MISSION: Ultra Large-Scale Feature Selection using Count-Sketches

Amirali Aghazadeh [*1] **Ryan Spring** [*2] **Daniel LeJeune** [3] **Gautam Dasarathy** [3] **Anshumali Shrivastava** [2] **Richard G. Baraniuk** [3]

Abstract

Feature selection is an important challenge in machine learning. It plays a crucial role in the *explainability* of machine-driven decisions that are rapidly permeating throughout modern society. Unfortunately, the explosion in the size and dimensionality of real-world datasets poses a severe challenge to standard feature selection algorithms. Today, it is not uncommon for datasets to have billions of dimensions. At such scale, even storing the feature vector is impossible, causing most existing feature selection methods to fail. Workarounds like feature hashing, a standard approach to large-scale machine learning, helps with the computational feasibility, but at the cost of losing the interpretability of features. In this paper, we present **MISSION**, a novel framework for ultra large-scale feature selection that performs stochastic gradient descent while maintaining an efficient representation of the features in memory using a Count-Sketch data structure. **MISSION** retains the simplicity of feature hashing without sacrificing the interpretability of the features while using only $\mathcal{O}(\log^2 p)$ working memory. We demonstrate that **MISSION** accurately and efficiently performs feature selection on real-world, large-scale datasets with billions of dimensions.

1. Introduction

Feature selection is an important step in extracting interpretable patterns from data. It has numerous applications in a wide range of areas, including natural-language processing, genomics, and chemistry. Suppose that there are n ordered pairs $(\mathbf{X}_i, y_i)_{i\in[n]}$, where $\mathbf{X}_i \in \mathbb{R}^p$ are p-dimensional *feature vectors*, and $y_i \in \mathbb{R}$ are scalar outputs. Feature selection aims to identify a small subset of features (coordinates of the p-dimensional feature vector) that best models the relationship between the data $\mathbf{X}_i$ and the output y_i.

A significant complication that is common in modern engineering and scientific applications is that the feature space p is ultra high-dimensional. For example, Weinberger introduced a dataset with 16 trillion ($p = 10^{13}$) unique features (Weinberger et al., 2009). A 16 trillion dimensional feature vector (of double 8 bytes) requires 128 terabytes of working memory. Problems from modern genetics are even more challenging. A particularly useful way to represent a long DNA sequence is by a feature vector that counts the occurrence frequency of all length-K sub-strings called K-mers. This representation plays an important role in large-scale regression problems in computational biology (Wood & Salzberg, 2014; Bray et al., 2015; Vervier et al., 2016; Aghazadeh et al., 2016). Typically, K is chosen to be larger than 12, and these strings are composed of all possible combinations of 16 characters ($\{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}$ in addition to 12 wild card characters). In this case, the feature vector dimension is $p = 16^{12} = 2^{48}$. A vector of size 2^{48} single-precision variables requires approximately 1 petabyte of space!

For ultra large-scale feature selection problems, it is impossible to run standard explicit regularization-based methods like ℓ_1 regularization (Shalev-Shwartz & Tewari, 2011; Tan et al., 2014) or to select hyperparameters with a constrained amount of memory (Langford et al., 2009). This is not surprising, because these methods are not scalable in terms of memory and computational time (Duchi et al., 2008). Another important operational concern is that most datasets represent features in the form of strings or tokens. For example, with DNA or n-gram datasets, features are represented by strings of characters. Even in click-through data (McMahan et al., 2013), features are indexed by textual tokens. Observe that mapping each of these strings to a vector component requires maintaining a dictionary whose size equals the length of the feature vector. As a result, one does not even have the capability to create a numerical exact vector representation of the features.

[*]These authors contributed equally and are listed alphabetically [1]Department of Electrical Engineering, Stanford University, Stanford, California [2]Department of Computer Science, Rice University, Houston, Texas [3]Department of Electrical and Computer Engineering, Rice University, Houston, Texas. Correspondence to: Anshumali Shrivastava <anshumali@rice.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Typically, when faced with such large machine learning tasks, the practitioner chooses to do *feature hashing* (Weinberger et al., 2009). Consider a 3-gram string "abc". With feature hashing, one uses a lossy, random hash function $h : \text{strings} \rightarrow \{0, 1, 2, \ldots, R\}$ to map "abc" to a feature number $h(abc)$ in the range $\{0, 1, 2, \ldots, R\}$. This is extremely convenient because it enables one to avoid creating a large look-up dictionary. Furthermore, this serves as a dimensionality reduction technique, reducing the problem dimension to R. Unfortunately, this convenience comes at a cost. Given that useful dimensionality reduction is strictly surjective (i.e., $R < p$), we lose the identity of the original features. This is not a viable option if one cares about both feature selection and interpretability.

One reason to remain hopeful is that in such high-dimensional problems, the data vectors $\mathbf{X}_i$ are extremely sparse (Wood & Salzberg, 2014). For instance, the DNA sequence of an organism contains only a small fraction (at most the length of the DNA sequence) of $p = 16^{12}$ features. The situation is similar whether we are predicting click-through rates of users on a website or if we seek n-gram representations of text documents (Mikolov et al., 2013). In practice, ultra high-dimensional data is almost always ultra-sparse. Thus, loading a sparse data vector into memory is usually not a concern. The problem arises in the intermediate stages of traditional methods, where dense iterates need to be tracked in the main memory. One popular approach is to use *greedy thresholding* methods (Maleki, 2009; Mikolov et al., 2013; Jain et al., 2014; 2017) combined with stochastic gradient descent (SGD) to prevent the feature vector $\boldsymbol{\beta}$ 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. We clearly observe this phenomenon in our evaluations. See Section 5 for details.

In this paper we tackle the ultra large-scale feature selection problem, i.e., feature selection with billions or more dimensions. We propose a novel feature selection algorithm called **MISSION**, a **M**emory-efficient, **I**terative **S**ketching algorithm for **S**parse feature select**ION**. **MISSION**, that takes on all the concerns outlined above. **MISSION** matches the accuracy performance of existing large-scale machine learning frameworks like Vowpal Wabbit (VW) (Agarwal et al., 2014) on real-world datasets. However, in contrast to VW, **MISSION** *can perform feature selection exceptionally well*. Furthermore, **MISSION** significantly surpasses the performance of classical algorithms such as Iterative Hard Thresholding (IHT), which is currently the popular feature selection alternative concerning the problem sizes we consider.

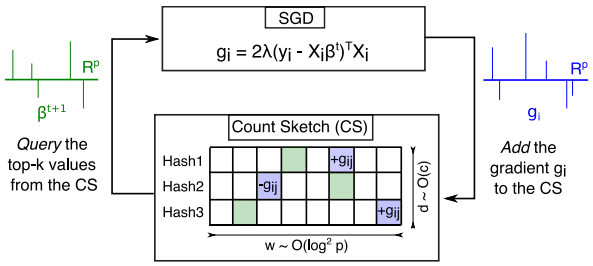

Figure 1. Schematic of the **MISSION** algorithm. **MISSION** iteratively adds the stochastic gradient term $g_i \in \mathbb{R}^p$ into a Count-Sketch and queries back the top-k heavy hitters from the Count-Sketch. The Count-Sketch requires $\mathcal{O}(\log^2 p)$ memory to store a sketch of the $\mathcal{O}(p)$-dimensional feature vector $\boldsymbol{\beta}$.

Contributions: In this work, we show that the two-decade old Count-Sketch data structure (Charikar et al., 2002) from the streaming algorithms literature is ideally suited for ultra large-scale feature selection. The Count-Sketch data structure enables us to retain the convenience of feature hashing along with the identity of important features. Moreover, Count-Sketch can accumulate gradients updates over several iterations because of linear aggregation. This aggregation eliminates the problem of myopia associated with existing greedy thresholding approaches.

In particular, we force the parameters (or feature vector) to reside in a memory-efficient Count-Sketch data structure (Charikar et al., 2002). SGD gradient updates are easily applied to the Count-Sketch. Instead of moving in the gradient direction and then greedily projecting into a subspace defined by the regularizer (e.g., in the case of LASSO-based methods), **MISSION** adds the gradient directly into the Count-Sketch data structure, where it aggregates with all the past updates. See Fig. 1 for the schematic. At any point of time in the iteration, this data structure stores a compressed, randomized, and noisy sketch of the sum of all the gradient updates, while preserving the information of the *heavy-hitters*—the coordinates that accumulate the highest amount of energy. In order to find an estimate of the feature vector, **MISSION** queries the Count-Sketch. The Count-Sketch is used in conjunction with a top-k heap, which explicitly stores the features with the heaviest weights. Only the features in the top-k heap are considered active, and the rest are set to zero. However, a representation for every weight is stored, in compressed form, inside the Count-Sketch.

We demonstrate that **MISSION** surpasses the sparse recovery performance of classical algorithms such as Iterative Hard Thresholding (IHT), which is the only other method we could run at our scale. In addition, experiments suggest that the memory requirements of **MISSION** scale well with the dimensionality p of the problem. **MISSION** matches the accuracy of existing large-scale machine learning frameworks like Vowpal Wabbit (VW) on real-world, large-scale

datasets. Moreover, MISSION achieves comparable or even better accuracy while using significantly fewer features.

2. Review: Streaming Setting and the Count-Sketch Algorithm

In the streaming setting, we are given a very high-dimensional vector $\boldsymbol{\beta} \in \mathbb{R}^p$ that is too costly to store in memory. We see only a very long sequence of updates over time. The only information available at time t is of the form (i, Δ), which means that coordinate i is incremented (or decremented) by the amount Δ. We are given a limited amount of storage, on the order of $\mathcal{O}(\log p)$, which means that we can never store the entire sequence of updates. Sketching algorithms aim to estimate the value of current item i, after any number of updates using only $\mathcal{O}(\log p)$ memory. Accurate estimation of heavy coordinates is desirable.

Count-Sketch is a popular algorithm for estimation in the streaming setting. Count-Sketch keeps a matrix of counters (or bins) $\mathcal{S}$ of size $d \times w \sim \mathcal{O}(\log p)$, where d and w are chosen based on the accuracy guarantees. The algorithm uses d random hash functions $h_j \;\; j \in \{1,\ 2, ...,\ d\}$ to map the vector's components to bins w. $h_j : \{1,\ 2, ...,\ p\} \rightarrow \{1,\ 2, ...,\ w\}$ Every component i of the vector is hashed to d different bins. In particular, for any row j of sketch $\mathcal{S}$, component i is hashed into bin $\mathcal{S}(j, h_j(i))$. In addition to h_j, Count-Sketch uses d random sign functions to map the components of the vectors randomly to $\{+1,\ -1\}$. i.e., $s_i : \{1,\ 2, ...,\ D\} \rightarrow \{+1, -1\}$ A picture of this sketch data structure with three hash functions in shown inside Fig. 1.

The Count-Sketch supports two operations: UPDATE(item i, increment Δ) and QUERY(item i). The UPDATE operation updates the sketch with any observed increment. More formally, for an increment Δ to an item i, the sketch is updated by adding $s_j(i)\Delta$ to the cell $\mathcal{S}(j, h_j(i))\ \forall j \in \{1,\ 2, ...,\ d\}$. The QUERY operation returns an estimate for component i, the median of all the d different associated counters.

It has been shown that, for any sequence of streaming updates (addition or subtraction) to the vector $\boldsymbol{\beta}$, Count-Sketch provides an unbiased estimate of any component i, $\widehat{\boldsymbol{\beta}}_i$ such that the following holds with high probability,

$$\boldsymbol{\beta}_i - \epsilon||\boldsymbol{\beta}||_2 \;\leq\; \widehat{\boldsymbol{\beta}}_i \;\leq\; \boldsymbol{\beta}_i + \epsilon||\boldsymbol{\beta}||_2. \tag{1}$$

It can be shown that the Eq. (1) is sufficient to achieve near-optimal guarantees for sparse recovery with the given space budget. Furthermore, these guarantees also meet the best compressed sensing lower bounds in terms of the number of counters (or measurements) needed for sparse recovery (Indyk, 2013).

3. Problem Formulation

Consider the feature selection problem in the ultra high-dimensional setting: We are given the dataset $(\mathbf{X}_i, y_i)$ for $i \in [n] = \{1, 2, \ldots, n\}$, where $\mathbf{X}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ denote the i^{th} measured and response variables. We are interested in finding the k-sparse (k non-zero entries) feature vector (or regressor) $\boldsymbol{\beta} \in \mathbb{R}^p$ from the optimization problem

$$\min_{\|\boldsymbol{\beta}\|_0 = k} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2, \tag{2}$$

where $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ and $\mathbf{y} = [y_1, y_1, \ldots, y_n]$ denote the data matrix and response vector and the ℓ_0-norm $\|\boldsymbol{\beta}\|_0$ counts the number of non-zero entries in $\boldsymbol{\beta}$.

We are interested in solving the feature selection problem for ultra 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.

3.1. Hard Thresholding Algorithms

Among the menagerie of 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 $\mathcal{O}(k\log(p))$ memory (Jain et al., 2014; Blumensath & Davies, 2009). The *iterative hard thresholding* (IHT) algorithm generates the following iterates for the i^{th} variable in an stochastic gradient descent (SGD) framework

$$\boldsymbol{\beta}^{t+1} \leftarrow H_k(\boldsymbol{\beta}^t - 2\lambda \left(y_i - \mathbf{X}_i\boldsymbol{\beta}^t\right)^T \mathbf{X}_i) \tag{3}$$

The sparsity of the feature vector $\boldsymbol{\beta}^t$, enforced by the hard thresholding operator H_k, alleviates the need to store a vector of size $\mathcal{O}(p)$ in the memory in order to keep track of the changes of the features over the iterates.

Unfortunately, because it only retains the top-k elements of $\boldsymbol{\beta}$, 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 the design matrix $\mathbf{X}$ is not random, normalized, or well-conditioned. In this regime, the gradient terms corresponding to the true support typically arrive in lagging order and are prematurely clipped in early iterations by H_k. The effect of these *lagging gradients* is present even in the SGD framework, because the gradients are quite noisy, and only a small fraction of the energy of the true gradient is expressed in each iteration. It is not difficult to see that these small energy, high noise signals can easily cause the greedy hard thresholding operator to make sub-optimal or incorrect decisions. Ideally, we want to accumulate the gradients to get enough confidence in

Algorithm 1 MISSION

Initialize: $\beta^0 = 0$, $\mathcal{S}$ (Count-Sketch), λ (Learning Rate)
while not stopping criteria **do**
  Find the gradient update $g_i = \lambda \left(2\left(y_i - \mathbf{X}_i \boldsymbol{\beta}^t\right)^T \mathbf{X}_i\right)$
  Add the gradient update to the sketch $g_i \rightarrow \mathcal{S}$
  Get the top-k heavy-hitters from the sketch $\boldsymbol{\beta}^{t+1} \leftarrow \mathcal{S}$
end while
Return: The top-k heavy-hitters from the Count-Sketch

signal and to average out any noise. However, accumulating gradients will make the gradient vector dense, blowing up the memory requirements. This aforementioned problem is in fact symptomatic of all other thresholding variants including the *Iterative algorithm with inversion* (ITI) (Maleki, 2009) and the *Partial hard thresholding* (PHT) algorithm (Jain et al., 2017).

4. The MISSION Algorithm

We now describe the MISSION algorithm. First, we initialize the Count-Sketch $\mathcal{S}$ and the feature vector $\boldsymbol{\beta}^{t=0}$ with zeros entries. The Count-Sketch hashes a p-dimensional vector into $\mathcal{O}(\log^2 p)$ buckets (Recall Fig. 1). We discuss this particular choice for the size of the Count-Sketch and the memory-accuracy trade offs of MISSION in Sections 5.3 and 6.1.

At iteration t, MISSION selects a random row $\mathbf{X}_i$ from the data matrix $\mathbf{X}$ and computes the stochastic gradient update term using the learning rate λ. $g_i = 2\lambda\left(y_i - \mathbf{X}_i \boldsymbol{\beta}^t\right)^T \mathbf{X}_i$ i.e. the usual gradient update that minimizes the *unconstrained* quadratic loss $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$. The data vector $\mathbf{X}_i$ and the corresponding stochastic gradient term are sparse. We then add the non-zero entries of the stochastic gradient term $\{g_{ij} : \forall j \;\; g_{ij} > 0\}$ to the Count-Sketch $\mathcal{S}$. Next, MISSION queries the top-k values of the sketch to form $\boldsymbol{\beta}^{t+1}$. We repeat the same procedure until convergence. MISSION returns the top-k values of the Count-Sketch as the final output of the algorithm. The MISSION algorithm is detailed in Alg. 1. MISSION easily extends to other loss functions such as the hinge loss and logistic loss.

MISSION is Different from Greedy Thresholding: Denote the gradient vector update at any iteration t as u_t. It is not difficult to see that starting with an all-zero vector $\boldsymbol{\beta}_0$, at any point of time t, the Count-Sketch state is equivalent to the sketch of the vector $\sum_{i=1}^{t} u_t$. In other words, the sketch aggregates the compressed aggregated vector. Thus, even if an individual SGD update is noisy and contains small signal energy, thresholding the Count-Sketch is based on the average update over time. This averaging produces a robust signal that cancels out the noise. We can therefore expect MISSION to be superior over thresholding.

In the supplementary materials, we present initial theoretical results on the convergence of MISSION. Our results show that, under certain assumptions, the full-gradient-descent version of MISSION converges *geometrically* to the true parameter $\boldsymbol{\beta} \in \mathbb{R}^p$ up to some additive constants. The exploration of these assumptions and the extension to the SGD version of MISSION are exciting avenues for future work.

Feature Selection with the Ease of Feature Hashing: As argued earlier, the features are usually represented with strings, and we do not have the capability to map each string to a unique index in a vector without spending $O(p)$ memory. Feature hashing is convenient, because we can directly access every feature using hashes. We can use any lossy hash function for strings. MISSION only needs a few independent hash functions (3 in our Count-Sketch implementation) to access any component. The top-k estimation is done efficiently using a heap data structure of size k. Overall, we only access the data using efficient hash functions, which can be easily implemented in large-scale systems.

5. Simulations

We designed a set of simulations to evaluate MISSION in a controlled setting. In contrast to the ultra large-scale, real-world experiments of Section 6, in the section the data matrices are drawn from a random Gaussian distribution and the ground truth features are known.

5.1. Phase Transition

We first demonstrate the advantage of MISSION over greedy thresholding in feature selection. For this experiment, we modify MISSION slightly to find the root of the algorithmic advantage of MISSION: we replace the Count-Sketch with an "identity" sketch, or a sketch with a single hash function, $h(i) = i$. In doing so, we eliminate the complexity that Count-Sketch adds to the algorithm, so that the main difference between MISSION and IHT is that MISSION accumulates the gradients. To improve stability, we scale the non top-k elements of $\mathcal{S}$ by a factor $\gamma \in (0, 1)$ that begins very near 1 and is gradually decreased until the algorithm converges. *Note:* it is also possible to do this scaling in the Count-Sketch version of MISSION efficiently by exploiting the linearity of the sketch.

Fig. 2 illustrates the empirical phase transition curves for sparse recovery using MISSION and the hard thresholding algorithms. The phase transition curves show the points where the algorithm successfully recovers the features in $> 50\%$ of the random trails. MISSION shows a better phase transition curve compared to IHT by a considerable gap.

Table 1. Comparison of **MISSION** against hard thresholding algorithms in subset selection under adversarial effects. We first report the percentage of instances in which the algorithms accurately find the solution (ACC) with no attenuation ($\alpha = 1$) over 100 random trials. We then report the mean of the maximum level of attenuation α applied to the columns of design $\mathbf{X}$ before the algorithms fail to recover the support of $\boldsymbol{\beta}$ (over the trials that all algorithms can find the solution with $\alpha = 1$).

(n, k)	**MISSION**		IHT		ITI		PHT	
	$\text{ACC}_{\alpha=1}$	α	$\text{ACC}_{\alpha=1}$	α	$\text{ACC}_{\alpha=1}$	α	$\text{ACC}_{\alpha=1}$	α
(100, 2)	**100%**	**2.68 ± 0.37**	**100%**	1.49 ± 0.33	91%	1.33 ± 0.23	64%	2.42 ± 0.87
(100, 3)	**100%**	**2.52 ± 0.36**	92%	1.36 ± 0.46	70%	1.15 ± 0.20	42%	2.05 ± 0.93
(100, 4)	**100%**	**2.53 ± 0.23**	72%	1.92 ± 0.91	37%	1.03 ± 0.09	39%	2.13 ± 1.07
(200, 5)	**100%**	**4.07 ± 0.36**	99%	2.34 ± 1.12	37%	1.15 ± 0.22	83%	2.75 ± 1.30
(200, 6)	**100%**	**4.17 ± 0.24**	97%	2.64 ± 1.14	23%	1.11 ± 0.12	73%	2.26 ± 1.33
(200, 7)	**100%**	**4.07 ± 0.11**	83%	1.64 ± 1.01	14%	1.11 ± 0.12	75%	3.39 ± 1.36

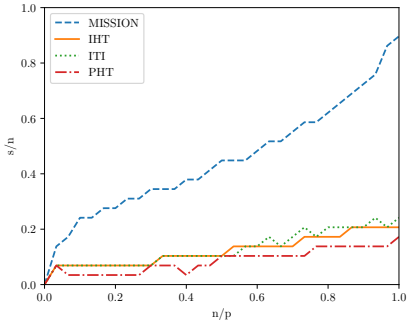

Figure 2. Empirical phase transition in recovering a binary feature vector $\boldsymbol{\beta}$ in $p = 1000$-dimensional space with a Gaussian data matrix $\mathbf{X}$. We illustrate the empirical 50% probability of success curves averaged over $T = 20$ trials. **MISSION** outperforms the thresholding algorithms by a large margin.

5.2. Lagging Gradient: Superiority of Count-Sketches over Greedy Thresholding

A major problem with the IHT algorithm, especially in large-scale SGD settings, is with thresholding the coordinates with small gradients in the earlier iterations. IHT misses these coordinates, since they become prominent only after the gradients accumulate with the progression of the algorithm. The problem is amplified with noisy gradient updates such as SGD, which is unavoidable for large datasets.

This phenomenon occurs frequently in sparse recovery problems. For example, when the coordinates that correspond to the columns of the data matrix with smaller energy lag in the iterations of gradient descent algorithm, IHT thresholds these *lagging-gradient* coordinates in first few iterations, and they never show up again in the support. In contrast, **MISSION** retains a footprint of the gradients of all the previous iterations in the Count-Sketch. When the total sum of the gradient of a coordinate becomes prominent, the coordinate joins the support after querying the top-k heavy hitters from the Count-Sketch. We illustrate this phenomena in sparse recovery using synthetic experiments. We recover sparse vector $\boldsymbol{\beta}$ from its random linear measurements $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$, where the energy of $\mathbf{X}$ is imbalanced across its columns. In this case, the gradients corresponding to the columns (coordinates) with smaller energy typically lag and are thresholded by IHT.

To this end, we first construct a random Gaussian data matrix $\mathbf{X} \in \mathbb{R}^{900\times1000}$, pick a sparse vector $\boldsymbol{\beta}$ that is supported on an index set $\mathcal{I}$, and then attenuate the energy of the columns of $\mathbf{X}$ supported by the indices in $\mathcal{I}$ by an attenuation factor of $\alpha = \{1, 1.25, 1.5, 1.75, 2, \ldots, 5\}$. Note that $\alpha = 1$ implies that no attenuation is applied to the matrix. In Table 1, we report the maximum attenuation level applied to a column of data matrix $\mathbf{X}$ before the algorithms fail to fully recover the support set $\mathcal{I}$ from $\mathbf{y} = \boldsymbol{\beta}\mathbf{X}$. We observe that **MISSION** is consistently and up to three times more robust against adversarial attenuation of the columns of the data matrix in various design settings.

The robustness of **MISSION** to the attenuation of the columns of $\mathbf{X}$ in sparse recovery task suggests that the Count-Sketch data structure enables gradient-based optimization methods such as IHT to store a footprint (or sketch) of all the gradients from the previous iterations and deliver them back when they become prominent.

5.3. Logarithmic Scaling of the Count-Sketch Memory in MISSION

In this section we demonstrate that the memory requirements of **MISSION** grows polylogarithmically in the dimension of the problem p. We conduct a feature selection experiment with a data matrix $\mathbf{X} \in \mathbb{R}^{100\times p}$ whose entries are drawn from i.i.d. random Gaussian distributions with zero mean and unit variance. We run **MISSION** and IHT to recover the feature vector $\boldsymbol{\beta}$ from the output vector $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$, where the feature vector $\boldsymbol{\beta}$ is a $k = 5$-sparse vector with random support. We repeat the same experiment 1000 times with different realizations for the sparse feature vector $\boldsymbol{\beta}$ and report the results in Fig. 3. The left plot illustrates the feature selection accuracy of the algorithms as the dimension of the problem p grows. The right plot illustrates the minimum memory requirements of the algorithms to recover the features with 100% accuracy.

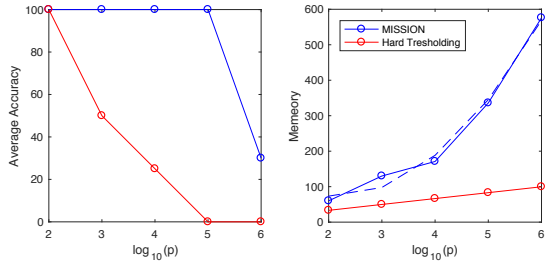

Figure 3. Feature selection accuracy and memory requirements of **MISSION** and Hard Thresholding. The memory requirements of **MISSION** grows polylogarithmcially $\sim \mathcal{O}(\log^2(p))$ (dotted line illustrates quadratic fit) in p. With only a logarithmic factor more memory, **MISSION** has significant advantage over Hard Thresholding in terms of feature selection accuracy.

The plots reveal an interesting phenomenon. The size of the Count-Sketch in **MISSION** scales only polylogarithmically with the dimension of the problem. This is surprising since the aggregate gradient in a classical SGD framework becomes typically dense in early iterations and thus requires a memory of order $\mathcal{O}(p)$. **MISSION**, however, stores only the *essential* information of the features in the sketch using a poly-logarithmic sketch size. Note that IHT sacrifices accuracy to achieve a small memory footprint. At every iteration IHT eliminates all the information except for the top-k features. We observe that, using only a logarithmic factor more memory, **MISSION** has a significant advantage over IHT in recovering the ground truth features.

6. Experiments

All experiments were performed on a single machine, 2x Intel Xeon E5-2660 v4 processors (28 cores / 56 threads) with 512 GB of memory. The code [1] for training and running our randomized-hashing approach is available online. We designed the experiments to answer these questions:

1. Does **MISSION** outperform IHT in terms of classification accuracy? In particular, how much does myopic thresholding affect IHT in practice?
2. How well does **MISSION** match the speed and accuracy of feature hashing (FH)?
3. How does changing the number of top-k features affect the accuracy and behaviour of the different methods?
4. What is the effect of changing the memory size of the Count-Sketch data structure on the classification accuracy of **MISSION** in read-world datasets?
5. Does **MISSION** scale well in comparison to the different methods on the ultra large-scale datasets (> 350 GB in size)?

[1] https://github.com/rdspring1/MISSION

6.1. Large-scale Feature Extraction

Datasets: We used four datasets in the experiments: 1) KDD2012, 2) RCV1, 3) Webspam–Trigram, 4) DNA [2]. The statistics of these datasets are summarized in Table 2.

Table 2. Feature extraction dataset statistics.

Dataset	Dim (p)	Train Size (n)	Test Size
KDD 2012	54,686,452	119,705,032	29,934,073
RCV1	47,236	20,242	677,399
Webspam	16,609,143	280,000	70,000
DNA (Tiny)	14,890,408	1,590,000	207,468

The DNA metagenomics dataset is a multi-class classification task where the model must classify 15 different bacteria species using DNA K-mers. We sub-sampled the first 15 species from the original dataset containing 193 species. We use all of the species in the DNA Metagenomics dataset for the large-scale experiments (See Section 6.2). Following standard procedures, each bacterial species is associated with a reference genome. Fragments are sampled from the reference genome until each nucleotide is covered c times on average. The fragments are then divided into K-mer sub-strings. We used fragments of length 200 and K-mers of length 12. Each model was trained and tested with mean coverage $c = \{0.1, 1\}$ respectively. For more details, see (Vervier et al., 2016). The feature extraction task is to find the DNA K-mers that best represent each bacteria class.

We implemented the following approaches to compare and contrast against our approach: For all methods, we used the logistic loss for binary classification and the cross-entropy loss for multi-class classification.

MISSION: As described in Section 4.
Iterative Hard Thresholding (IHT): 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. Since the features are strings or integers, we used a sorted heap to store and manipulate the top-k elements. This was the only algorithm we could successfully run over the large datasets on our single machine.
Batch IHT: A modification to IHT that uses mini-batches such that the gradient sparsity is the same as the number of elements in the count-sketch. We accumulate features and then sort and prune to find the top-k features. This accumulate, sort, prune process is repeated several times during training. **Note:** This setup requires significantly more memory than **MISSION**, because it explicitly stores the feature strings. The memory cost of maintaining a set of string features can be orders of magnitude more than the flat array used by **MISSION**. See Bloom Filters (Broder & Mitzenmacher, 2004) and related literature. This setup is not scalable to large-scale datasets.

[2] http://projects.cbio.mines-paristech.fr/largescalemetagenomics/

Feature Hashing (FH): A standard machine learning algorithm for dimensionality reduction that reduces the memory cost associated with large datasets. FH is not a feature selection algorithm and cannot identify important features. (Agarwal et al., 2014)

Experimental Settings: The MISSION and IHT algorithms searched for the same number of top-k features. To ensure fair comparisons, the size of the Count-Sketch and the feature vector allocated for the FH model were equal. The size of the MISSION and FH models were set to the nearest power of 2 greater than the number of features in the dataset. For all the experiments, the Count-Sketch data structure used 3 hash functions, and the model weights were divided equally among the hash arrays. For example, with the (Tiny) DNA metagenomics dataset, we allocated 24 bits or 16,777,216 weights for the FH model. Given 3 hash functions and 15 classes, roughly 372,827 elements were allocated for each class in the Count-Sketch.

MISSION, IHT, FH Comparison: Fig. 4 shows that MISSION surpasses IHT in classification accuracy in all four datasets, regardless of the number of features. In addition, MISSION closely matches FH, which is significant because FH is allowed to model a much larger set of features than MISSION or IHT. MISSION is 2–4$\times$ slower than FH, which is expected given that MISSION has the extra overhead of using a heap to track the top-k features.

MISSION's accuracy rapidly rises with respect to the number of top-k features, while IHT's accuracy plateaus and then grows slowly to match MISSION. This observation corroborates our insight that the greedy nature of IHT hurts performance. When the number of top-k elements is small, the capacity of IHT is limited, so it picks the first set of features that provides good performance, ignoring the rest. On the other hand, MISSION decouples the memory from the top-k ranking, which is based on the aggregated gradients in the compressed sketch. By the linear property of the count-sketch, this ensures that the heavier entries occur in the top-k features with high probability.

Count-Sketch Memory Trade-Off: Fig. 5 shows how MISSION's accuracy degrades gracefully, as the size of the Count-Sketch decreases. In this experiment, MISSION only used the top 500K features for classifying the Tiny DNA metagenomics dataset. When the top-k to Count-Sketch ratio is 1, then 500K weights were allocated for each class and hash array in the Count-Sketch data structure. The Batch IHT baseline was given 8,388,608 memory elements per class, enabling it to accumulate a significant number of features before thresholding to find the top-k features. This experiment shows that MISSION immediately outperforms IHT and Batch IHT, once the top-k to Count-Sketch ratio is 1:1. Thus, MISSION provides a unique memory-accuracy knob at any given value of top-k.

6.2. Ultra Large-Scale Feature Selection

Here we demonstrate that MISSION can extract features from three large-scale datasets: Criteo 1TB, Splice-Site, and DNA Metagenomics.

Table 3. Ultra Large-Scale dataset statistics

Dataset	Dim (p)	Train Size (n)	Test Size
Criteo	1M	4,195,197,692	178,274,637
Splice-Site	11.7M	50,000,000	4,627,840
DNA	17.3M	13,792,260	354,285

Criteo 1TB: The Criteo 1TB [3] dataset represents 24 days of click-through logs—23 days (training) + 1 day (testing). The task for this dataset is click-through rate (CTR) prediction—How likely is a user to click an ad? The dataset contains over 4 billion (training) and 175 million (testing) examples (2.5 TB of disk space). The performance metric is Area Under the ROC Curve (AUC). The VW baseline [4] achieved 0.7570 AUC score. MISSION and IHT scored close to the VW baseline with 0.751 AUC using only the top 250K features.

Table 4. Criteo 1TB. Top-K Features: 250K

Metric	MISSION	IHT	VW
AUC	0.751	0.752	0.757

Splice-Site: The task for this dataset is to distinguish between true and fake splice sites using the local context around the splice site in-question. The dataset is highly skewed (few positive, many negative values), and so the performance metric is average precision (AP). Average precision is the precision score averaged over all recall scores ranging from 0 to 1. The dataset contains over 50 million (training) and 4.6 million (testing) examples (3.2 TB of disk space). All the methods were trained for a single epoch with a learning rate of 0.5. MISSION, Batch IHT, and SGD IHT tracked the top 16,384 features. FH, MISSION, and Batch IHT used 786,432 extra memory elements. MISSION significantly outperforms Batch IHT and SGD IHT by 2.3%. Also, unlike in Fig. 5, the extra memory did not help Batch IHT, since it performed the same as SGD IHT. MISSION (17.5 hours) is 15% slower than FH (15 hours) in wall-clock running time.

Table 5. Splice-Site: Top-k features: 16,384. Memory elements: 786,432. MISSION outperforms Batch IHT and SGD IHT.

Metric	FH	MISSION	Batch IHT	SGD IHT
AP	0.522	0.510	0.498	0.498

DNA Metagenomics: This experiment evaluates MISSION's performance on a medium-sized metagenomics dataset. The parameters from the Tiny (15 species) dataset in Section 6.1 are shared with this experiment, except the

[3] https://www.kaggle.com/c/criteo-display-ad-challenge

[4] https://github.com/rambler-digital-solutions/criteo-1tb-benchmark

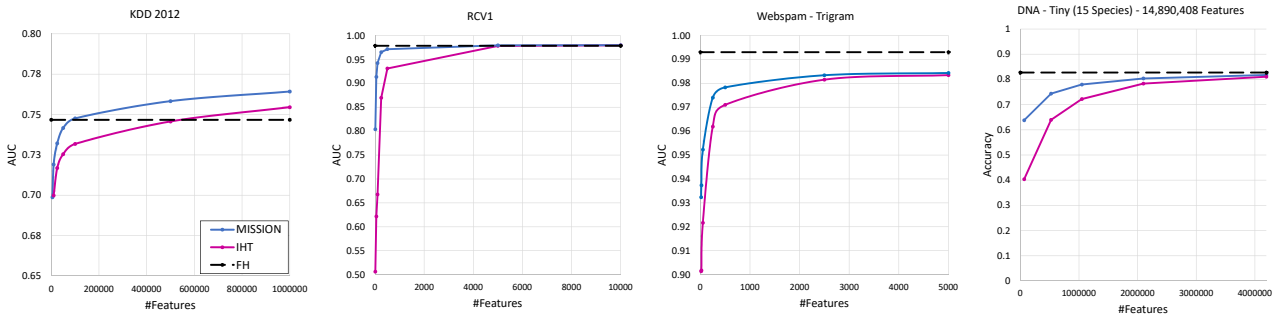

Figure 4. Feature selection on the KDD-2012, RCV1, Webspam, and DNA Metagenomic (Tiny) datasets. FH is not a feature selection baseline. It is marked by a dashed black line, indicating that its performance is invariant to the number of top-k features.

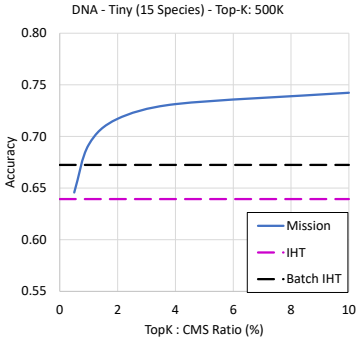

Figure 5. Count-Sketch Memory/Accuracy Trade-off using the DNA Metagenomics (Tiny) dataset. The x-axis is the ratio of Count-Sketch memory per class and hash-array to the # top-k features per class. Memory Elements for Batch IHT: 8,388,608.

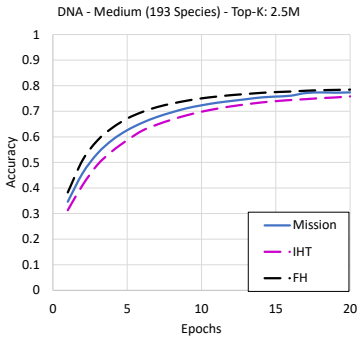

Table 6. Ultra Large-Scale Feature Selection for the DNA Metagenomics (Medium) Dataset (193 species), Mean Coverage $c = 5$.

number of species is increased to 193. The size of a sample batch with mean coverage $c = 1$ increased from 7 GB (Tiny) to 68 GB (Medium). Each round (mean coverage $c = 0.25$) contains 3.45 million examples and about 16.93 million unique non-zero features (p). **MISSION** and IHT tracked the top 2.5 million features per class. The FH baseline used 2^{31} weights, about 11.1 million weights per class, and we allocated the same amount of space for the Count-Sketch. Each model was trained on a dataset with coverage $c = 5$.

Fig. 6 shows the evolution of classification accuracy over time for **MISSION**, IHT, and the FH baseline. After 5 epochs, **MISSION** closely matches the FH baseline. **Note: MISSION** *converges faster than IHT* such that **MISSION** is 1–4 rounds ahead of IHT, with the gap gradually increasing over time. On average, the running time of **MISSION** is 1–2$\times$ slower than IHT. However, this experiment demonstrates that since **MISSION** converges faster, it actually needs less time to reach a certain accuracy level. Therefore, **MISSION** is effectively faster and more accurate than IHT.

7. Implementation Details and Discussion

Scalability and Parallelism: IHT finds the top-k features after each gradient update, which requires sorting the features based on their weights before thresholding. The speed of the sorting process is improved by using a heap data structure, but it is still costly per update. **MISSION** also uses a heap to store its top-k elements, but it achieves the same accuracy as IHT with far fewer top-k elements because of the Count-Sketch. (Recall Section 4)

Another suggested improvement for the top-k heap is to use lazy updates. Updating the weight of a feature does not change its position in the heap very often, but still requires an $O(\log n)$ operation. With lazy updates, the heap is updated only if it the change is significant. $|x_t - x_0| \geq \epsilon$, i.e. the new weight at time t exceeds the original value by some threshold. This tweak significantly reduces the number of heap updates at the cost of slightly distorting the heap.

8. Conclusion and Future Work

In this paper, we presented **MISSION**, a new framework for ultra large-scale feature selection that performs hard thresholding and SGD while maintaining an efficient, approximate representation for all features using a Count-Sketch data structure. **MISSION** retains the simplicity of feature hashing without sacrificing the interpretability of the features.

Interaction features are important for scientific discovery with DNA Metagenomics (Basu et al., 2018). Traditionally, the polynomial kernel trick enabled machine learning algorithms to explore this feature space implicitly without the exponential increase in dimensionality. However, this exponential cost is unavoidable with feature extraction. Going forward, we are interested in leveraging our **MISSION** framework to explore pairwise or higher interaction features.

Acknowledgements

AAA, DL, GD, and RB were supported by the DOD Vannevar Bush Faculty Fellowship grant N00014-18-1-2047, NSF grant CCF-1527501, ARO grant W911NF-15-1-0316, AFOSR grant FA9550-14-1-0088, ONR grant N00014-17-1-2551, DARPA REVEAL grant HR0011-16-C-0028, and an ONR BRC grant for Randomized Numerical Linear Algebra. RS and AS were supported by NSF-1652131, AFOSR-YIP FA9550-18-1-0152, and ONR BRC grant for Randomized Numerical Linear Algebra. The authors would also like to thank NVIDIA and Amazon for gifting computing resources.

References

Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

Aghazadeh, A., Lin, A. Y., Sheikh, M. A., Chen, A. L., Atkins, L. M., Johnson, C. L., Petrosino, J. F., Drezek, R. A., and Baraniuk, R. G. Universal microbial diagnostics using random dna probes. *Science advances*, 2(9): e1600025, 2016.

Basu, S., Kumbier, K., Brown, J. B., and Yu, B. Iterative random forests to discover predictive and stable high-order interactions. *Proceedings of the National Academy of Sciences*, 2018. ISSN 0027-8424. doi: 10.1073/pnas.1711236115. URL http://www.pnas.org/content/early/2018/01/17/1711236115.

Blumensath, T. and Davies, M. E. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

Bray, N., Pimentel, H., Melsted, P., and Pachter, L. Near-optimal RNA-Seq quantification. *arXiv preprint arXiv:1505.02710*, 2015.

Broder, A. and Mitzenmacher, M. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4): 485–509, 2004.

Charikar, M., Chen, K., and Farach-Colton, M. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pp. 693–703. Springer, 2002.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 272–279. ACM, 2008.

Indyk, P. Sketching via hashing: From heavy hitters to compressed sensing to sparse fourier transform. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 87–90. ACM, 2013.

Jain, P., Tewari, A., and Kar, P. On iterative hard thresholding methods for high-dimensional m-estimation. In *Advances in Neural Information Processing Systems*, pp. 685–693, 2014.

Jain, P., Tewari, A., and Dhillon, I. S. Partial hard thresholding. *IEEE Transactions on Information Theory*, 63(5): 3029–3038, 2017.

Langford, J., Li, L., and Zhang, T. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10(Mar):777–801, 2009.

Maleki, A. Coherence analysis of iterative thresholding algorithms. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pp. 236–243. IEEE, 2009.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1222–1230. ACM, 2013.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Shalev-Shwartz, S. and Tewari, A. Stochastic methods for ℓ_1-regularized loss minimization. *Journal of Machine Learning Research*, 12(6):1865–1892, 2011.

Tan, M., Tsang, I. W., and Wang, L. Towards ultrahigh dimensional feature selection for big data. *Journal of Machine Learning Research*, 15(1):1371–1429, 2014.

Vervier, K., Mahé, P., Tournoud, M., Veyrieras, J.-B., and Vert, J.-P. Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, 32(7): 1023–1032, 2016.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1113–1120. ACM, 2009.

Wood, D. E. and Salzberg, S. L. Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):1, 2014.