Sketching Linear Classifiers over Data Streams

Kai Sheng Tai, Vatsal Sharan, Peter Bailis, Gregory Valiant
Stanford University

ABSTRACT

We introduce a new sub-linear space sketch—the Weight-Median Sketch—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, relative deltoid detection, and streaming estimation of pointwise mutual information. Unlike related sketches that capture the most frequently-occurring features (or items) in a data stream, the Weight-Median Sketch captures the features that are most discriminative of one stream (or class) compared to another. The Weight-Median Sketch adopts the core data structure used in the Count-Sketch, but, instead of sketching counts, it captures sketched gradient updates to the model parameters. We provide a theoretical analysis that establishes recovery guarantees for batch and online learning, and demonstrate empirical improvements in memory-accuracy trade-offs over alternative memory-budgeted methods, including count-based sketches and feature hashing.

ACM Reference Format:
Kai Sheng Tai, Vatsal Sharan, Peter Bailis, Gregory Valiant. 2018. Sketching Linear Classifiers over Data Streams. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196930

1 INTRODUCTION

With the rapid growth of streaming data volumes, memory-efficient sketches are an increasingly important tool in analytics tasks such as finding frequent items [10, 15, 41, 52], estimating quantiles [26, 45], and approximating the number of distinct items [24]. Sketching algorithms trade off between space utilization and approximation accuracy, and are therefore well suited to settings where memory is scarce or where highly-accurate estimation is not essential. For example, sketches are used in measuring traffic statistics on resource-constrained network switch hardware [74] and in processing approximate aggregate queries in sensor networks [12]. Moreover, even in commodity server environments where memory is more plentiful, sketches are useful as a lightweight means to perform approximate analyses like identifying frequent search queries or URLs within a broader stream processing pipeline [6].

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196930

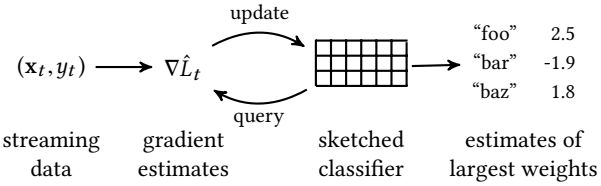

Figure 1: Overview of our approach, where online updates are applied to a sketched (i.e., compressed) classifier from which estimates of the largest weights can be retrieved.

Machine learning is applicable in many of the same resource-constrained deployment scenarios as existing sketching algorithms. With the widespread adoption of mobile devices, wearable electronics, and smart home appliances, there is increasing interest in memory-constrained learning, where statistical models on these devices are updated on-the-fly in response to locally-observed data [36, 44, 49, 62]. These online updates allow ML-enabled systems to adapt to individual users or local environments. For example, language models on mobile devices can be personalized in order to improve the accuracy of speech recognition systems [49], mobile facial recognition systems can be updated based on user supervision [36], packetfilters on network routers can be incrementally improved [18, 67], and human activity classifiers can be tailored to individual motion patterns for more accurate classification [44, 75].

Online learning in memory-constrained environments is particularly challenging in high-dimensional feature spaces. For example, consider a spam classifier on text data that is continually updated as new messages are observed and labeled as *spam* or *not spam*. The memory cost of retaining n-gram features grows rapidly as new token combinations are observed in the stream. In an experiment involving an ~80M token newswire dataset [11], we recorded ~47M unique word pairs that co-occur within 5-word spans of text. Disregarding the space required to store strings, maintaining integer vocabulary indexes and 32-bitfloating point weights for each of these features would require approximately 560MB of memory. Thus, the memory footprint of classifiers over high-dimensional streaming data can quickly exceed the memory constraints of many deployment environments. Moreover, it is not sufficient to simply apply existing sketches for identifying frequently-occurring features, since the features that occur most often are not necessarily the most discriminative.

In this work, we develop a new sketching algorithm that targets ML applications in these memory-constrained settings. Building on prior work on sketching for identifying frequent items, we introduce the *Weight-Median Sketch* (WM-Sketch) for learning compressed linear classifiers over data streams. Figure 1 illustrates the high-level approach: wefirst allocate afixed region of memory as the sketch data structure, and as new examples are observed in the stream, the weights stored in this structure are updated via gradient descent on a given loss function. In contrast to previous work that employs the "hashing trick" to reduce the memory footprint

of a classifier [60, 69], the WM-Sketch supports the approximate recovery of the most heavily-weighted features in the classifier: at any time, we can efficiently return a list of the top-K features along with estimates of their weights in an uncompressed classifier trained over the same sequence of examples.

The ability to retrieve heavily-weighted features from the WM-Sketch confers several benefits. First, the sketch provides a classifier with low memory footprint that retains a degree of *model interpretability*. This is often practically important as understanding which features are most influential in making predictions is relevant to feature selection [76], model debugging, issues of fairness in ML systems [13], and human perceptions of model trustworthiness [55]. Second, the ability to retrieve heavily-weighted features enables the execution of a range of analytics workloads that can be formulated as classification problems over streaming data. In this paper, we demonstrate the effectiveness of the WM-Sketch in three such applications: (i) streaming data explanation [3, 51], (ii) detecting large relative differences between data streams (i.e., detecting relative deltoids) [16] and (iii) streaming identification of highly-correlated pairs of features via pointwise mutual information [22]. The WM-Sketch is able to perform these analyses while using far less memory than uncompressed classifiers.

The key intuition behind the WM-Sketch is that by randomly projecting the gradient updates to a linear classifier, we can incrementally maintain a compressed version of the true, high-dimensional model. By choosing this random projection appropriately, we can support efficient approximate recovery of the model weights. In particular, the WM-Sketch maintains a Count-Sketch projection [10] of the weight vector of the linear classifier. However, unlike Heavy Hitters sketches that simply increment or decrement counters, the WM-Sketch updates its state using online gradient descent [29]. Since these updates themselves depend on the current weight estimates, a careful analysis is needed to ensure that the estimated weights do not diverge from the true (uncompressed) model parameters over the course of multiple online updates.

We analyze the WM-Sketch both theoretically and empirically. Theoretically, we provide guarantees on the approximation error of these weight estimates, showing that it is possible to accurately recover large-magnitude weights using space sub-linear in the feature dimension. We describe an optimized variant, the Active-Set Weight-Median Sketch (AWM-Sketch) that outperforms alternative memory-constrained algorithms in experiments on benchmark datasets. For example, on the standard Reuters RCV1 binary classification benchmark, the AWM-Sketch recovers the most heavily-weighted features in the model with 4× better approximation error than a frequent-features baseline using the Space Saving algorithm [52] and 10× better than a naïve weight truncation baseline, while using the same amount of memory. Moreover, we demonstrate that the additional interpretability of the AWM-Sketch does not come at the cost of reduced classification accuracy: empirically, the AWM-Sketch in fact improves on the classification accuracy of feature hashing, which does not support weight recovery.

To summarize, we make the following contributions in this work:

- We introduce the Weight-Median Sketch, a new sketch for learning linear classifiers over data streams that supports approximate retrieval of the most heavily-weighted features.
- We provide a theoretical analysis that provides guarantees on the accuracy of the WM-Sketch estimates. In particular, we show that for feature dimension d and with success probability $1-\delta$, we can learn a compressed model of dimension $O\left(\epsilon^{-4}\log^3(d/\delta)\right)$ that supports approximate recovery of the optimal weight vector $\mathbf{w}_*$, where the absolute error of each weight is bounded above by $\epsilon\|\mathbf{w}_*\|_1$.
- We empirically demonstrate that the optimized AWM-Sketch outperforms several alternative methods in terms of memory-accuracy trade-offs across a range of real-world datasets.[1]

The full version of this paper with extended proofs of our theoretical results is available at https://arxiv.org/abs/1711.02305.

2 RELATED WORK

Heavy Hitters in Data Streams. Given a sequence of items, the heavy hitters problem is to return the set of all items whose frequency exceeds a specified fraction of the total number of items. Algorithms forfi nding frequent items include counter-based approaches [20, 37, 47, 52], quantile algorithms [26, 61], and sketch-based methods [10, 15]. Mirylenka et al. [54] develop streaming algorithms forfinding *conditional* heavy hitters, i.e. items that are frequent in the context of a separate "parent" item. Our proposed sketch builds on the Count-Sketch [10], which was originally introduced for identifying frequent items. In Sec. 4, we show how frequency estimation can in fact be related to the problem of estimating classifier weights.

Characterizing Changes in Data Streams. Cormode and Muthukrishnan [16] propose a Count-Min-based algorithm for identifying items whose frequencies change significantly, while Schweller et al. [57] propose the use of reversible hashes to avoid storing key information. In order to explain anomalous trafficfl ows, Brauckhoff et al. [7] use histogram-based detectors and association rules to detect large absolute differences. In our network monitoring application (Sec. 8), we focus instead on detecting large *relative* differences, a problem which has previously been found to be challenging [16].

Resource-Constrained and On-Device Learning. In contrast to *federated learning*, where the goal is to learn a global model on distributed data [38] or to enforce global regularization on a collection of local models [62], our focus is on memory-constrained learning on a single device without communication over a network. Gupta et al. [27] and Kumar et al. [39] perform inference with small-space classifiers on IoT devices, whereas we focus on online learning. Unlike budget kernel methods that aim the reduce the number of stored examplars [17, 19], our methods instead reduce the dimensionality of feature vectors. Our work also differs from *model compression* or *distillation* [2, 8, 31], which aims to imitate a large, expensive model using a smaller one with lower memory and computation costs—in our setting, the full uncompressed model is never instantiated and the compressed model is learned directly.

Sparsity-Inducing Regularization. ℓ_1-regularization is a standard technique for encouraging parameter sparsity in online learning [21, 40, 71]. In practice, it is difficult to *a priori* select an ℓ_1-regularization strength in order to satisfy a given sparsity budget.

[1] Our implementations of the WM-Sketch, AWM-Sketch and the baselines evaluated in our experiments are available at https://github.com/stanford-futuredata/wmsketch.

Here, we propose a different approach: wefi rstfi x a memory budget and then use the allocated space to approximate a classifier, with the property that our approximation will be better for sparse parameter vectors with small ℓ_1-norm.

Learning Compressed Classifiers. Feature hashing [60, 69] is a technique where the classifier is trained on features that have been hashed to afi xed-size table. This approach lowers memory usage by reducing the dimension of the feature space, but at the cost of model interpretability. Our sketch is closely related to this approach—we show that an appropriate choice of random projection enables the recovery of model weights. Calderbank et al. [9] describe *compressed learning*, where a classifier is trained on compressively-measured data. The authors focus on classification performance in the compressed domain and do not consider the problem of recovering weights in the original space.

3 BACKGROUND

In Section 3.1, we review the relevant material on random projections for dimensionality reduction. In Section 3.2, we describe *online learning*, which models learning on streaming data.

Conventions and Notation. The notation w_i denotes the ith element of the vector $\mathbf{w}$. The notation $[n]$ denotes the set $\{1, \ldots, n\}$. We write p-norms as $\|\mathbf{w}\|_p$, where the p-norm of $\mathbf{w} \in \mathbb{R}^d$ is defined as $\|\mathbf{w}\|_p := \left(\sum_{i=1}^d |w_i|^p\right)^{1/p}$. The infinity-norm $\|\mathbf{w}\|_\infty$ is defined as $\|\mathbf{w}\|_\infty := \max_i |w_i|$.

3.1 Dimensionality Reduction via Random Projection

Count-Sketch. The Count-Sketch [10] is a linear projection of a vector $\mathbf{x} \in \mathbb{R}^d$ that supports efficient approximate recovery of the entries of $\mathbf{x}$. The sketch of $\mathbf{x}$ can be built incrementally as entries are observed in a stream—for example, $\mathbf{x}$ can be a vector of counts that is updated as new items are observed.

For a given size k and depth s, the Count-Sketch algorithm maintains a collection of s hash tables, each with width k/s (Figure 2). Each index $i \in [d]$ is assigned a random bucket $h_j(i)$ in table j along with a random sign $\sigma_j(i)$. Increments to the ith entry are multiplied by $\sigma_j(i)$ and then added to the corresponding buckets $h_j(i)$. The estimator for the ith coordinate is the median of the values in the assigned buckets multiplied by the corresponding sign flips. Charikar et al. [10] showed the following recovery guarantee for this procedure:

Lemma1. *[10] Let $\mathbf{x}_{\text{cs}}$ be the Count-Sketch estimate of the vector $\mathbf{x}$. For any vector $\mathbf{x}$, with probability $1-\delta$, a Count-Sketch matrix with width $\Theta(1/\epsilon^2)$ and depth $\Theta(\log(d/\delta))$ satisfies*

$$\|\mathbf{x} - \mathbf{x}_{\text{cs}}\|_\infty \leq \epsilon \|\mathbf{x}\|_2.$$

In other words, point estimates of each entry of the vector $\mathbf{x}$ can be computed from its compressed form $\mathbf{x}_{\text{cs}}$. This enables accurate recovery of high-magnitude entries that comprise a large fraction of the norm $\|\mathbf{x}\|_2$.

Johnson-Lindenstrauss (JL) property. A random projection matrix is said to have the Johnson-Lindenstrauss (JL) property [33] if it preserves the norm of a vector with high probability:

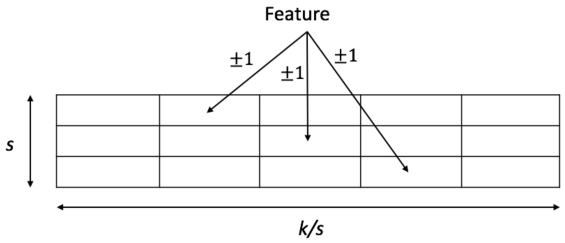

Figure 2: An illustration of the Count-Sketch of size k with depth s and width k/s. Each feature hashes to s locations, multiplied by a random ± 1 sign.

Definition 1. A random matrix $R \in \mathbb{R}^{k \times d}$ has the JL property with error ϵ and failure probability δ if for any given $\mathbf{x} \in \mathbb{R}^d$, we have with probability $1-\delta$:

$$\left| \|R\mathbf{x}\|_2 - \|\mathbf{x}\|_2 \right| \leq \epsilon \|\mathbf{x}\|_2.$$

The JL property holds for dense matrices with independent Gaussian or Bernoulli entries [1], and recent work has shown that it applies to certain sparse matrices as well [35]. Intuitively, JL matrices preserve the geometry of a set of points, and we leverage this key fact to ensure that we can still recover the original solution after projecting to low dimension.

3.2 Online Learning

The online learning framework deals with learning on a stream of examples, where the model is updated over a series of rounds $t = 1, 2, \ldots, T$. In each round, we update the model weights $\mathbf{w}_t$ via the following process: (1) receive an input example $(\mathbf{x}_t, y_t)$, (2) incur loss $L_t(\mathbf{w}_t) := \ell(\mathbf{w}_t, \mathbf{x}_t, y_t)$, where ℓ is a given loss function, and (3) update weights $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$. There are numerous algorithms for updating the model weights (e.g., [21, 30, 71]). In our algorithm, we use online gradient descent (OGD) [29; Chp. 3], which uses the following update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla L_t(\mathbf{w}_t),$$

where $\eta_t > 0$ is the learning rate at step t. OGD enjoys the advantages of simplicity and minimal space requirements: we only need to maintain a representation of the weight vector $\mathbf{w}_t$ and a global scalar learning rate.

4 PROBLEM STATEMENT

We focus on online learning for binary classification with linear models. We observe a stream of examples $(\mathbf{x}_t, y_t)$, where each $\mathbf{x}_t \in \mathbb{R}^d$ is a feature vector and each $y_t \in \{-1, +1\}$ is a binary label. A linear classifier parameterized by weights $\mathbf{w} \in \mathbb{R}^d$ makes predictions $\hat{y}$ by returning $+1$ for all inputs with non-negative inner product with $\mathbf{w}$, and -1 otherwise: $\hat{y} = \text{sign}\left(\mathbf{w}^T \mathbf{x}\right)$. The goal of learning is to select weights $\mathbf{w}$ that minimize the total loss $\sum_t L_t(\mathbf{w})$ on the observed data. In the following, we refer to $\mathbf{w}$ interchangeably as the *weights* and as the *classifier*.

Suppose we have observed T examples in the stream, and consider the classifier $\mathbf{w}_*$ that minimizes the loss over those T examples.

It may not be possible to precisely represent each entry[2] of the vector $\mathbf{w}_*$ within a memory budget that is much less than the cost of representing a general vector in $\mathbb{R}^d$. In particular, $\mathbf{w}_*$ may be a dense vector. Thus, it may not be possible to represent $\mathbf{w}_*$ in a memory-constrained setting, and in practical applications this is particularly problematic when the dimension d is large.

For afi xed memory budget B, our goal is to obtain a summary $\mathbf{z}$ that uses space at most B from which we are able to estimate the value of each entry of the optimal classifier $\mathbf{w}_*$. We formalize this problem as the Weight Estimation Problem, which we make precise in the following section. In addition to supporting weight estimation, a secondary goal is to be able to use the summary $\mathbf{z}$ to perform classification on data points $\mathbf{x}$ via some inference function f, i.e. $\hat{y} = f(\mathbf{z}, \mathbf{x})$. We would like to classify data points using the summary without too much additional error compared to $\mathbf{w}_*$.

4.1 The Weight Estimation Problem

In this section, we formalize the problem of estimating the weights of the optimal classifier $\mathbf{w}_*$ from a compact summary. To facilitate the presentation of this problem and to build intuition, we highlight the connection between our goal of weight estimation and previous work on the approximate recovery of frequency estimates from compressed count vectors. To this end, we formalize a general problem setup that subsumes both the approximate recovery of frequencies and the approximate recovery of weights in linear classifiers as special cases.

The ϵ-approximate frequency estimation problem can be defined as follows:

Definition 2. [14] (ϵ-Approximate Frequency Estimation) Given a sequence of T items, each drawn from the set $[d]$, let v_i denote the count of the number of times item i is seen over the stream. The ϵ-approximate frequency estimation problem is to return, for any $i \in [d]$, a value $\hat{v}_i$ such that $|\hat{v}_i - v_i| \leq \epsilon T$.

The frequency estimation problem commonly appears in the context offi nding heavy hitters—i.e., items whose frequencies exceed a given threshold ϕT. Given an algorithm that solves the ϵ-approximate frequency estimation problem, we can thenfi nd all heavy hitters (possibly with false positives) by returning all items with estimated frequency above $(\phi - \epsilon)T$.

We now define an analogous setup for online convex optimization problems that formalizes our goal of weight recovery from summarized classifiers:

Definition 3. ((ϵ, p)-Approximate Weight Estimation for Convex Functions) Given a sequence of T convex functions $L_t : \mathcal{X} \to \mathbb{R}$ over a convex domain $\mathcal{X} \subseteq \mathbb{R}^d$, let $\mathbf{w}_* \coloneqq \arg\min_{\mathbf{w}} \sum_{t=1}^{T} L_t(\mathbf{w})$. The (ϵ, p)-approximate weight estimation problem is to return, for any $i \in [d]$, a value $\hat{w}_i$ such that $|\hat{w}_i - (w_*)_i| \leq \epsilon \|\mathbf{w}_*\|_p$.

Note that frequency estimation (Definition 2) can be viewed as a special case of this problem. Set $L_t(\mathbf{w}) = -\mathbf{w}^T \mathbf{x}_t$, where $(x_t)_i = 1$ if item i is observed at time t and 0 otherwise (assume that only one item is observed at each t), define $\mathbf{x}_{1:T} \coloneqq \sum_{t=1}^{T} \mathbf{x}_t$, and let $\mathcal{X} = \{\mathbf{w} : \|\mathbf{w}\|_2 \leq \|\mathbf{x}_{1:T}\|_2\}$. Then $\mathbf{w}_* = \mathbf{x}_{1:T}$, and we note that $\|\mathbf{w}_*\|_1 = T$. Thus, the frequency estimation problem is an instance of the $(\epsilon, 1)$-approximate weight estimation problem.

Weight Estimation for Linear Classifiers. We now specialize to the case of online learning for linear classifiers. Define the losses L_t as:

$$L_t(\mathbf{w}) = \ell\left(y_t \mathbf{w}^T \mathbf{x}_t\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2, \tag{1}$$

where ℓ is a convex, differentiable function, $(\mathbf{x}_t, y_t)$ is the example observed at time t, and $\lambda > 0$ controls the strength of ℓ_2-regularization. The choice of ℓ defines the linear classification model to be used. For example, the logistic loss $\ell(\tau) = \log(1 + \exp(-\tau))$ defines logistic regression, and smoothed versions of the hinge loss $\ell(\tau) = \max\{0, 1 - \tau\}$ define close relatives of linear support vector machines.

To summarize, for each time step, we wish to maintain a *compact summary* $\mathbf{z}_t$ that allows us to estimate each weight in the optimal classifier $\mathbf{w}_*$ over all the examples seen so far in the stream. In the following sections, we describe a method for maintaining such a summary and provide theoretical guarantees on the accuracy of the recovered weights.

5 FINDING HEAVILY-WEIGHTED FEATURES

In this section, we describe our proposed method, the Weight-Median Sketch (WM-Sketch), along with a simple variant, the Active-Set Weight-Median Sketch (AWM-Sketch), that empirically improves on the basic WM-Sketch in both classification and recovery accuracy.

5.1 Weight-Median Sketch

The main data structure in the WM-Sketch is identical to that used in the Count-Sketch. The sketch is parameterized by size k, depth s, and width k/s. We initialize the sketch with a size-k array set to zero. For a given depth s, we view this array as being arranged in s rows, each of width k/s (assume that k is a multiple of s). We denote this array as $\mathbf{z}$, and equivalently view it as a vector in $\mathbb{R}^k$.

The high-level idea is that each row of the sketch is a compressed version of the model weight vector $\mathbf{w} \in \mathbb{R}^d$, where each index $i \in [d]$ is mapped to some assigned bucket $j \in [k/s]$. Since $k/s \ll d$, there will be many collisions between these weights; therefore, we maintain s rows—each with different assignments of features to buckets—in order to disambiguate weights.

Hashing Features to Buckets. In order to avoid explicitly storing the mapping from features to buckets, which would require space linear in d, we implement the mapping using hash functions as in the Count-Sketch. For each row $j \in [s]$, we maintain a pair of hash functions, $h_j : [d] \to [k/s]$ and $\sigma_j : [d] \to \{-1, +1\}$. Let the matrix $A \in \{-1, +1\}^{k \times d}$ denote the Count-Sketch projection implicitly represented by the hash functions h_j and σ_j, and let R be a scaled version of this projection, $R = \frac{1}{\sqrt{s}} A$. We use the projection R to compress feature vectors and update the sketch.

Updates. We update the sketch by performing gradient descent updates directly on the compressed classifier $\mathbf{z}$. We compute gradients on a "compressed" version $\hat{L}_t$ of the regularized loss L_t defined in Eq. 1:

$$\hat{L}_t(\mathbf{z}) = \ell\left(y_t \mathbf{z}^T R \mathbf{x}_t\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2.$$

[2]For example, storing each nonzero entry as a single-precisionfl oating point number.

Algorithm 1: Weight-Median (WM) Sketch

input: size k, depth s, loss function ℓ, ℓ_2-regularization parameter λ, learning rate schedule η_t
initialization
  $\mathbf{z} \leftarrow s \times k/s$ array of zeroes
  Sample R, a Count-Sketch matrix scaled by $\frac{1}{\sqrt{s}}$
  $t \leftarrow 0$
function `Update`($\mathbf{x}$, y)
  $\tau \leftarrow \mathbf{z}^T R\mathbf{x}$ ▷ Prediction for $\mathbf{x}$
  $\mathbf{z} \leftarrow (1 - \lambda\eta_t)\mathbf{z} - \eta_t y \nabla\ell(y\tau) R\mathbf{x}$
  $t \leftarrow t + 1$
function `Query`(i)
  return output of Count-Sketch retrieval on $\sqrt{s}\mathbf{z}$

This yields the following update to $\mathbf{z}$:

$$\hat{\Delta}_t := -\eta_t \nabla \hat{L}_t(\mathbf{z}) = -\eta_t \left(y_t \nabla\ell(y_t \mathbf{z}^T R\mathbf{x}_t) R\mathbf{x}_t + \lambda\mathbf{z} \right).$$

To build intuition, it is helpful to compare this update to the Count-Sketch update rule [10]. In the frequent items setting, the input $\mathbf{x}_t$ is a one-hot encoding for the item seen in that time step. The update to the Count-Sketch state $\mathbf{z}_{\text{cs}}$ is the following:

$$\tilde{\Delta}_t^{\text{cs}} = A\mathbf{x}_t,$$

where A is defined identically as above. Ignoring the regularization term, our update rule is simply the Count-Sketch update scaled by the constant $-\eta_t y_t s^{-1/2} \nabla\ell(y_t \mathbf{z}^T R\mathbf{x}_t)$. However, an important detail to note is that the Count-Sketch update is *independent* of the sketch state $\mathbf{z}_{\text{cs}}$, whereas the WM-Sketch update does depend on $\mathbf{z}$. This cyclical dependency between the state and state updates is the main challenge in our analysis of the WM-Sketch.

Queries. To obtain an estimate $\hat{w}_i$ of the ith weight, we return the median of the values $\{\sqrt{s}\sigma_j(i) z_{j,h_j(i)} : j \in [s]\}$. Save for the $\sqrt{s}$ factor, this is identical to the query procedure for the Count-Sketch.

We summarize the update and query procedures for the WM-Sketch in Algorithm 1. In the next section, we show how the sketch size k and depth s parameters can be chosen to satisfy an ϵ approximation guarantee with failure probability δ over the randomness in the sketch matrix.

Efficient Regularization. A naïve implementation of ℓ_2 regularization on $\mathbf{z}$ that scales each entry in $\mathbf{z}$ by $(1 - \eta_t\lambda)$ in each iteration incurs an update cost of $O(k + s \cdot \text{nnz}(\mathbf{x}))$. This masks the computational gains that can be realized when $\mathbf{x}$ is sparse. Here, we use a standard trick [58]: we maintain a global *scale* parameter α that scales the sketch values $\mathbf{z}$. Initially, $\alpha = 1$ and we update $\alpha \leftarrow (1 - \eta_t\lambda)\alpha$ to implement weight decay over the entire feature vector. Our weight estimates are therefore additionally scaled by α: $\hat{w}_i = \text{median}\left\{\sqrt{s}\alpha\sigma_j(i) z_{j,h_j(i)} : j \in [s]\right\}$. This optimization reduces the update cost from $O(k + s \cdot \text{nnz}(\mathbf{x}))$ to $O(s \cdot \text{nnz}(\mathbf{x}))$.

5.2 Active-Set Weight-Median Sketch

We now describe a simple, heuristic extension to the WM-Sketch that significantly improves the recovery accuracy of the sketch in practice. We refer to this variant as the Active-Set Weight-Median Sketch (AWM-Sketch).

Algorithm 2: Active-Set Weight-Median (AWM) Sketch

initialization
  $S \leftarrow \{\}$ ▷ Empty heap
  $\mathbf{z} \leftarrow s \times k/s$ array of zeroes
  Sample R, a Count-Sketch matrix scaled by $\frac{1}{\sqrt{s}}$
  $t \leftarrow 0$
function `Update`($\mathbf{x}$, y)
  $\mathbf{x}_s \leftarrow \{x_i : i \in S\}$ ▷ Features in heap
  $\mathbf{x}_{\text{wm}} \leftarrow \{x_i : i \notin S\}$ ▷ Features in sketch
  $\tau \leftarrow \sum_{i \in S} S[i] \cdot x_i + \mathbf{z}^T R\mathbf{x}_{\text{wm}}$ ▷ Prediction for $\mathbf{x}$
  $S \leftarrow (1 - \lambda\eta_t)S - \eta_t y \nabla\ell(y\tau)\mathbf{x}_s$ ▷ Heap update
  $\mathbf{z} \leftarrow (1 - \lambda\eta_t)\mathbf{z}$ ▷ Apply regularization
  for $i \notin S$ **do**
    ▷ Either update i in sketch or move to heap
    $\tilde{w} \leftarrow$ `Query`(i) $- \eta_t y x_i \nabla\ell(y\tau)$
    $i_{\text{min}} \leftarrow \arg\min_j(|S[j]|)$
    if $|\tilde{w}| > |S[i_{\text{min}}]|$ **then**
      Remove i_{min} from S
      Add i to S with weight $\tilde{w}$
      Update i_{min} in sketch with $S[i_{\text{min}}] -$ `Query`(i_{min})
    else
      Update i in sketch with $\eta_t y x_i \nabla\ell(y\tau)$
  $t \leftarrow t + 1$

To efficiently track the top elements across sketch updates, we can use a min-heap ordered by the absolute value of the estimated weights. This technique is also used alongside heavy-hitters sketches to identify the most frequent items in the stream [10]. In the basic WM-Sketch, the heap merely functions as a mechanism to passively maintain the heaviest weights. This baseline scheme can be improved by noting that weights that are already stored in the heap need not be tracked in the sketch; instead, the sketch can be updated lazily only when the weight is evicted from the heap. This heuristic has previously been used in the context of improving count estimates derived from a Count-Min Sketch [56]. The intuition here is the following: since we are already maintaining a heap of heavy items, we can utilize this structure to reduce error in the sketch as a result of collisions with heavy items.

The heap can be thought of as an "active set" of high-magnitude weights, while the sketch estimates the contribution of the tail of the weight vector. Since the weights in the heap are represented exactly, this active set heuristic should intuitively yield better estimates of the heavily-weighted features in the model.

As a general note, similar coarse-to-fine approximation schemes have been proposed in other online learning settings. A similar scheme for memory-constrained sparse linear regression was analyzed by Steinhardt and Duchi [64]. Their algorithm similarly uses a Count-Sketch for approximating weights, but in a different setting (K-sparse linear regression) and with a different update policy for the active set.

6 THEORETICAL ANALYSIS

We derive bounds on the recovery error achieved by the WM-Sketch for given settings of the size k and depth s. The main challenge in our analysis is that the updates to the sketch depend on gradient estimates which in turn depend on the state of the sketch. This

reflexive dependence makes it difficult to straightforwardly transplant the standard analysis for the Count-Sketch to our setting. Instead, we turn to ideas drawn from norm-preserving random projections and online convex optimization.

In this section, we begin with an analysis of recovery error in the batch setting, where we are given access to afi xed dataset of size T consisting of thefirst T examples observed in the stream and are allowed multiple passes over the data. Subsequently, we use this result to show guarantees in a restricted online case where we are only allowed a single pass through the data, but with the assumption that the order of the data is not chosen adversarially.

6.1 Batch Setting

First, we briefly outline the main ideas in our analysis. With high probability, we can sample a random projection to dimension $k \ll d$ that satisfies the JL norm preservation property (Definition 1). We use this property to show that for anyfi xed dataset of size T, optimizing a projected version of the objective yields a solution that is close to the projection of the minimizer of the original, high-dimensional objective. Since our specific construction of the JL projection is also a Count-Sketch projection, we can make use of existing error bounds for Count-Sketch estimates to bound the error of our recovered weight estimates.

Let $R \in \mathbb{R}^{k \times d}$ denote the scaled Count-Sketch matrix defined in Sec. 5.1. This is the hashing-based sparse JL projection proposed by Kane and Nelson [35]. We consider the following pair of objectives defined over the T observed examples—thefi rst defines the problem in the original space and the second defines the corresponding problem where the learner observes sketched examples $(R\mathbf{x}_t, y_t)$:

$$L(\mathbf{w}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(y_t \mathbf{w}^T \mathbf{x}_t\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2,$$

$$\hat{L}(\mathbf{z}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(y_t \mathbf{z}^T R\mathbf{x}_t\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2.$$

Suppose we optimized these objectives over $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{z} \in \mathbb{R}^k$ respectively to obtain solutions $\mathbf{w}_* = \arg\min_{\mathbf{w}} L(\mathbf{w})$ and $\mathbf{z}_* = \arg\min_{\mathbf{z}} \hat{L}(\mathbf{z})$. How then does $\mathbf{w}_*$ relate to $\mathbf{z}_*$ given our choice of sketching matrix R and regularization parameter λ? Intuitively, if we stored all the data observed up to time T and optimized $\mathbf{z}$ over this dataset, we should hope that the optimal solution $\mathbf{z}_*$ is close to $R\mathbf{w}_*$, the sketch of $\mathbf{w}_*$, in order to have any chance of recovering the largest weights of $\mathbf{w}_*$. We show that in this batch setting, $\|\mathbf{z}_* - R\mathbf{w}_*\|_2$ is indeed small; we then use this property to show element-wise error guarantees for the Count-Sketch recovery process. We now state our result for the batch setting:

Theorem1. *Let the loss function ℓ be β-strongly smooth[3] (w.r.t. $\|\cdot\|_2$) and $\max_t \|\mathbf{x}_t\|_1 = \gamma$. Forfi xed constants $C_1, C_2 > 0$, let:*

$$k = \left(C_1/\epsilon^4\right)\log^3(d/\delta)\max\left\{1, \beta^2\gamma^4/\lambda^2\right\},$$

$$s = \left(C_2/\epsilon^2\right)\log^2(d/\delta)\max\left\{1, \beta\gamma^2/\lambda\right\}.$$

[3] A function $f : \mathcal{X} \to \mathbb{R}$ is β-strongly smooth w.r.t. a norm $\|\cdot\|$ if f is everywhere differentiable and if for all $\mathbf{x}, \mathbf{y}$ we have:

$$f(\mathbf{y}) \le f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^T\nabla f(\mathbf{x}) + \frac{\beta}{2}\|\mathbf{y}-\mathbf{x}\|^2.$$

Let $\mathbf{w}_*$ be the minimizer of the original objective function $L(\mathbf{w})$ and $\mathbf{w}_{\text{est}}$ be the estimate of $\mathbf{w}_*$ returned by performing Count-Sketch recovery on the minimizer $\mathbf{z}_*$ of the projected objective function $\hat{L}(\mathbf{z})$. Then with probability $1-\delta$ over the choice of R,

$$\|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty \le \epsilon\|\mathbf{w}_*\|_1.$$

We note that for standard loss functions such as the logistic loss and the smoothed hinge loss, we have smoothness parameter $\beta = 1$. Moreover, we can assume that input vectors are normalized so that $\|\mathbf{x}_t\|_1 = 1$, and that typically $\lambda < 1$. Given these parameter choices, we can obtain simpler expressions for the sketch size k and sketch depth s:

$$k = O\left(\epsilon^{-4}\lambda^{-2}\log^3(d/\delta)\right),$$

$$s = O\left(\epsilon^{-2}\lambda^{-1}\log^2(d/\delta)\right).$$

We defer the full proof of the theorem to Appendix A.1. We now highlight some salient properties of this recovery result:

Sub-linear Dimensionality Dependence. Theorem 1 implies that we can achieve error bounded by $\epsilon\|\mathbf{w}_*\|_1$ with a sketch of size only polylogarithmic in the feature dimension d—this implies that memory-efficient learning and recovery is possible in the large-d regime that we are interested in. Importantly, the sketch size k is independent of the number of observed examples T—this is crucial since our applications involve learning over data streams of possibly unbounded length.

Update Time. Recall that the WM-Sketch can be updated in time $O(s \cdot \text{nnz}(\mathbf{x}))$ for a given input vector $\mathbf{x}$. Thus, the sketch supports an update time of $O(\epsilon^{-2}\lambda^{-1}\log^2(d/\delta) \cdot \text{nnz}(\mathbf{x}))$ in each iteration.

ℓ_2-Regularization. k and s scale inversely with the strength of ℓ_2 regularization: this is intuitive because additional regularization will shrink both $\mathbf{w}_*$ and $\mathbf{z}_*$ towards zero. We observe this inverse relationship between recovery error and ℓ_2 regularization in practice (see Figure 3).

Input Sparsity. The recovery error depends on the maximum ℓ_1-norm γ of the data points $\mathbf{x}_t$, and the bound is most optimistic when γ is small. Across all of the applications we consider in Sections 7 and 8, the data points are sparse with small ℓ_1-norm, and hence the bound is meaningful across a number of real-world settings.

Weight Sparsity. The per-parameter recovery error in Theorem 1 is bounded above by a multiple of the ℓ_1-norm of the optimal weights $\mathbf{w}_*$ for the uncompressed problem. This supports the intuition that sparse solutions with small ℓ_1-norm should be more easily recovered. In practice, we can augment the objective with an additional $\|\mathbf{w}\|_1$ (resp. $\|\mathbf{z}\|_1$) term to induce sparsity; this corresponds to elastic net-style composite ℓ_1/ℓ_2 regularization on the parameters of the model [79].

Comparison with Frequency Estimation. We can compare our guarantees for weight estimation in linear classifiers with existing guarantees for frequency estimation. The Count-Sketch requires $\Theta(\epsilon^{-2}\log(d/\delta))$ space and $\Theta(\log(d/\delta))$ update time to obtain frequency estimates $\mathbf{v}_{\text{cs}}$ with error $\|\mathbf{v} - \mathbf{v}_{\text{cs}}\|_\infty \le \epsilon\|\mathbf{v}\|_2$, where $\mathbf{v}$ is the true frequency vector (Lemma 1). The Count-Min Sketch uses $\Theta\left(\epsilon^{-1}\log(d/\delta)\right)$ space and $\Theta(\log(d/\delta))$ update time to obtain frequency estimates $\mathbf{v}_{\text{cm}}$ with error $\|\mathbf{v} - \mathbf{v}_{\text{cm}}\|_\infty \le \epsilon\|\mathbf{v}\|_1$ [15].

Thus, our analysis yields guarantees of a similar form to bounds for frequency estimation in this more general framework, but with somewhat worse polynomial dependence on $1/\epsilon$ and $\log(d/\delta)$, and additional $1/\epsilon$ dependence in the update time.

6.2 Online Setting

We now provide guarantees for WM-Sketch in the online setting. We make two small modifications to WM-Sketch for the convenience of analysis. First, we assume that the iterate $\mathbf{z}_t$ is projected onto a ℓ_2 ball of radius D at every step. Second, we also assume that we perform thefi nal Count-Sketch recovery on the average $\bar{\mathbf{z}} = \frac{1}{T}\sum_{i=1}^{T} \mathbf{z}_t$ of the weight vectors, instead of on the current iterate $\mathbf{z}_t$. While using this averaged sketch is useful for the analysis, maintaining a duplicate data structure in practice for the purpose of accumulating the average would double the space cost of our method. Therefore, in our implementation of the WM-Sketch, we simply maintain the current iterate $\mathbf{z}_t$. As we show in the next section this approach achieves good performance on real-world datasets, in particular when combined with the active set heuristic.

Our guarantee holds in expectation over uniformly random permutations of $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$. In other words, we achieve low recovery error on average over all orderings in which the T data points could have been presented. We believe this condition is necessary to avoid worst-case adversarial orderings of the data points—since the WM-Sketch update at any time step depends on the state of the sketch itself, adversarial orderings can potentially lead to high error accumulation.

Theorem2. *Let the loss function ℓ be β-strongly smooth (w.r.t. $\|\cdot\|_2$), and have its derivative bounded by H. Assume $\|\mathbf{x}_t\|_2 \leq 1, \max_t \|\mathbf{x}_t\|_1 = \gamma$, $\|\mathbf{w}_*\|_2 \leq D_2$ and $\|\mathbf{w}_*\|_1 \leq D_1$. Let G be a bound on the ℓ_2 norm of the gradient at any time step t, in our case $G \leq H(1+\epsilon\gamma) + \lambda D$. Forfi xed constants $C_1, C_2, C_3 > 0$, let:*

$$k = \left(C_1/\epsilon^4\right)\log^3(d/\delta)\max\left\{1, \beta^2\gamma^4/\lambda^2\right\},$$
$$s = \left(C_2/\epsilon^2\right)\log^2(d/\delta)\max\left\{1, \beta\gamma^2/\lambda\right\},$$
$$T \geq (C_3/\epsilon^4)\zeta\log^2(d/\delta)\max\{1, \beta\gamma^2/\lambda\},$$

where $\zeta = (1/\lambda^2)(D_2/\|w_*\|_1)^2(G + (1+\epsilon\gamma)H)^2$. Let $\mathbf{w}_*$ be the minimizer of the original objective function $L(\mathbf{w})$ and $\mathbf{w}_{\text{wm}}$ be the estimate $\mathbf{w}_*$ returned by the WM-Sketch algorithm with averaging and projection on the ℓ_2 ball with radius $D = (D_2 + \epsilon D_1)$. Then with probability $1-\delta$ over the choice of R,

$$\mathbb{E}[\|\mathbf{w}_* - \mathbf{w}_{\text{wm}}\|_\infty] \leq \epsilon\|\mathbf{w}_*\|_1,$$

where the expectation is taken with respect to uniformly sampling a permutation in which the samples are received.

Theorem 2 shows that in this restricted online setting, we achieve a bound with the same scaling of the sketch parameters k and s as the batch setting (Theorem 1). Again, we defer the full proof to Appendix A.2.

Intuitively, it seems reasonable to expect that we would need an "average case" ordering of the stream in order to obtain a similar recovery guarantee to the batch setting. An adversarial, worst-case ordering of the examples could be one where all the negatively-labeled examples arefi rst presented, followed by all the positively-labeled examples. In such a setting, it appears implausible that a single-pass online algorithm should be able to accurately estimate the weights obtained by a batch algorithm that is allowed multiple passes over the data.

Dataset	# Examples	# Features	Space (MB)
Reuters RCV1	6.77×10^5	4.72×10^4	0.4
Malicious URLs	2.40×10^6	3.23×10^6	25.8
KDD Cup Algebra	8.41×10^6	2.02×10^7	161.8
Senate/House Spend.	4.08×10^7	5.14×10^5	4.2
Packet Trace	1.86×10^7	1.26×10^5	1.0
Newswire	2.06×10^9	4.69×10^7	375.2

Table 1: Summary of benchmark datasets with the space cost of representing full weight vectors and feature identifiers using 32-bit values. Thefi rst set of three consists of standard binary classification datasets used in Sec. 7; the second set consists of datasets specific to the applications in Sec. 8.

7 EMPIRICAL EVALUATION

In this section, we evaluate the Weight-Median Sketch on three standard binary classification datasets. Our goal here is to compare the WM-Sketch and AWM-Sketch against alternative limited-memory methods in terms of (1) recovery error in the estimated top-K weights, (2) classification error rate, and (3) runtime performance. In the next section, we explore specific applications of the WM-Sketch in stream processing tasks.

7.1 Datasets and Experimental Setup

We evaluated our proposed sketches on several standard benchmark datasets as well as in the context of specific streaming applications. Table 1 lists summary statistics for these datasets.

Classification Datasets. We evaluate the recovery error on ℓ_2-regularized online logistic regression trained on three standard binary classification datasets: Reuters RCV1 [43], malicious URL identification [46], and the `Algebra` dataset from the KDD Cup 2010 large-scale data mining competition [63, 73]. We use the standard training split for each dataset except for the RCV1 dataset, where we use the larger "test" split as is common in experimental evaluations using this dataset [25].

For each dataset, we make a single pass through the set of examples. Across all our experiments, we use an initial learning rate $\eta_0 = 0.1$ and $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. We used the following set of space constraints: 2KB, 4KB, 8KB, 16KB and 32KB. For each setting of the space budget and for each method, we evaluate a range of configurations compatible with that space constraint; for example, for evaluating the WM-Sketch, this corresponds to varying the space allocated to the heap and the sketch, as well as trading off between the sketch depth s and the width k/s. For each setting, we run 10 independent trials with distinct random seeds; our plots show medians and the range between the worst and best run.

Memory Cost Model. In our experiments, we control for memory usage and configure each method to satisfy the given space constraints using the following cost model: we charge 4B of memory utilization for each feature identifier, feature weight, and auxiliary weight (e.g., random keys in Algorithm 4 or counts in the Space Saving baseline) used. For example, a simple truncation instance

	WM-Sketch			AWM-Sketch		
Budget (KB)	$\|S\|$	width	depth	$\|S\|$	width	depth
2	128	128	2	128	256	1
4	256	256	2	256	512	1
8	128	128	14	512	1024	1
16	128	128	30	1024	2048	1
32	128	256	31	2048	4096	1

Table 2: Sketch configurations with minimal ℓ_2 recovery error on RCV1 dataset ($|S|$ denotes heap capacity).

(Algorithm 3 in the Appendix) with 128 entries uses 128 identifiers and 128 weights, corresponding to a memory cost of 1024B.

7.2 Baseline Methods

Here, we describe the baseline algorithms that we use in our evaluation.

Simple Truncation. Given a budget of K weights, a natural baseline method is to simply truncate $\mathbf{w}$ after each update to the K entries with highest absolute value, setting all other entries to zero. The simple truncation baseline is similar to the truncated Perceptron algorithm proposed by Hoi et al. [32]. We give a pseudocode description in Appendix B.

Probabilistic Truncation. A problem with the simple truncation method is that it may end up "stuck" with a bad set of weights: a "good" index that would have been included in the top-K set by the unconstrained classifier may fail to be included in the feature set under Algorithm 3 if its gradient updates are insufficiently large relative to the smallest weight in the set; this results in the weight being repeatedly zeroed-out in each iteration. To remedy this problem, we can instead adopt a *randomized* approach where indices are accepted into the K-sparse set with probability proportional to the magnitude of their weights. Therefore, even if some feature has small but nonzero weight after an update, there is still a positive probability that it is accepted into the feature set. This "probabilistic truncation" algorithm is inspired by weighted reservoir sampling [23]. We give the pseudocode in Appendix B.

Count-Min Frequent Features. The Count-Min sketch [15] is a commonly-used method forfi nding frequent items in data streams. This baseline uses a Count-Min sketch to identify the K most frequently-occurring features; the weights for these frequent features are maintained, while the remaining weights are set to 0.

Space Saving Frequent Features. This method is identical to the previous approach except for the use of the Space Saving algorithm [52] in place of the Count-Min sketch for frequent item estimation. The Space Saving algorithm has previously been found to outperform Count-Min in insertion-only settings such as ours [14].

Feature Hashing. In feature hashing, input vectors are mapped to lower dimension by adding each input feature, multiplied by a random sign, to a randomly-assigned index in the compressed vector [60, 69]. This basic scheme is essentially equivalent to computing a Count-Sketch of depth 1 on the input. As with the WM-Sketch, the model weights are learned in this compressed space; however, due to hash collisions, it is in general not possible to recover accurate estimates of model weights from the compressed weight vector

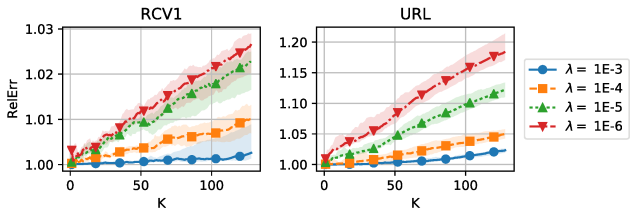

Figure 3: Relative ℓ_2 error of top-K AWM-Sketch estimates with varying regularization parameter λ on RCV1 and URL datasets under 8KB memory budget.

in feature hashing. While multiple hashing is suggested by Weinberger et al. [69], the proposed method does not support efficient weight recovery.

7.3 Recovery Error Comparison

We measure the accuracy to which our methods are able to recover the top-K weights in the model using the following relative ℓ_2 error metric:

$$\text{RelErr}(\mathbf{w}^K, \mathbf{w}_*) = \|\mathbf{w}^K - \mathbf{w}_*\|_2 \,/\, \|\mathbf{w}_*^K - \mathbf{w}_*\|_2,$$

where $\mathbf{w}^K$ is the K-sparse vector representing the top-K weights returned by a given method, $\mathbf{w}_*$ is the weight vector obtained by the uncompressed model, and $\mathbf{w}_*^K$ is the K-sparse vector representing the true top-K weights in $\mathbf{w}_*$. The relative error metric is therefore bounded below by 1 and quantifies the relative suboptimality of the estimated top-K weights. The best configurations for the WM- and AWM-Sketch on RCV1 are listed in Table 2; the optimal configurations for the remaining datasets are similar.

We compare our methods across datasets (Fig. 4) and across memory constraints on a single dataset (Fig. 5). For clarity, we omit the Count-Min Frequent Features baseline since we found that the Space Saving baseline achieved consistently better performance. We found that the AWM-Sketch consistently achieved lower recovery error than alternative methods on our benchmark datasets. The Space Saving baseline is competitive on RCV1 but underperforms the simple Probabilistic Truncation baseline on URL: this demonstrates that tracking frequent features can be effective if frequently-occurring features are also highly discriminative, but this property does not hold across all datasets. Standard feature hashing achieves poor recovery error since colliding features cannot be disambiguated.

In Fig. 3, we compare recovery error on RCV1 across different settings of λ. Higher ℓ_2-regularization results in less recovery error since both the true weights and the sketched weights are closer to 0; however, λ settings that are too high can result in increased classification error.

7.4 Classification Error Rate

We evaluated the classification performance of our models by measuring their online error rate [5]: for each observed pair $(\mathbf{x}_t, y_t)$, we record whether the prediction $\hat{y}_t$ (made without observing y_t) is correct before updating the model. The error rate is defined as the cumulative number of mistakes made divided by the number of iterations. Our results here are summarized in Fig. 6. For each dataset, we used the value of λ that achieved the lowest error rate across all our memory-limited methods. For each method and budget, we

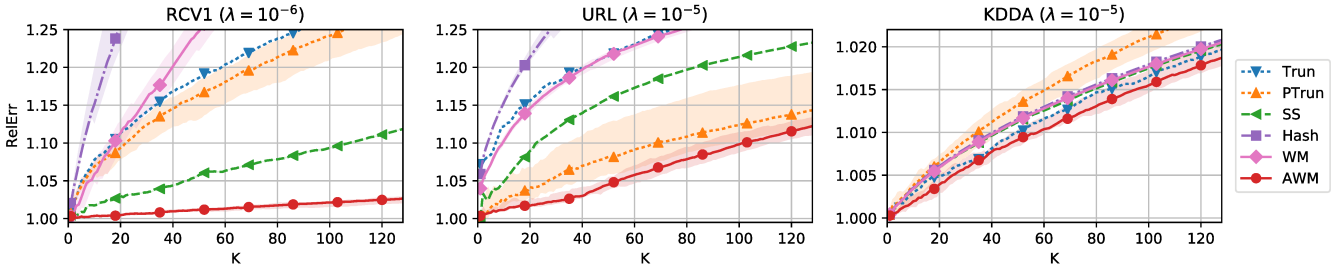

Figure 4: Relative ℓ_2 error of estimated top-K weights vs. true top-K weights for ℓ_2-regularized logistic regression under 8KB memory budget. Shaded area indicates range of errors observed over 10 trials. The AWM-Sketch achieves lower recovery error across all three datasets.

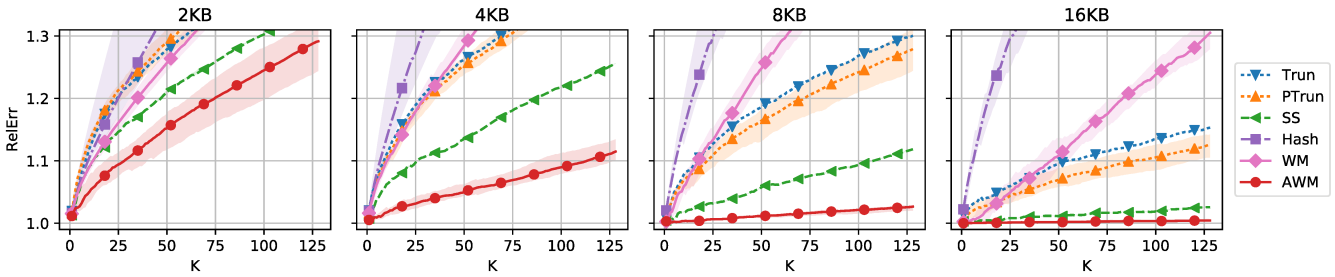

Figure 5: Relative ℓ_2 error of estimated top-K weights on RCV1 dataset under different memory budgets ($\lambda = 10^{-6}$). Shaded area indicates range of errors observed over 10 trials. The recovery quality of the AWM-Sketch quickly improves with more allocated space.

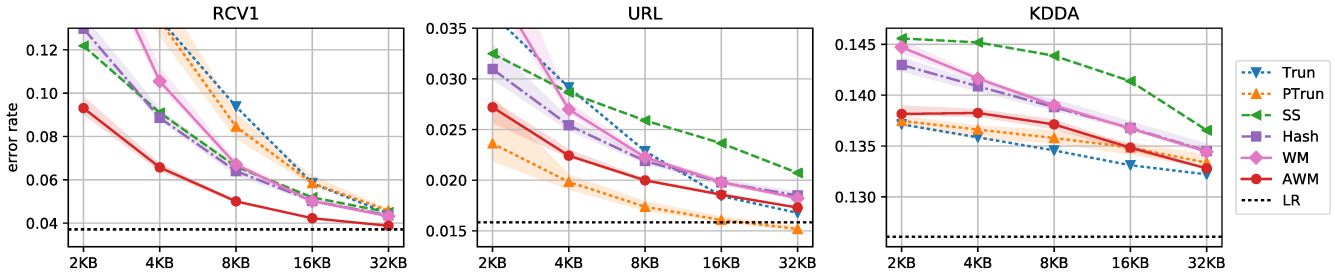

Figure 6: Online classification error rates with ℓ_2-regularized logistic regression under different memory budgets (Trun = Simple Truncation, PTrun = Probabilistic Truncation, SS = Space Saving Frequent, Hash = Feature Hashing, LR = Logistic Regression without memory constraints). The AWM-Sketch consistently achieves better classification accuracy than methods that track frequent features.

chose the configuration that achieved the lowest error rate. For the WM-Sketch, this corresponded to a width of 2^7 or 2^8 with depth scaling proportionally with the memory budget; for the AWM-Sketch, the configuration that uniformly performed best allocated half the space to the active set and the remainder to a depth-1 sketch.

We found that across all tested memory constraints, the AWM-Sketch consistently achieved lower error rate than heavy-hitter-based methods. Surprisingly, the AWM-Sketch outperformed feature hashing by a small but consistent margin: 0.5–3.7% on RCV1, 0.1–0.4% on URL, and 0.2–0.5% on KDDA, with larger gains seen at smaller memory budgets. This suggests that the AWM-Sketch benefits from the precise representation of the largest, most-influential weights in the model, and that these gains are sufficient to offset the increased collision rate due to the smaller hash table. The Space Saving baseline exhibited inconsistent performance across the three datasets, demonstrating that tracking the most frequent features is an unreliable heuristic: features that occur frequently are not necessarily the most predictive. We note that higher values of the regularization parameter λ correspond to greater penalization of rarely-occurring features; therefore, we would expect the Space Saving baseline to better approximate the performance of the unconstrained classifier as λ increases.

7.5 Runtime Performance

We evaluated runtime performance relative to a memory unconstrained logistic regression model using the same configurations as those chosen to minimize ℓ_2 recovery error (Table 2). In all our timing experiments, we ran our implementations of the baseline methods, the WM-Sketch, and the AWM-Sketch on Intel Xeon E5-2690 v4 processor with 35MB cache using a single core. The

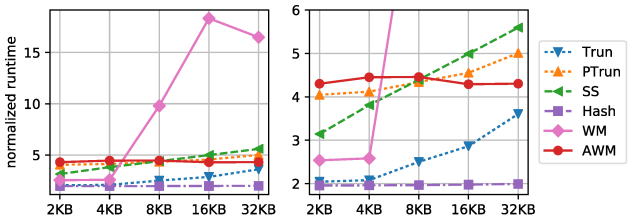

Figure 7: Normalized runtime of each method vs. memory-unconstrained logistic regression on RCV1 using configurations that minimize recovery error (see Table 2). The right panel is a zoomed-in view of the left panel.

memory-unconstrained logistic regression weights were stored using a 32-bitfl oating point array of size equal to the dimensionality of the feature space, with the highest-weighted features tracked using a heap of size $K = 128$; reads and writes to the weight vector therefore required single array accesses. The remaining methods tracked heavy weights alongside 32-bit feature identifiers using a heap sized according to the corresponding configuration.

In our experiments, the fastest method was feature hashing, with about a 2× overhead over the baseline. This overhead was due to the additional hashing step needed for each read and write to a feature index. The AWM-Sketch incurred an additional 2× overhead over feature hashing due to more frequent heap maintenance operations.

8 APPLICATIONS

We now show that a variety of tasks in stream processing can be framed as memory-constrained classification. The unifying theme between these applications is that classification is a useful abstraction whenever the use case calls for *discriminating* between streams or between subpopulations of a stream. These distinct classes can be identified by partitioning a single stream into quantiles (Sec. 8.1), comparing separate streams (Sec. 8.2), or even by generating *synthetic* examples to be distinguished from real samples (Sec. 8.3).

8.1 Streaming Explanation

In data analysis workflows, it is often necessary to identify characteristic attributes that are particularly indicative of a given subset of data [51]. For example, in order to diagnose the cause of anomalous readings in a sensor network, it is helpful to identify common features of the outlier points such as geographical location or time of day. This use case has motivated the development of methods for finding common properties of outliers found in aggregation queries [70] and in data streams [3].

This task can be framed as a classification problem: assign positive labels to the outliers and negative labels to the inliers, then train a classifier to discriminate between the two classes. The identification of characteristic attributes is then reduced to the problem of identifying heavily-weighted features in the trained model. In order to identify indicative *conjunctions* of attributes, we can simply augment the feature space to include arbitrary combinations of singleton features.

The *relative risk* or *risk ratio* $r_x = p(y = 1 \mid x = 1)/p(y = 1 \mid x = 0)$ is a statistical measure of the relative occurrence of the positive label $y = 1$ when the feature x is active versus when it is inactive. In the context of stream processing, the relative risk has been used

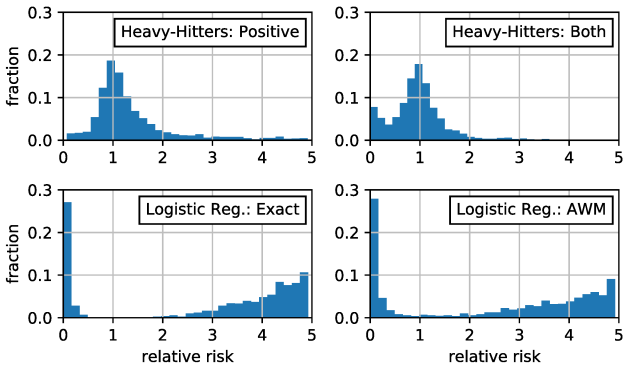

Figure 8: Distribution of relative risks among top-2048 features retrieved by each method. *Top Row*: Heavy-Hitters. *Bottom Row*: Classifier-based methods.

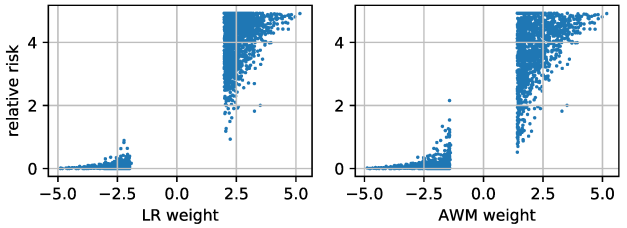

Figure 9: Correlation between top-2048 feature weights and relative risk. *Left*: Memory-unconstrained logistic regression (Pearson correlation 0.95). *Right*: AWM-Sketch (Pearson correlation 0.91).

to quantify the degree to which a particular attribute or attribute combination is indicative of a data point being an outlier relative to the overall population [3]. Here, we are interested in comparing our classifier-based approach to identifying high-risk features against the approach used in MacroBase [3], an existing system for explaining outliers over streams, that identifies candidate attributes using a variant of the Space Saving heavy-hitters algorithm.

Experimental Setup. We used a publicly-available dataset of itemized disbursements by candidates in U.S. House and Senate races from 2010–2016.[4] The outlier points were set to be the set of disbursements in the top-20% by dollar amount. For each row of the data, we generated a sequence of 1-sparse feature vectors[5] corresponding to the observed attributes. We set a space budget of 32KB for the AWM-Sketch.

Results. Our results are summarized in Figs. 8 and 9. The former empirically demonstrates that the heuristic offi ltering features on the basis of frequency can be suboptimal for afi xed memory budget. This is due to features that are frequent in both the inlier and outlier classes: it is wasteful to maintain counts for these items since they have low relative risk. In Fig. 8, the top row shows the distribution of relative risks among the most frequent items within the positive class (left) and across both classes (right). In contrast, our

[4]FEC candidate disbursements data: http://classic.fec.gov/data/CandidateDisbursement.do

[5]We can also generate a single feature vector per row (with sparsity greater than 1), but the learned weights would then correlate more weakly with the relative risk. This is due to the effect of correlations between features.

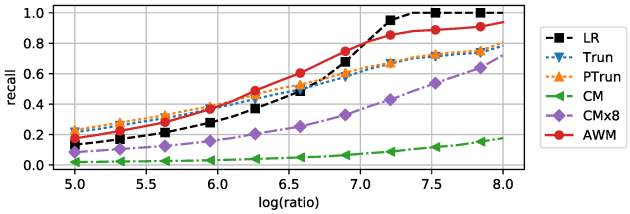

Figure 10: Recall of IP addresses with relative occurrence ratio above the given threshold with 32KB of space. `LR` denotes recall by full memory-unconstrained logistic regressor. `CMx8` denotes Count-Min baseline with 8× memory usage.

classifier-based approaches use the allocated space more efficiently by identifying features at the extremes of the relative risk scale.

In Fig. 9, we show that the learned classifier weights are strongly correlated with the relative risk values estimated from true counts. Indeed, logistic regression weights can be interpreted in terms of log odds ratios, a related quantity to relative risk. These results show that the AWM-Sketch is a superiorfi lter compared to heavy hitters approaches for identifying high-risk features.

8.2 Network Monitoring

IP network monitoring is one of the primary application domains for sketches and other small-space summary methods [4, 68, 74]. Here, we focus on the problem offi nding packet-level features (for instance, source/destination IP addresses and prefixes, port numbers, network protocols, and header or payload characteristics) that differ significantly in relative frequency between a pair of network links.

This problem of identifying significant relative differences—also known as *relative deltoids*—was studied by Cormode and Muthukrishnan [16]. Concretely, the problem is to estimate—for each item i—ratios $\phi(i) = n_1(i)/n_2(i)$ (where n_1, n_2 denote occurrence counts in each stream) and to identify those items i for which this ratio, or its reciprocal, is large. Here, we are interested in identifying differences between traffic streams that are observed *concurrently*; in contrast, the empirical evaluation in [16] focused on comparisons between different time periods.

Experimental Setup. We used a subset of an anonymized, publicly-available passive traffic trace dataset recorded at a peering link for a large ISP [66]. The positive class was the stream of outbound source IP addresses and the negative class was the stream of inbound destination IP addresses. We compared against several baseline methods, including ratio estimation using a pair of Count-Min sketches (as in [16]). For each method we retrieved the top-2048 features (i.e., IP addresses in this case) and computed the recall against the set of features above the given ratio threshold, where the reference ratios were computed using exact counts.

Results. We found that the AWM-Sketch performed comparably to the memory-unconstrained logistic regression baseline on this benchmark. We significantly outperformed the paired Count-Min baseline by a factor of over 4× in recall while using the same memory budget, as well as a paired CM baseline that was allocated 8x the memory budget. These results indicate that linear classifiers can be used effectively to identify relative deltoids over pairs of data streams.

Pair	PMI	Est.	Pair	PMI
prime minister	6.339	7.609	, the	0.044
los angeles	7.197	7.047	the ,	-0.082
http /	6.734	7.001	the of	0.611
human rights	6.079	6.721	the .	0.057

Table 3: *Left:* Top recovered pairs with PMI computed from true counts and PMI estimated from model weights (2^{16} bins, 1.4MB total memory). *Right:* Most common pairs in corpus.

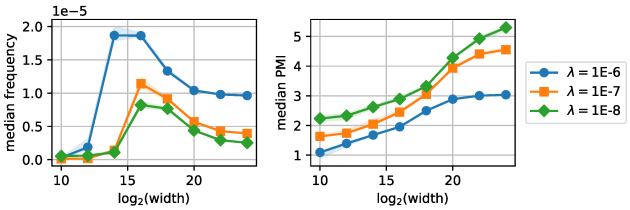

Figure 11: Median frequencies and exact PMIs of retrieved pairs with AWM-Sketch estimation. Lower λ and higher bin counts favor less frequent pairs.

8.3 Streaming Pointwise Mutual Information

Pointwise mutual information (PMI), a measure of the statistical correlation between a pair of events, is defined as:

$$\text{PMI}(x, y) = \log \frac{p(x, y)}{p(x)p(y)}.$$

Intuitively, positive values of the PMI indicate events that are positively correlated, negative values indicate events that are negatively correlated, and a PMI of 0 indicates uncorrelated events.

In natural language processing, PMI is a frequently-used measure of word association [65]. Traditionally, the PMI is estimated using empirical counts of unigrams and bigrams obtained from a text corpus. The key problem with this approach is that the number of bigrams in standard natural language corpora can grow very large; for example, we found ~47M unique co-occurring pairs of tokens in a small subset of a standard newswire corpus. This combinatorial growth in the feature dimension is further amplified when considering higher-order generalizations of PMI.

More generally, streaming PMI estimation can be used to detect pairs of events whose occurrences are strongly correlated. For example, we can consider a streaming log monitoring use case where correlated events are potentially indicative of cascading failures or trigger events resulting in exceptional behavior in the system. Therefore, we expect that the techniques developed here should be useful beyond standard NLP applications.

Sparse Online PMI Estimation. Streaming PMI estimation using approximate counting has previously been studied [22]; however, this approach has the drawback that memory usage still scales linearly with the number of observed bigrams. Here, we explore streaming PMI estimation from a different perspective: we pose a binary classification problem over the space of bigrams with the property that the model weights asymptotically converge to an estimate of the PMI.[6]

[6]This classification formulation is used in the popular `word2vec` skip-gram method for learning word embeddings [53]; the connection to PMI approximation wasfirst observed by Levy et al. [42].

The classification problem is set up as follows: in each iteration t, with probability 0.5 sample a bigram (u, v) from the bigram distribution $p(u, v)$ and set $y_t = +1$; with probability 0.5 sample (u, v) from the unigram product distribution $p(u)p(v)$ and set $y_t = -1$. The input $\mathbf{x}_t$ is the 1-sparse vector where the index corresponding to (u, v) is set to 1. We train a logistic regression model to discriminate between the *true* and *synthetic* samples. If $\lambda = 0$, the model asymptotically converges to the distribution $\hat{p}(y = 1 \mid (u, v)) = f(w_{uv}) = p(u, v)/\left(p(u, v) + p(u)p(v)\right)$ for all pairs (u, v), where f is the logistic function. It follows that $w_{uv} = \log(p(u, v)/p(u)p(v))$, which is exactly the PMI of (u, v). If $\lambda > 0$, we obtain an estimate that is biased, but with reduced variance in the estimates for rare bigrams.

Experimental Setup. We train on a subset of a standard newswire corpus [11]; the subset contains 77.7M tokens, 605K unique unigrams and 47M unique bigrams over a sliding window of size 6. In our implementation, we approximate sampling from the unigram distribution by sampling from a reservoir sample of tokens [34, 48]. We estimated weights using the AWM-Sketch with heap size 1024 and depth 1; the reservoir size wasfi xed at 4000. We make a single pass through the dataset and generate 5 negative samples for every true sample. Strings werefi rst hashed to 32-bit values using `MurmurHash3`;[7] these identifiers were hashed again to obtain sketch bucket indices.

Results. For width settings up to 2^{16}, our implementation's total memory usage was at most 1.4MB. In this regime, memory usage was dominated by the storage of strings in the heap and the unigram reservoir. For comparison, the standard approach to PMI estimation requires 188MB of space to store exact 32-bit counts for all bigrams, excluding the space required for storing strings or the token indices corresponding to each count. In Table 3, we show sample pairs retrieved by our method; the PMI values estimated from exact counts are well-estimated by the classifier weights. In Fig. 11, we show that at small widths, the high collision rate results in the retrieval of noisy, low-PMI pairs; as the width increases, we retrieve higher-PMI pairs which typically occur with lower frequency. Further, regularization helps discard low-frequency pairs but can result in the model missing out on high-PMI but less-frequent pairs.

9 DISCUSSION

Active Set vs. Multiple Hashing. In the basic WM-Sketch, multiple hashing is needed in order to disambiguate features that collide in a heavy bucket; we should expect that features with truly high weight should correspond to large values in the majority of buckets that they hash to. The active set approach uses a different mechanism for disambiguation. Suppose that all the features that hash to a heavy bucket are added to the active set; we should expect that the weights for those features that were erroneously added will eventually decay (due to ℓ_2-regularization) to the point that they are evicted from the active set. Simultaneously, the truly high-weight features are retained in the active set. The AWM-Sketch can therefore be interpreted as a variant of feature hashing where the highest-weighted features are not hashed.

[7] https://github.com/aappleby/smhasher/wiki/MurmurHash3

The Cost of Interpretability. A surprisingfi nding in our evaluation on standard binary classification datasets was that the AWM-Sketch consistently improved on the classification accuracy of feature hashing. We hypothesize that the observed gains are due to reduced collisions with heavily-weighted features. Notably, we are able to improve model interpretability by identifying important features without sacrificing any classification accuracy.

Per-Feature Learning Rates. In previous work on online learning applications, practitioners have found that the per-feature learning rates can significantly improve classification performance [50]. An open question is whether variable learning rate across features is worth the associated memory cost in the streaming setting.

Multiclass Classification. The WM-Sketch be extended to the multiclass setting using the following simple extension. Given M output classes, maintain M copies of the WM-Sketch. In order to predict the output, we evaluate the output on each copy and return the maximum. For large M, for instance in language modeling applications, this procedure can be computationally expensive since update time scales linearly with M. In this regime, we can apply noise contrastive estimation [28]—a standard reduction to binary classification—to learn the model parameters.

10 CONCLUSIONS

In this paper, we introduced the Weight-Median Sketch for the problem of identifying heavily-weighted features in linear classifiers over streaming data. We showed theoretical guarantees for our method, drawing on techniques from online learning and norm-preserving random projections. In our empirical evaluation, we showed that the active set extension to the basic WM-Sketch achieves superior weight recovery and competitive classification error compared to baseline methods across several standard binary classification benchmarks. Finally, we explored promising applications of our methods by framing existing stream processing tasks as classification problems. We believe this machine learning perspective on sketch-based stream processing may prove to be a fruitful direction for future research in advanced streaming analytics.

ACKNOWLEDGMENTS

We thank Daniel Kang, Sahaana Suri, Pratiksha Thaker, and the anonymous reviewers for their feedback on earlier drafts of this work. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Google, Intel, Microsoft, Teradata, and VMware—as well as industrial gifts and support from Toyota Research Institute, Juniper Networks, Keysight Technologies, Hitachi, Facebook, Northrop Grumman, and NetApp. The authors were also supported by DARPA under No. FA8750-17-2-0095 (D3M), ONR Award N00014-17-1-2562, NSF Award CCF-1704417, and a Sloan Research Fellowship.

REFERENCES

[1] Dimitris Achlioptas. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences* 66, 4 (2003), 671–687.
[2] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in neural information processing systems*. 2654–2662.
[3] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 541–556.

[4] Nagender Bandi, Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2007. Fast data stream algorithms using associative memories. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 247–256.

[5] Avrim Blum, Adam Kalai, and John Langford. 1999. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*. ACM, 203–208.

[6] Oscar Boykin, Sam Ritchie, Ian O'Connell, and Jimmy Lin. 2014. Summingbird: A framework for integrating batch and online mapreduce computations. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1441–1451.

[7] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner, and Kavè Salamatian. 2012. Anomaly extraction in backbone networks using association rules. *IEEE/ACM Transactions on Networking (TON)* 20, 6 (2012), 1788–1799.

[8] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 535–541.

[9] Robert Calderbank, Sina Jafarpour, and Robert Schapire. [n. d.]. Compressed Learning: Universal Sparse Dimensionality Reduction and Learning in the Measurement Domain. ([n. d.]).

[10] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. *Automata, languages and programming* (2002), 784–784.

[11] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005* (2013).

[12] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. 2004. Approximate aggregation techniques for sensor databases. In *Data Engineering, 2004. Proceedings. 20th International Conference on*. IEEE, 449–460.

[13] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. 2017. Algorithmic decision making and the cost of fairness. *arXiv preprint arXiv:1701.08230* (2017).

[14] Graham Cormode and Marios Hadjieleftheriou. 2008. Finding frequent items in data streams. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1530–1541.

[15] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[16] Graham Cormode and S Muthukrishnan. 2005. What's new: Finding significant differences in network data streams. *IEEE/ACM Transactions on Networking (TON)* 13, 6 (2005), 1219–1232.

[17] Koby Crammer, Jaz Kandola, and Yoram Singer. 2004. Online classification on a budget. In *Advances in neural information processing systems*. 225–232.

[18] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. 2012. Issues and future directions in traffic classification. *IEEE network* 26, 1 (2012).

[19] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2006. The Forgetron: A kernel-based perceptron on afi xed budget. In *Advances in neural information processing systems*. 259–266.

[20] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*. Springer, 348–360.

[21] John Duchi and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research* 10, Dec (2009), 2899–2934.

[22] Benjamin V Durme and Ashwin Lall. 2009. Streaming pointwise mutual information. In *Advances in Neural Information Processing Systems*. 1892–1900.

[23] Pavlos S Efraimidis and Paul G Spirakis. 2006. Weighted random sampling with a reservoir. *Inform. Process. Lett.* 97, 5 (2006), 181–185.

[24] Philippe Flajolet. 1985. Approximate counting: a detailed analysis. *BIT Numerical Mathematics* 25, 1 (1985), 113–134.

[25] Daniel Golovin, D Sculley, Brendan McMahan, and Michael Young. 2013. Large-scale learning with less ram via randomization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 325–333.

[26] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In *ACM SIGMOD Record*, Vol. 30. ACM, 58–66.

[27] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. In *International Conference on Machine Learning*. 1331–1340.

[28] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 297–304.

[29] Elad Hazan et al. 2016. Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2, 3-4 (2016), 157–325.

[30] Elad Hazan, Amit Agarwal, and Satyen Kale. 2007. Logarithmic regret algorithms for online convex optimization. *Machine Learning* 69, 2 (2007), 169–192.

[31] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[32] Steven CH Hoi, Jialei Wang, Peilin Zhao, and Rong Jin. 2012. Online feature selection for mining big data. In *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications*. ACM, 93–100.

[33] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics* 26, 189-206 (1984), 1.

[34] Nobuhiro Kaji and Hayato Kobayashi. 2017. Incremental skip-gram model with negative sampling. *arXiv preprint arXiv:1704.03956* (2017).

[35] Daniel M Kane and Jelani Nelson. 2014. Sparser Johnson-Lindenstrauss transforms. *Journal of the ACM (JACM)* 61, 1 (2014), 4.

[36] Ashish Kapoor, Simon Baker, Sumit Basu, and Eric Horvitz. 2012. Memory constrained face recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2539–2546.

[37] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. 2003. A simple algorithm forfi nding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)* 28, 1 (2003), 51–55.

[38] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).

[39] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In *International Conference on Machine Learning*. 1935–1944.

[40] John Langford, Lihong Li, and Tong Zhang. 2009. Sparse online learning via truncated gradient. *Journal of Machine Learning Research* 10, Mar (2009), 777–801.

[41] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. 2016. Heavy hitters via cluster-preserving clustering. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE, 61–70.

[42] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[43] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of machine learning research* 5, Apr (2004), 361–397.

[44] Brent Longstaff, Sasank Reddy, and Deborah Estrin. 2010. Improving activity classification for health applications on mobile devices using active and semi-supervised learning. In *International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*. IEEE.

[45] Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. 2016. Quantiles over data streams: experimental comparisons, new analyses, and further improvements. *The VLDB Journal* 25, 4 (2016), 449–472.

[46] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 681–688.

[47] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 346–357.

[48] Chandler May, Kevin Duh, Benjamin Van Durme, and Ashwin Lall. 2017. Streaming Word Embeddings with the Space-Saving Algorithm. *arXiv preprint arXiv:1704.07463* (2017).

[49] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al. 2016. Personalized speech recognition on mobile devices. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 5955–5959.

[50] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[51] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and explanations in databases. In *VLDB*.

[52] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*. Springer, 398–412.

[53] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[54] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas, and Divesh Srivastava. 2015. Conditional heavy hitters: detecting interesting correlations in data streams. *The VLDB Journal* 24, 3 (2015), 395–414.

[55] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1135–1144.

[56] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1449–1463.

[57] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. 2004. Reversible sketches for efficient and accurate change detection over network data streams.

In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 207–212.
[58] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming* 127, 1 (2011), 3–30.
[59] Ohad Shamir. 2016. Without-Replacement Sampling for Stochastic Gradient Methods. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 46–54. http://papers.nips.cc/paper/6245-without-replacement-sampling-for-stochastic-gradient-methods.pdf
[60] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander L Strehl, Alex J Smola, and SVN Vishwanathan. 2009. Hash kernels. In *International Conference on Artificial Intelligence and Statistics*. 496–503.
[61] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 239–249.
[62] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated Multi-Task Learning. In *Advances in Neural Information Processing Systems*. 4427–4437.
[63] J. Stamper, A. Niculescu-Mizil, S. Ritter, G.J. Gordon, and K.R. Koedinger. 2010. Algebra I 2008-2009. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge. (2010). Find it at http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp.
[64] Jacob Steinhardt and John Duchi. 2015. Minimax rates for memory-bounded sparse linear regression. In *Conference on Learning Theory*. 1564–1587.
[65] Peter D Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research* 37 (2010), 141–188.
[66] CAIDA UCSD. 2008. The CAIDA UCSD Anonymized Passive OC48 Internet Traces Dataset. (2008). http://www.caida.org/data/passive/passive_oc48_dataset.xml.
[67] Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2010. EffiCuts: optimizing packet classification for memory and throughput. In *ACM SIGCOMM Computer Communication Review*, Vol. 40. ACM, 207–218.
[68] Shoba Venkataraman, Dawn Song, Phillip B Gibbons, and Avrim Blum. 2005. New streaming algorithms for fast detection of superspreaders. *Department of Electrical and Computing Engineering* (2005), 6.
[69] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 1113–1120.
[70] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment* 6, 8 (2013), 553–564.
[71] Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research* 11, Oct (2010), 2543–2596.
[72] Tianbao Yang, Lijun Zhang, Rong Jin, and Shenghuo Zhu. 2015. Theory of dual-sparse regularized randomized reduction. In *International Conference on Machine Learning*. 305–314.
[73] Hsiang-Fu Yu, Hung-Yi Lo, Hsun-Ping Hsieh, Jing-Kai Lou, Todd G McKenzie, Jung-Wei Chou, Po-Han Chung, Chia-Hua Ho, Chun-Fu Chang, Yin-Hsuan Wei, et al. 2010. Feature engineering and classifier ensemble for KDD cup 2010. In *KDD Cup*.
[74] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch.. In *NSDI*, Vol. 13. 29–42.
[75] Tong Yu, Yong Zhuang, Ole J Mengshoel, and Osman Yagan. 2016. Hybridizing personal and impersonal machine learning models for activity recognition on mobile devices.
[76] Ce Zhang, Arun Kumar, and Christopher Ré. 2016. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)* 41, 1 (2016), 2.
[77] Lijun Zhang, Mehrdad Mahdavi, Rong Jin, Tianbao Yang, and Shenghuo Zhu. 2014. Random projections for classification: A recovery approach. *IEEE Transactions on Information Theory* 60, 11 (2014), 7300–7316.
[78] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 928–936.
[79] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 2 (2005), 301–320.

A PROOFS

A.1 Proof of Theorem 1

We will use the duals of $L(\mathbf{w})$ and $\hat{L}(\mathbf{z})$ to show that $\mathbf{z}_*$ is close to $R\mathbf{w}_*$, following the analysis of Zhang et al. [77] and Yang et al. [72]. Define $\tilde{\mathbf{x}}_i = y_i\mathbf{x}_i$, i.e. the ith data point $\mathbf{x}_i$ times its label. Let $\tilde{X} \in \mathbb{R}^{d\times T}$ be the matrix of data points such that the ith column is $\tilde{\mathbf{x}}_i$. Let $G = \tilde{X}^T\tilde{X}$ be the Gram matrix corresponding to the original data points. Forming the Lagrangian and minimizing with respect to the primal variables gives us the following dual objective function in terms of the dual variable $\alpha \in \mathbb{R}^T$,

$$J(\alpha) = \frac{1}{T}\sum_i \ell^*(\alpha_i) + \frac{1}{2\lambda T^2}\alpha^T G\alpha,$$

where $\ell^*(\alpha_i)$ is the Fenchel conjugate of $\ell(z_i)$. Note that if α_* is the minimizer of $J(\alpha)$, then the minimizer $\mathbf{w}_*$ of $L(\mathbf{w})$ is given by $\mathbf{w}_* = -\frac{1}{\lambda T}\tilde{X}\alpha_*$.

We similarly define $G = \tilde{X}^T R^T R\tilde{X}$ as the Gram matrix corresponding to the projected data points. We can write down the dual $\hat{L}(\alpha)$ of the projected primal objective function $\hat{J}(\mathbf{w})$ in terms of the dual variable $\hat{\alpha}$ as follows:

$$\hat{J}(\hat{\alpha}) = \frac{1}{T}\sum_i \ell^*(\hat{\alpha}_i) + \frac{1}{2\lambda T^2}\hat{\alpha}^T\hat{G}\hat{\alpha}.$$

As before, if $\hat{\alpha}_*$ is the minimizer of $\hat{J}(\hat{\alpha})$, then the minimizer $\mathbf{z}_*$ of $\hat{L}(\mathbf{z})$ is given by $\hat{\mathbf{w}}_* = -\frac{1}{\lambda T}R\tilde{X}\hat{\alpha}_*$.

We willfi rst express the distance between $\mathbf{z}_*$ and $R\mathbf{w}_*$ in terms of the distance between the dual variables. We can write:

$$\begin{aligned}\|\mathbf{z}_* - R\mathbf{w}_*\|_2^2 &= \frac{1}{\lambda^2T^2}\|R\tilde{X}\hat{\alpha}_* - R\tilde{X}\alpha_*\|_2^2 \\ &= \frac{1}{\lambda^2T^2}(\hat{\alpha}_* - \alpha_*)^T\hat{G}(\hat{\alpha}_* - \alpha_*). \qquad (2)\end{aligned}$$

Hence, our goal will be to upper bound $(\hat{\alpha}_* - \alpha_*)^T\hat{G}(\hat{\alpha}_* - \alpha_*)$. Define $\Delta = \frac{1}{\lambda T}(\hat{G} - G)\alpha_*$. We will show that $(\hat{\alpha}_* - \alpha_*)^T\hat{G}(\hat{\alpha}_* - \alpha_*)$ can be upper bounded in terms of Δ as follows.

Lemma2.

$$\frac{1}{\lambda^2T^2}(\hat{\alpha}_* - \alpha_*)^T\hat{G}(\hat{\alpha}_* - \alpha_*) \le \frac{2\beta}{\lambda}\|\Delta\|_\infty^2$$

Due to space constraints, we omit the proof of Lemma 2 here, deferring it to the full version of the paper. The proof relies on the convexity and strong-smoothness of the loss function ℓ.

We now bound $\|\Delta\|_\infty$. The result relies on the JL property of the projection matrix R (recall Definition 1). If R is a JL matrix with error ϵ and failure probability δ/d^2, then it is straightforward to verify that with failure probability δ, for all coordinate basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_d\}$,

$$\|R\mathbf{e}_i\|_2 = 1 \pm \epsilon,\ \forall\ i, \quad |\langle R\mathbf{e}_i, R\mathbf{e}_j\rangle| \le\ \epsilon,\ \forall\ i \ne j. \qquad (3)$$

Using this projection, we show the following bound on $\|\Delta\|_\infty$:

Lemma3. *If R satisfies condition 3, then:*

$$\|\Delta\|_\infty \le 2\gamma\epsilon\|\mathbf{w}_*\|_1,$$

where $\gamma = \max_i \|\mathbf{x}_i\|_1$.

Proof. Wefi rst rewrite Δ as follows,

$$\Delta = \frac{1}{\lambda T}(\tilde{X}^T R^T R\tilde{X} - \tilde{X}^T\tilde{X})\alpha_* = \frac{1}{\lambda T}\tilde{X}^T(R^T R - I)\tilde{X}\alpha_* = \tilde{X}^T(I - R^T R)\mathbf{w}_*,$$

using the relation that $\mathbf{w}_* = -\frac{1}{\lambda T}\tilde{X}\alpha_*$. Therefore,

$$\|\Delta\|_\infty \le \max_i |\mathbf{x}_i^T(I - R^T R)\mathbf{w}_*| = \max_i |\mathbf{x}_i^T\mathbf{w}_* - (R\mathbf{x}_i)^T(R\mathbf{w}_*)|.$$

We now claim that if condition 3 is satisfied, then for any two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$,

$$|\mathbf{v}_1^T\mathbf{v}_2 - (R\mathbf{v}_1)^T(R\mathbf{v}_2)| \le 2\epsilon\,\|\mathbf{v}_1\|_1\,\|\mathbf{v}_2\|_1\,. \tag{4}$$

The proof follows from simple algebra, and is omitted from this version for lack of space. Using this relation, it follows that,

$$\|\Delta\|_\infty \le \max_i |\mathbf{x}_i^T\mathbf{w}_* - (R\mathbf{x}_i)^T(R\mathbf{w}_*)| \le 2\epsilon\gamma\|\mathbf{w}_*\|_1.$$

□

We will now combine Lemma 2 and 3. By Eq. 2 and Lemma 2,

$$\|\mathbf{z}_* - R\mathbf{w}_*\|_2^2 \le \frac{2\beta}{\lambda}\|\Delta\|_\infty^2.$$

If R is a JL matrix with error ϵ and failure probability δ/d^2, then by Lemma 3, with failure probability δ,

$$\|\mathbf{z}_* - R\mathbf{w}_*\|_2 \le 4\gamma\epsilon\sqrt{\frac{\beta}{\lambda}}\|\mathbf{w}_*\|_1. \tag{5}$$

By Kane and Nelson [35], the random projection matrix R satisfies the JL property with error θ and failure probability δ'/d^2 for $k \ge C\log(d/\delta')/\theta^2$, where C is afi xed constant. Using Eq. 5, with failure probability δ',

$$\|\mathbf{z}_* - R\mathbf{w}_*\|_2 \le 4\gamma\theta\sqrt{\frac{\beta}{\lambda}}\|\mathbf{w}_*\|_1. \tag{6}$$

Recall that $\sqrt{s}R$ is a Count-Sketch matrix with width C_1/θ and depth $s = C_2\log(d/\delta')/\theta$, where C_1 and C_2 arefi xed constants. Let $\mathbf{w}_{\text{proj}}$ be the projection of $\mathbf{w}_*$ with the Count-Sketch matrix $\tilde{R}$, hence $\mathbf{w}_{\text{proj}} = \sqrt{s}R\mathbf{w}_*$. Let $\mathbf{z}_{\text{proj}} = \sqrt{s}\mathbf{z}_*$. By Eq. 6, with failure probability δ',

$$\left\|\mathbf{z}_{\text{proj}} - \mathbf{w}_{\text{proj}}\right\|_2 \le \sqrt{\frac{16\beta\gamma^2\theta\log(d/\delta')}{\lambda}}\|\mathbf{w}_*\|_1.$$

Let $\mathbf{w}_{\text{cs}}$ be the Count-Sketch estimate of $\mathbf{w}_*$ derived from $\mathbf{w}_{\text{proj}}$, and $\mathbf{w}_{\text{est}}$ be the Count-Sketch estimate of $\mathbf{w}_*$ derived from $\mathbf{z}_{\text{proj}}$. Recall that the Count-Sketch estimate of a vector is the median of the estimates of all the locations to which the vector hashes. As the difference between the median of any two vectors is at most the ℓ_∞-norm of their difference,

$$\|\mathbf{w}_{\text{est}} - \mathbf{w}_{\text{cs}}\|_\infty \le \left\|\mathbf{z}_{\text{proj}} - \mathbf{w}_{\text{proj}}\right\|_\infty.$$

Therefore with failure probability δ',

$$\|\mathbf{w}_{\text{est}} - \mathbf{w}_{\text{cs}}\|_\infty \le \left\|\mathbf{z}_{\text{proj}} - \mathbf{w}_{\text{proj}}\right\|_\infty \le \left\|\mathbf{z}_{\text{proj}} - \mathbf{w}_{\text{proj}}\right\|_2 \le \sqrt{\frac{16\beta\gamma^2\theta\log(d/\delta')}{\lambda}}\|\mathbf{w}_*\|_1. \tag{7}$$

We now use Lemma 1 to bound the error for Count-Sketch recovery. Using Lemma 1 for the matrix $\sqrt{s}R$, with failure probability δ',

$$\|\mathbf{w}_* - \mathbf{w}_{\text{cs}}\|_\infty \le \sqrt{\theta}\|\mathbf{w}_*\|_2.$$

Now using the triangle inequality and Eq. 7, with failure probability $2\delta'$ (due to a union bound),

$$\begin{aligned}\|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty &\le \|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty + \|\mathbf{w}_{\text{est}} - \mathbf{w}_{\text{cs}}\|_\infty \\ &\le \sqrt{\theta}\|\mathbf{w}_*\|_2 + \sqrt{\frac{16\beta\gamma^2\theta\log(d/\delta')}{\lambda}}\|\mathbf{w}_*\|_1 \\ &\le \left(\sqrt{\theta} + \sqrt{\frac{16\beta\gamma^2\theta\log(d/\delta')}{\lambda}}\right)\|\mathbf{w}_*\|_1.\end{aligned}$$

Therefore choosing $\theta = \min\{1, \lambda/(16\beta\gamma^2\log(d/\delta'))\}\epsilon^2/4$, with failure probability $2\delta'$,

$$\|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty \le \epsilon\|\mathbf{w}_*\|_1.$$

Choosing $\delta' = \delta/2$, we have that forfi xed constants C_1, C_2,

$$k = (C_1/\epsilon^4)\log^3(d/\delta)\max\{1, \beta^2\gamma^4/\lambda^2\},$$
$$s = (C_2/\epsilon^2)\log^2(d/\delta)\max\{1, \beta\gamma^2/\lambda\},$$

$\|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty \le \epsilon\|\mathbf{w}_*\|_1$, with probability $1 - \delta$.

A.2 Proof of Theorem 2

Let $f_t(\mathbf{z})$ be the loss function corresponding to the data point chosen in the tth time step:

$$f_t(\mathbf{z}) = \ell\left(y_t\mathbf{z}^T R\mathbf{x}_t\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2. \tag{8}$$

Let $\mathbf{z}_t$ be the weight vector at the tth time step for online updates on the projected problem. Let $\bar{\mathbf{z}} = \frac{1}{T}\sum_{i=1}^T \hat{\mathbf{z}}_i$ be the average of the weight vectors for all the T time steps. We claim that $\bar{\mathbf{z}}$ is close to $\mathbf{z}_*$, the optimizer of $\hat{L}(\mathbf{z})$, using Corollary 1 of Shamir [59]. In order to apply the result wefi rst need to define a few parameters of the function $\hat{L}(\mathbf{z})$. Note that $\hat{L}(\mathbf{z})$ is λ-strongly convex (since $\hat{L}(\mathbf{z}) - \frac{\lambda}{2}\|\mathbf{z}\|_2^2$ is convex). Moreover, since the derivative of ℓ is bounded above by H, ℓ is H-Lipschitz. We assume $\|R\mathbf{x}_i\|_2 \le B, \|\mathbf{z}_*\|_2 \le D$ and $\max_t \|\nabla f_t(\mathbf{w})\|_2 \le G$. We will bound B, D and G in the end. We now apply Corollary 1 of Shamir [59], with the notation adapted for our setting.

Lemma4. *[59] Consider any loss function $\hat{L}(\mathbf{z}) = \sum_{i=1}^T f_t(\mathbf{z})$, where $f_t(\mathbf{z})$ is defined in Eq. 8. For any H-Lipchitz ℓ_i, $\|R\mathbf{x}_i\|_2 \le B$, $\|\mathbf{z}_t\|_2 \le D$, and somefi xed constant C, over the randomness in the order in which the samples are received:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^T \hat{L}(\mathbf{z}_t) - \hat{L}(\mathbf{z}_*)\right] \le \frac{C(R_T/\sqrt{T} + BDH)}{\sqrt{T}},$$

where R_T is the regret of online gradient descent with respect to the batch optimizer $\mathbf{z}_*$, defined as $R_T = \sum_{t=1}^T[f_t(\hat{\mathbf{z}}) - f_t(\mathbf{z}_*)]$.

By standard regret bounds on online gradient descent (see Zinkevich [78]), $R_T \le GD\sqrt{T}$. Therefore,

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^T \hat{L}(\mathbf{z}_t) - \hat{L}(\mathbf{z}_*)\right] \le \frac{CD(G + BH)}{\sqrt{T}}.$$

Note that by Jensen's inequality,

$$\mathbb{E}[\hat{L}(\mathbf{z})] \leq \mathbb{E}\Big[\frac{1}{T}\sum_{t=1}^{T}\hat{L}(\mathbf{z}_t)\Big]$$
$$\implies \mathbb{E}\Big[\hat{L}(\bar{\mathbf{z}}) - \hat{L}(\mathbf{z}_*)\Big] \leq \frac{CD(G+BH)}{\sqrt{T}}. \quad (9)$$

We will now bound the expected distance between $\bar{\mathbf{z}}$ and $\mathbf{z}_*$ using Eq. 9 and the strong convexity of $\hat{L}(\mathbf{w})$. As $\hat{L}(\mathbf{w})$ is λ-strongly convex and $\nabla\hat{L}(\mathbf{z}_*) = 0$, we can write:

$$\hat{L}(\mathbf{z}_*) + (\lambda/2)\|\bar{\mathbf{z}} - \mathbf{z}_*\|_2^2 \leq \hat{L}(\bar{\mathbf{z}})$$
$$\implies \|\bar{\mathbf{z}} - \mathbf{z}_*\|_2^2 \leq (\lambda/2)[\hat{L}(\bar{\mathbf{z}}) - \hat{L}(\mathbf{z}_*)]$$
$$\implies \mathbb{E}\Big[\|\bar{\mathbf{z}} - \mathbf{z}_*\|_2^2\Big] \leq (2/\lambda)\Big[\mathbb{E}[\hat{L}(\bar{\mathbf{z}})] - \hat{L}(\mathbf{z}_*)\Big].$$

Using Eq. 9 and then Jensen's inequality,

$$\mathbb{E}\Big[\|\bar{\mathbf{z}} - \mathbf{z}_*\|_2\Big] \leq \frac{2CD(G+BH)}{\lambda\sqrt{T}}. \quad (10)$$

Let $\bar{\mathbf{z}}_{\text{proj}} = \sqrt{s}\bar{\mathbf{z}}$. Let $\mathbf{z}_{\text{wm}}$ be the Count-Sketch estimate of $\mathbf{w}_*$ derived from $\bar{\mathbf{z}}_{\text{proj}}$. Recall from the proof of Theorem 1 that $\mathbf{z}_{\text{proj}} = \sqrt{s}\mathbf{z}$ and $\mathbf{w}_{\text{est}}$ is the Count-Sketch estimate of $\mathbf{w}_*$ derived from $\mathbf{z}_{\text{proj}}$. As in the proof of Theorem 1, we note that the difference between the medians of any two vectors is at most the ℓ_∞ norm of the difference of the vectors, and hence we can write,

$$\|\mathbf{w}_{\text{est}} - \mathbf{z}_{\text{wm}}\|_\infty \leq \Big\|\mathbf{z}_{\text{proj}} - \bar{\mathbf{z}}_{\text{proj}}\Big\|_\infty \leq \Big\|\mathbf{z}_{\text{proj}} - \bar{\mathbf{z}}_{\text{proj}}\Big\|_2$$
$$= \sqrt{s}\|\mathbf{z}_* - \bar{\mathbf{z}}\|_2.$$

Therefore, using Eq. 10,

$$\mathbb{E}[\|\mathbf{w}_{\text{est}} - \mathbf{z}_{\text{wm}}\|_\infty] \leq \frac{2CD(G+BH)}{\lambda}\sqrt{\frac{s}{T}}. \quad (11)$$

By the triangle inequality,

$$\|\mathbf{w}_* - \mathbf{z}_{\text{wm}}\|_\infty \leq \|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty + \|\mathbf{w}_{\text{est}} - \mathbf{z}_{\text{wm}}\|_\infty$$
$$\implies \mathbb{E}\Big[\ \|\mathbf{w}_* - \mathbf{z}_{\text{wm}}\|_\infty\ \Big] \leq \mathbb{E}\Big[\ \|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty\ \Big] + \mathbb{E}\Big[\ \|\mathbf{w}_{\text{est}} - \mathbf{z}_{\text{wm}}\|_\infty\ \Big].$$

By Theorem 1, forfi xed constants C_1, C_2 and

$$k = (C_1/\epsilon^4)\log^3(d/\delta)\max\{1, \beta^2\gamma^4/\lambda^2\},$$
$$s = (C_2/\epsilon^2)\log^2(d/\delta)\max\{1, \beta\gamma^2/\lambda\},$$

$\|\mathbf{w}_* - \mathbf{w}_{\text{est}}\|_\infty \leq \epsilon\|\mathbf{w}_*\|_1$ with probability $1-\delta$. Therefore, forfi xed constants C_1' and C_2' and probability $1-\delta$,

$$\mathbb{E}\Big[\ \|\mathbf{w}_* - \mathbf{z}_{\text{wm}}\|_\infty\ \Big] \leq \frac{\epsilon}{2}\|\mathbf{w}_*\|_1 + \sqrt{\frac{4C_2'(GD+BDH)^2\log^2(d/\delta)\max\{1, LR^2/\lambda\}}{\lambda^2\epsilon^2 T}}.$$

Therefore, for

$$T \geq (C_3'/(\epsilon^4\lambda^2))(D/\|\mathbf{w}_*\|_1)^2(G+BH)^2\log^2(d/\delta)\max\{1, LR^2/\lambda\},$$

$$\mathbb{E}\Big[\ \|\mathbf{w}_* - \mathbf{z}_{\text{wm}}\|_\infty\ \Big] \leq \frac{\epsilon}{2}\|\mathbf{w}_*\|_1 + \frac{\epsilon}{2}\|\mathbf{w}_*\|_2 \leq \epsilon\|\mathbf{w}_*\|_1.$$

We will now bound B, D and G, starting with B. Note that R is a JL matrix which satisfies condition 3 with $\epsilon = \theta$. Using Eq. 4 and the fact that $\|\mathbf{x}_i\|_2 \leq 1$,

$$\|R\mathbf{x}_i\|_2 \leq \sqrt{1+\theta\gamma^2} \implies B \leq 1 + \sqrt{\theta}\gamma \leq 1 + \epsilon\gamma,$$

where for the last bound we use the setting of

$$\theta = \min\{1, \lambda/(4\beta\gamma^2\log(d/\delta'))\}\epsilon^2/4$$

from the proof of Theorem 1. We next bound $\|\mathbf{z}_*\|_2$. Using Eq. 4,

$$\|\mathbf{z}_* - R\mathbf{w}_*\|_2 \leq 2R\theta\sqrt{\beta/\lambda}\,\|\mathbf{w}_*\|_1$$
$$\implies \|\mathbf{z}_*\|_2 \leq \|R\mathbf{w}_*\|_2 + 2R\theta\sqrt{\beta/\lambda}\,\|\mathbf{w}_*\|_1.$$

By Eq. 4, $\|R\mathbf{w}_*\|_2 \leq \sqrt{\|\mathbf{w}_*\|_2^2 + \theta\|\mathbf{w}_*\|_1^2} \leq \|\mathbf{w}_*\|_2 + \sqrt{\theta}\|\mathbf{w}_*\|_1$. Therefore,

$$\|\mathbf{z}_*\|_2 \leq \|\mathbf{w}_*\|_2 + \sqrt{\theta}\|\mathbf{w}_*\|_1 + 2R\theta\sqrt{\beta/\lambda}\,\|\mathbf{w}_*\|_1$$
$$= \|\mathbf{w}_*\|_2 + \Big(\sqrt{\theta} + 2R\theta\sqrt{\beta/\lambda}\Big)\|\mathbf{w}_*\|_1.$$

For our choice of θ,

$$\|\mathbf{z}_*\|_2 \leq \|\mathbf{w}_*\|_2 + \epsilon\|\mathbf{w}_*\|_1 \implies D \leq D_2 + \epsilon D_1.$$

This implies that the $(D/\|\mathbf{w}_*\|_1)$ term in our bound for T can be upper bounded by $2D_2/\|\mathbf{w}_*\|_1$, yielding the bound on T stated in Theorem 2. Finally, we need to upper bound $G = \max_t \|\nabla f_t(\mathbf{w})\|_2$. We do this as follows:

$$\nabla f_t(\mathbf{z}) = \ell'(y_t\mathbf{z}_t^T R\mathbf{x_t})A\mathbf{x_t} + \lambda\mathbf{z}_t$$
$$\implies \|\nabla f_t(\mathbf{z})\|_2 \leq |\ell'(y_t\mathbf{z}_t^T Rx)|\|Rx\|_2 + \lambda\|\mathbf{z}_t\|_2$$
$$\leq H(1+\epsilon\gamma) + \lambda D.$$

B BASELINE ALGORITHMS

Here we give pseudocode for the simple truncation and probabilistic truncation baselines evaluated in our experiments.

Algorithm 3: Simple Truncation

input: loss function ℓ, budget K, ℓ_2-regularization parameter λ, learning rate schedule η_t
initialization
  $S \leftarrow \{\}$ ▷ Empty heap
function `Update(x, y)`
  $\tau \leftarrow \sum_{i\in S} S[i]\cdot x_i$ ▷ Make prediction
  $S \leftarrow (1-\lambda\eta_t)S - \eta_t y x_i \nabla\ell(y\tau)$
  Truncate S to top-K entries by magnitude
  $t \leftarrow t+1$

Algorithm 4: Probabilistic Truncation

initialization
  $S_0 \leftarrow \{\}$ ▷ Empty heap
  $W \leftarrow \{\}$ ▷ Reservoir weights
function `Update(x, y)`
  $\tau \leftarrow \sum_{i\in S_t} S_t[i]\cdot x_i$ ▷ Make prediction
  $S_{t+1} \leftarrow (1-\lambda\eta_t)S_t - \eta_t y\mathbf{x}\nabla\ell(y\tau)$
  for $i \in S_{t+1}$ **do**
    if $i \notin S_t$ **then**
      $r \sim \mathcal{U}(0,1)$
      $W[i] \leftarrow r^{1/|S_{t+1}[i]|}$ ▷ New reservoir weight
    else
      $W[i] \leftarrow W[i]^{|S_t[i]/S_{t+1}[i]|}$ ▷ Update weight
  Truncate S_{t+1} to top-K entries by reservoir weight
  $t \leftarrow t+1$