Y_j]$$

Since $s(j*)^2 = 1$

$$\implies \sum_{i \neq j^*} \sum_{j \neq j^*} f_i f_j \, E[s(i) s(j) Y_i Y_j]$$

As for $i \neq j$, since s is pairwise independent and independent of h, $E[s(i)s(j)Y_iY_j] = 0$. Therefore the only terms in the variance that are survive are when i = j.

$$\implies Var(\hat{f}_{j^*}) = \sum_{j \neq j^*} f_j^2 \, E[Y_j^2]$$

As $E[Y_j^2] = E[Y^j]$ and Y_j is indicator function so $E[Y_j] = Pr(h(j) = h(j^*)) = \frac{1}{w}$. Therefore,

$$Var(\hat{f}_{j^*}) = \sum_{j \neq j^*} \frac{f_j^2}{w} = \frac{||f||_2^2 - f_{j^*}^2}{w} = \frac{||f_{-j^*}||^2}{w}$$

By Chebyshev Inequality:

$$Pr(||\hat{f}_{j^*} - f_{j^*}|| \geq \epsilon ||f_{-j^*}||) \leq \frac{Var(\hat{f}_j^*)}{\epsilon^2 ||f_{-j^*}||_2^2}$$

The query has a running time proportional to the depth of the sketch. As for the update procedure of Count-Median Sketch, two invocations of a hash function, a multiplication, and an addition is required at each row. All these operations are constant and add up to $O(d)$ time. The median of the d estimates must be found, which can be done in linear $O(d)$ time, yielding a total of $O(d) = O(\ln \delta^{-1})$ running time for the point query overall.

3.4 Comparison of Count Min and Count Sketch

The Count-Median Sketch provides a better guarantee, since it guarantees that $\hat{v}_i$ is within an additive factor of $\epsilon||v||_2$ of the true frequency vi with probability $1 - \delta$. This guarantee is significantly stronger in most cases as $||v||_2 \leq ||v||_1$. The cost of this guarantee is significantly larger, since the Count-Median Sketch requires $O(\log(\frac{1}{\delta})\frac{k}{\epsilon^2})$ in comparison to count min sketch which takes $O(\log(\frac{1}{\delta})\frac{k}{\epsilon})$ space to support updates and queries in the same time as Count-Min Sketch.

If we compare the precision of the two sketches, it is observed that for data with a near uniform distribution, the Count-Median Sketch provides smaller errors, whereas when the data becomes more and more skewed, the Count-Min Sketch provides the smallest error. This is not surprising since the Count-Median Sketch still has a relationship with the L2-norm, which increases when the data becomes more skewed implying that the error increases as well.

Under equal space constraint, The precision of the Count-Median Sketch is also expected to change drastically, due to the decrease in space usage. In fact it is expected that the decrease in space implies that the error guarantee now is bounded according to the L1-norm instead of the L2-norm according to [7]. So Count-Median Sketch can in fact be shown to provide an error guarantee according to the L1-norm, by changing the width to be equal to the width of a Count-Min Sketch.

Summing it up, we can say that the Count-Min Sketch and the Count-Median Sketch with width $w = O(\epsilon^{-1})$ is indeed comparable, and that comparing the sketches according to space, precision and running times gives very similar results, where different data distributions determines which sketch performs the best. The only notable difference is in the running time of the query algorithms where the Count-Min Sketch in general seems to be faster than the Count-Median Sketch. It is likely due to calculating the minimum compared to calculating the median is faster and easier operation. This is where the motivation came for us to come up with a data structure inspired from count min sketch for accumulation of gradients.

3.5 Complementary Count Min Sketch

Count min sketch doesn't support reducing frequency or removing an item in a stream of updates. As if we decrement the frequency the error bound will not hold which states that the frequency estimated by count min would be equal or would be an overestimate of the true frequency. However, Count Sketch supports reducing frequency or removing an item and also holds the error bounds as it gives the estimate as an median. As we discussed in 3.4 that count min sketch is faster in comparison to count sketch and with equal space bound count sketch and count min sketch both provides error within L1-norm bound. To support negative updates in count min sketch we proposed this novel data structure - Complementary Count Min Sketch (CCMS). In Complementary (positive and negative) Count Min Sketch, we will have two sketches S^{pos} and S^{neg} which will accumulate the positive and negative updates respectively. So for **UPDATE**(item i, update Δ), we will first check if Δ is positive or negative. If Δ is positive then will update it in S^{pos}, the same way we did in Count Min Sketch's update method. If Δ is negative then we will update the absolute value of Δ in S^{neg}. For **QUERY**(item i), we will query the item in S^{pos} and S^{neg} and will return the $min_j S^{pos}[j, h_j(i)] - min_j S^{neg}[j, h_j(i)]$.

Let us do the Mathematical Analysis of Complementary Count Min Sketch: From Count Min Sketch analysis we know: For Single Count Min Sketch

$$S_j[h_j(i)] = f_i(x) + \sum_{y \neq i: h_j(y) = h_j(i)} f(y)$$

Where $f_i(x)$ is actual value of item i and $\sum_{y \neq i: h_j(y) = h_j(i)} f(y)$ is error in estimating value for item i. Similarly:

$$S_j^{pos}[h_j(i)] = f_i(x) + \sum_{y \neq i: h_j(y) = h_j(i)} f(y)$$

Algorithm 3: Complementary Count Min Sketch

v universal hash functions h_j

Initialize positive count-min-sketch matrix $S^{pos} \in R^{v,w} = 0$, negative count-min-sketch matrix $S^{neg} \in R^{v,w} = 0$

Update (item i, update: Δ)

Update component i with update Δ

if $\Delta > 0$ **then**

| $S^{pos}(j, h_j(i)) = S^{pos}(j, h_j(i)) + \Delta \ \ \forall j \in \{1...d\}$

else

| $S^{neg}(j, h_j(i)) = S^{neg}(j, h_j(i)) + abs(\Delta) \ \ \forall j \in \{1...d\}$

end

Query (item i):

Query Sketch for an estimate for item i

return Min($S^{pos}_{j,h_j(i)}$) - Min($S^{neg}_{j,h_j(i)}$)

Where $f_i(x)$ is actual value of item i and $\sum_{y \neq i : h_j(y) = h_j(i)} f(y)$ is error in estimating value for item i.

$$S_j^{neg}[h_j(i)] = f_i(x) + \sum_{y \neq i : h_j(y) = h_j(i)} f(y)$$

Where $f_i(x)$ is actual value of item i and $\sum_{y \neq i : h_j(y) = h_j(i)} f(y)$ is error in estimating value for item i from negative items of stream.

For Complementary Count Min Sketch:

$$E[g(i)] = E[f^{pos}(i) + \sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y)] - E[f^{neg}(i) + \sum_{y' \neq i : h_j(y') = h_j(i)} f^{neg}(y')]$$

Here y and y' are hash functions for positive and negative sketch. $g(x)$ is true estimate of particular item.

$$\implies E[f^{pos}(i) - f^{neg}(i)] - E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y) - \sum_{y' \neq i : h_j(y') = h_j(i)} f^{neg}(y')]$$

As

$$\implies E[f^{pos}(i) - f^{neg}(i)] = E[f(i)] = f(i)$$

So Error from CCMS:

$$Error = E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y) - \sum_{y' \neq i : h_j(y') = h_j(i)} f^{neg}(y')]$$

$$Error = I(y = y').E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y) - f^{neg}(y)] + I(y \neq y').E[\sum_{y' \neq y \neq i : h_j(y') = h_j(i)} f^{pos}(y) - f^{neg}(y')]$$

$$\implies \frac{1}{m^2} E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y) - f^{neg}(y)] + \frac{1}{m(m-1)} E[\sum_{y' \neq y \neq i : h_j(y') = h_j(i)} f^{pos}(y) - f^{neg}(y')]$$

So upper bound for the error will be:

$$\implies UpperBound = \frac{1}{m^2}E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{pos}(y)] + \frac{1}{m(m-1)}E[\sum_{y \neq y' \neq i : h_j(y) = h_j(i)} f^{pos}(y')]$$

So lower bound for the error will be:

$$\implies LowerBound = \frac{1}{m^2}E[\sum_{y \neq i : h_j(y) = h_j(i)} f^{neg}(y)] + \frac{1}{m(m-1)}E[\sum_{y \neq y' \neq i : h_j(y) = h_j(i)} f^{neg}(y')]$$

We observe the error is bounded between error from negative count min sketch to positive count min sketch. Thus the estimated frequency can be lower as well as greater then true frequency. However, If there are proportionate amount of negative and positive updates these error terms will cancel out each other and would be close to the true estimate.

3.5.1 Variants of Complementary Count Min Sketch

Complementary Count Min Sketch with Different Hash Functions:
In the previously proposed complementary count min sketch, we used same hash functions for positive as well as negative count min sketch meaning positive and negative updates will have same positions in sketches. Thus, if hash of two particular items are colliding with each other then they will collide in both the sketches. We tried to analyze if we have different hash functions for positive and negative count min sketches, would it help us to minimize the error. Algorithm is described here 4.

Algorithm 4: Complementary Count Min Sketch using different Hash Functions

---

v universal hash functions h_j^{pos}, h_j^{neg} for positive and negative count min sketches
Initialize positive count-min-sketch matrix $S^{pos} \in R^{v,w} = 0$
Initialize negative count-min-sketch matrix $S^{neg} \in R^{v,w} = 0$

Update (item i, update: Δ)
Update component i with update Δ
if $\Delta > 0$ **then**
  $S^{pos}(j, h_j^{pos}(i)) = S^{pos}(j, h_j^{pos}(i)) + \Delta \ \ \forall j \in \{1...d\}$
else
  $S^{neg}(j, h_j^{neg}(i)) = S^{neg}(j, h_j^{neg}(i)) + abs(\Delta) \ \ \forall j \in \{1...d\}$
end

Query (item i):
Query Sketch for an estimate for item i
return Min($S^{pos}_{j,h_j^{pos}(i)}$) - Min($S^{neg}_{j,h_j^{neg}(i)}$)

Complementary Count Min Sketch with Single Hash Function:
We realized that the previous approach is quite similar to having a single count min sketch where in we use different hash functions for positive and negative updates. This is synonymous to using one count min sketch where we update the positive one as usual and for the negative we get the hash of negative number and update at that position. Algorithm is described here 5.

Conservative Complementary Count Min Sketch:
As we studied in our last semester, count min sketch with conservative updates gives more accurate

Algorithm 5: Complementary Count Min Sketch using Single Count Min Sketch

v universal hash functions h_j
Initialize count-min-sketch matrix $S \in R^{v,w} = 0$

Update (item i, update: Δ)
Update component i with update Δ
if $\Delta > 0$ **then**
| $S(j, h_j(i)) = S(j, h_j(i)) + \Delta \ \ \forall j \in \{1...d\}$
else
| $S(j, h_j(-i)) = S(j, h_j(-i)) + abs(\Delta) \ \ \forall j \in \{1...d\}$
end

Query (item i):
Query Sketch for an estimate for item i
return Min($S(j, h_j(i))$) - Min($Sj, h_j(-i)$) $\forall j \in \{1...d\}$

and lower bounds on error. In standard count min sketch, the update operation updates the sketch with any observed increment. For an increment Δ to an item i, the sketch is updated by adding Δ to the cell $S(j, h_j(i)) \forall j \in \{1, 2, ..., d\}$. In this variant, instead of incrementing each counter, we first compute $M = min_{j \in \{1,..d\}} S(j, h_j(i))$. Then we only increment $S(j, h_j(i))$ if $S(j, h_j(i)) = M$. We follow this technique for both positive and negative count min sketch. Query method remains same to the standard count min sketch method. Algorithm is described here 6.

Algorithm 6: Conservative Complementary Count Min Sketch

v universal hash functions h_j
Initialize positive count-min-sketch matrix $S^{pos} \in R^{v,w} = 0$
Initialize negative count-min-sketch matrix $S^{neg} \in R^{v,w} = 0$

Update (item i, update: Δ)
Update component i with update Δ
Compute $M = min_{j \in \{1,..d\}} S(j, h_j(i))$
if $\Delta > 0$ **then**
| $S^{pos}(j, h_j(i)) = S^{pos}(j, h_j(i)) + \Delta \ \ \forall j \in \{1...d\} \ s.t \ \ S(j, h_j(i)) = M$
else
| $S^{neg}(j, h_j(i)) = S^{neg}(j, h_j(i)) + abs(\Delta) \ \ \forall j \in \{1...d\} \ s.t. \ \ S(j, h_j(i)) = M$
end

Query (item i):
Query Sketch for an estimate for item i
return Min($S^{pos}_{j,h_j(i)}$) - Min($S^{neg}_{j,h_j(i)}$)

3.6 Feature selection techniques

3.6.1 Feature selection Using Sketches

Consider the feature selection problem in the high dimensional setting where we are given a dataset (X_i, y_i) for $i \in [n]$. Each data point $X_i \in R^p$ and label $y_i \in R$. We are interested in finding the k-sparse feature vector $\beta \in R^p$ from below optimization problem which solves our feature selection task where k non zero elements are the selected features.

$$min_{||\beta||_0 = k} ||y - X\beta||_2$$

Where $X = [X_1; X_2; :::; X_n]$ and $y = [y_1; y_2; :::; y_n]$ denote the data matrix and label vector and l_o norm $||\beta||_0$ denotes the number of non zero entries in $||\beta||_0$.

We are interested in solving the feature selection problem for high-dimensional datasets where the number of features p is so large that a dense vector (or matrix) of size p cannot be stored explicitly in memory. Sketch data structures allow us to accumulate the gradients updates over several iterations because of linear aggregation. We will be using the same algorithm as described in [1]. Our novel contribution is to use above described sketch variants instead of Count Sketch. First, we initialize the Sketch S and the feature vector $\beta^{t=0}$ with zeros entries. The sketch hashes a p-dimensional vector into $O(\log^2 p)$ buckets as fig(1). At iteration t, this algorithm selects a random row X_i from the data matrix X and computes the stochastic gradient update term using the learning rate λ. For logistic regression, the gradient of softmax function can be defined as $g_i = \lambda * (y_i - X_i\beta^t)X_i$. As the data vector X_i and the corresponding stochastic gradient term are sparse, we only add the non-zero entries of the stochastic gradient term $\{g_{ij} : \forall j \; g_{ij} > 0\}$ to the Count-Sketch S. After adding the non-zero entry to the sketch, we perform query operation for those items and insert it into the heap structure - top k if the absolute value from query operation output is greater than the minimum absolute value in heap along with feature position. In gist, we are maintaining the most discriminating features (ones with high absolute weights) in Top-K. This structure is solely responsible for maintaining the interpretability of the features. After performing this stochastic gradient update step for all the examples and performing the described operation. Finally, we query the Top-K values of the sketch as the final output. This is detailed in Algorithm 7

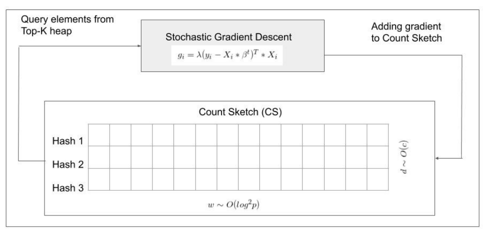

Figure 1: Schematic of Algorithm

Algorithm 7: Feature Selection: Using Sketch

Result: The top-k heavy-hitters from the Sketch
Initialize: $\beta^0 = 0,\ \boldsymbol{S}\ (Sketch),\ \lambda(learning\ rate)$
while *not stopping criteria* **do**
  Find the gradient update $g_i = \lambda(y_i - X_i\beta^t)^T X_i$
  Add the gradient update to the sketch $g_i \rightarrow \boldsymbol{S}$
  Get the top-k heavy hitters from the sketch $\beta^{t+1} \leftarrow \boldsymbol{S}$
end

3.6.2 Greedy Thresholding

In the feature selection algorithms, the class of hard thresholding algorithms have the smallest memory footprint. Hard thresholding algorithms retain only the top-k values and indices of the entire feature vector using $O(k \log(p))$ memory. An algorithm where, after each gradient update, a hard threshold is applied to the features. Only the top-k features are kept active, while the rest are set to zero. This algorithm generates the following iterates for the i^{th} variable in an stochastic gradient descent (SGD) framework.

$$\beta^{t+1} \leftarrow H_k(\beta^t - 2\lambda(y_i - X_i\beta^t)^T X_i)$$

The sparsity of the feature vector B^t, enforced by the hard thresholding operator H^k, alleviates the need to store a vector of size $O(p)$ in the memory in order to keep track of the changes of the features over the iterates. As in this algorithm, we only retains the top-k elements of β, the hard thresholding procedure greedily discards the information of the non top-k coordinates from the previous iteration. In particular, it clips off coordinates that might add to the support set in later iterations. This drastically affects the performance of hard thresholding algorithms, especially in real-world scenarios where by the design matrix X is not random, normalized, or well-conditioned. In this scenario, the gradient terms corresponding to the true support typically arrive in lagging order and are prematurely clipped in early iterations by H^k.

3.6.3 Logistic Regression with Heap

In this approach we maintain a heap similar to the the heap as we maintained for feature selection using sketch technique. After each iteration of SGD, we update the heap in order to maintain the top-k features with high magnitude where k is user defined. This ensures that after each iteration, we just have weights for k features.

3.7 Top K Recovery

Our aim for feature selection of high dimensional data set is to find most discriminating features i.e features which have most impact on the true label. In order to do this, we maintain a heap kind of data structure which maintains fixed size of the high magnitude feature weights and their indexes. This structure is updated after each update on count sketch i.e when the weight of any feature is modified. At the end of training, the Top-K heap is used to recover the K-sparse weight vector. The key idea of recovery lies in that a suitably high dimensional sparse signal can be inferred from very few linear observations.

4 Dataset

We have used two datasets in our experiment.

4.1 RCV1

Reuters Corpus Volume I (RCV1) contains over 800,000 manually categorized newswire stories. This dataset contains the Non-zero values cosine-normalized, log TF-IDF values for each document. All the features are real and between 0 and 1. RCV1 [8] dataset is categorized into 4 groups to capture the major subjects of a story. The 4 groups are Economics(ECAT), Corporate/Industrial(CCAT), Government/Social(GCAT) and Markets(MCAT). This data was processed and converted into binary classes where in positive class includes CCAT, ECAT and negative class includes GCAT, MCAT. [1]

The statistics of these datasets are summarized in table 1.

Dataset	Dimension	Train Size	Test Size
RCV1	47236	20242	677399

Table 1: Dataset Statistics

As the data set has large number of features, data is represented in dictionary format where the key is the position of feature and value is the feature value.

For simplicity purpose we chose to implement binary classification using logistic loss. Our proposed methodologies can be easily extended for multiclass classification.

[1] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.htmlrcv1.binary

4.2 Synthetic Dataset

To analyze the performance of Complementary Count Min Sketch for feature selection and heavy hitters, the results of RCV1 dataset were not easy to interpret due its high dimensions and lack of true model parameters. In order to mitigate this, we came up with the idea of generating a synthetic dataset where we can play with the number of examples, feature dimensions and sparsity of features. We also generated the true feature weights and calculated the true labels based on it. Knowing true model parameters was a major advantage of working with synthetic dataset. This helped us to analyze the behaviour of sketch and top-k heap at any stage easily. We created dataset using Gaussian distribution as well as with power law distribution with by inducing sparsity as per choice.

4.2.1 Synthetic Dataset using Gaussian Distribution

As we saw in section 3.4 that under uniform distribution count sketch has smaller error in estimation of frequency in comparison to count min sketch. We wanted to verify this for our data structure. We created the dataset matrix using standard Gaussian distribution. We randomly selected the important features and generated random weights and based on these data points and feature weight assigned labels to each example. We experimented with different dataset sparsity level as well as feature sparsity levels.

4.2.2 Synthetic Dataset using Power Law Distribution

As we saw in section 3.4 for skewed distributions like Power Law and Zipfian, count min sketch should provide smallest error. To understand the performance of our novel data structure we generated the synthetic data examples under power law distribution. We used multiple values of $\alpha = 2, 3$ and selected minimum value of k=1 and maximum value of k= number of features. We experimented with different dataset sparsity levels and feature sparsity levels to understand the performance of our data structure in comparison to other sketch variants for heavy hitters as well as feature selection task.

5 Experiments and Results

5.1 Analyzing Gradient Updates

Complementary Count Min Sketch will give least error when the error in positive and negative count min sketch negate each other. Since our experiments with RCV1 and KDD dataset performed well with CCMS data structure, we believed this was due to negation of the sketch errors. So we decided to analyze the gradient updates of each dimension of the feature vector. We performed comparative analysis of gradient updates for different approaches of feature selection using all the variants (including CCMS, conservative CCMS, CS, logistic regression, and standard logistic regression with top-k) to understand how gradient updates are being store and how top-k is changing over time.

In figure 5 we present few samples for gradient updates. Analyzing these sample gradient updates, we observe that feature 13 and feature 29 are behaving completely opposite. Where in feature 13, number of gradient updates of count sketch are lesser in comparison to standard logistic regression method, in feature 29 number of updates are more for count sketch. If we analyze feature 25, gradients fluctuations have higher magnitudes for count sketches in comparison to logistic regression updates but number of updates in logistic regression are way highers then number of updates in count sketch. To sum up, we can say we didn't find any specific pattern in gradient updates to conclude anything.

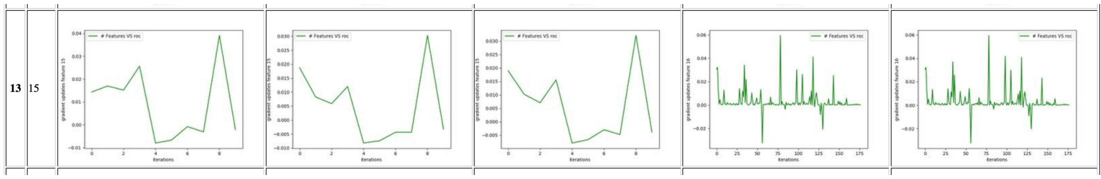

Figure 2: Gradient updates for feature 13

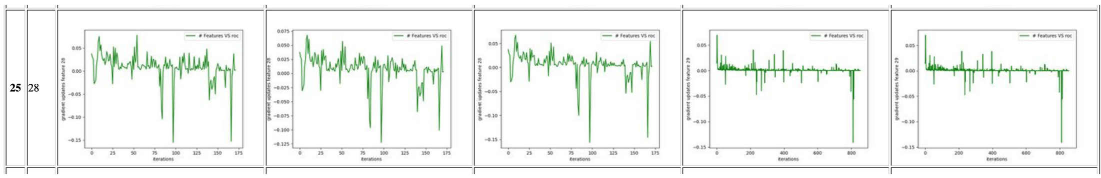

Figure 3: Gradient updates for feature 25

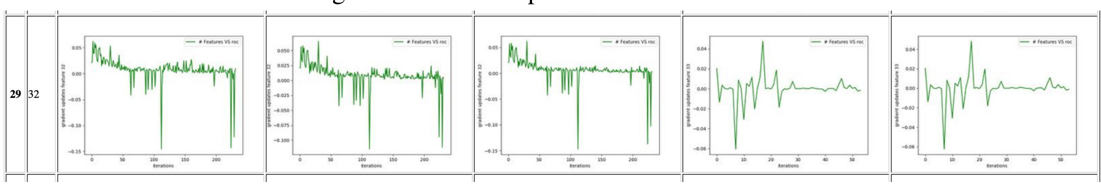

Figure 4: Gradient updates for feature 29

Figure 5: In these gradient updates, first column belongs to CCMS, second column belongs to complementary CCMS, third column belongs to CS, forth column belong to logistic regression and fifth belongs to logistic regression with heap

5.2 Performance of Datasets for feature selection

We performed experiments using different feature selection techniques for both RCV1 dataset and synthetic dataset.

RCV1 Dataset follows power law. The results of RCV1 dataset with top K = 8000 are summarised in the table below:

	Num Hash Fun.	Sketch Size	Total Space	Total Time (Sec.)	Accuracy
Logistic Regression	1	47236	47236		97.65%
Logistic Regression with Heap	1	47236	47236		97.64%
CS	3	19000	57000	45.53	95.33%
CCMS	2	14000	56000	34.42	95.36%
Conservative CCMS	2	14000	56000	44.84	94.98%
Greedy Thresholding	1	47236	47236		81.94%

Table 2: Results for RCV1 Dataset

We have approximately allocated equal space to all the sketches. We observe that accuracy of logistic regression outperforms all of them. All the sketches nearly give same accuracy. The time taken by CCMS is relatively less compared to the other sketches which is as expected.

	Num Hash Func	Sketch Size	Total Space	Accuracy
Logistic Regression	1	10000	10000	91.32%
Logistic Regression with Heap	1	10000	10000	92.45%
CS	3	2000	6000	78.55%
CCMS	2	1500	6000	74.77%
Conservative CCMS	2	1500	6000	74.41%
Greedy Thresholding	1	10000	10000	74.42%

Table 3: Results for Gaussian Synthetic Dataset

Above are the summarised results for synthetic Gaussian dataset with top K = 100. As expected, the standard logistic regression outperforms as we do not compress any gradients. We observed all other approaches nearly perform similarly. We haven't stated number of most important features recovered as all the methods perform poorly when we consider a Gaussian dataset.

	Num Hash Func	Sketch Size	Total Space	Accuracy	Positions Recovered
Logistic Regression	1	10000	10000	87.5%	80
Logistic Regression with Heap	1	10000	10000	86.35%	80
CS	3	2000	6000	87.80%	80
CCMS	2	1500	6000	87%	80
Conservative CCMS	2	1500	6000	88.67%	80
Greedy Thresholding	1	10000	10000	54.74%	80

Table 4: Results for Power Law Synthetic Dataset

From the summarised results for synthetic power law following dataset with top K = 100. We observed all approaches nearly perform similarly except greedy thresholding. All approaches are able to require the true important features.

5.3 Comparing Top-K Heap for various sketches

For RCV1 dataset, we were not aware of true parameters of the model and were clueless what should be right baseline to compare with. So we decided to compare the most important features obtained from each technique. Here we observed something very strange, the match in most important features varied a lot for each technique used. These results made us pivot towards experiments with synthetic dataset.

Also, we analyzed the magnitude of updates for most impacting features which vary a lot. We notice the range of weights for CCMS is much wider compared to cs. We can say range of weights of CCMS is closer to the range given by logistic regression.

	Logistic Regression	Logistic Regression with Top K	Count Sketch	Complementary Count Sketch	Conservative Complementary Count Sketch
Logistic Regression	**100**	90.45	52.45	27.78	48.31
Logistic Regression with Top K	90.45	**100**	55.81	28.63	51.81
Count Sketch	52.45	55.81	**100**	29.47	55.72
Complementary Count Sketch	27.78	26.83	29.47	**100**	29.35
Conservative Complementary Count Sketch	48.31	51.81	55.72	29.35	**100**

Figure 6: Top K Overlap for all feature selection methods over RCV1 Dataset

Minimum weights:

Logistic Regression	Logistic Regression with Top K	Count Sketch	Complementary Count Sketch	Conservative Complementary Count Sketch
-35.90	-36.10	-5.75	-20.27	-6.18

Maximum weights:

Logistic Regression	Logistic Regression with Top K	Count Sketch	Complementary Count Sketch	Conservative Complementary Count Sketch
21.32	21.48	-9.10	11.29	11.48

Figure 7: Maximum and Minimum weights for all feature selection methods over RCV1 Dataset

5.4 Frequency estimation

To estimate frequency we created a synthetic dataset with followed power law distribution with varying α. The dataset included positive and negative numbers where negative number indicated deletion of the particular element from the stream. We observed that the error in frequency estimation for frequent hitters using CCMS was decent when the data set was very skewed. Also, we observe the first proposed CCMS outperforms compared to the CCMS with different hash functions 4 and CCMS using single sketch 5.

alpha values	CS loss	CCMS loss	CCMS using different hash functions	CCMS using Single Sketch
2	3.27	12.79	13.87	100.17
3	0.025	0.27	0.20	1.84

Table 5: Mean Square Loss for top 30 items from variants of count sketches

6 Conclusion

The analysis of complementary count min sketch does not give a better bound than count sketch in terms of error. However, they have promising results in terms of time taken compared to count sketch. From the results we observed under same space constraint count sketch and complementary count min sketch both give same error bound which is $L1$-norm of true frequency vector when dataset is very skewed.

References

[1] Amirali Aghazadeh, Ryan Spring, Daniel LeJeune, Gautam Dasarathy, Anshumali Shrivastava, and Richard G. Baraniuk. MISSION: ultra large-scale feature selection using count-sketches. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 80–88, 2018.

[2] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, Berlin, Heidelberg, 2002. Springer-Verlag.

[3] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, February 2010.

[4] 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, pages 1113–1120, New York, NY, USA, 2009. ACM.

[5] A. Maleki. Coherence analysis of iterative thresholding algorithms. In *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 236–243, 2009.

[6] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 757–772, New York, NY, USA, 2018. ACM.

[7] Morten Houmller Nygaard Jonas Nicolai Hovmand. Estimating frequencies and finding heavy hitters, 2006.

[8] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, December 2004.