ABSTRACT
We introduce the Weight-Median Sketch, a sub-linear space data structure that captures the most heavily weighted features in linear classifiers trained over data streams. 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. On a standard binary classification benchmark, the Weight-Median Sketch enables estimation of the top-100 weights with 10% relative error using two orders of magnitude less space than an uncompressed classifier.

1 INTRODUCTION
Memory-efficient sketching algorithms are a well-established tool in stream processing tasks such as frequent item identification [4, 5, 16], quantile estimation [10], and approximate counting of distinct items [9]. Sketching algorithms compute approximations of these quantities in exchange for significant reductions in memory utilization. Therefore, they are a good choice when highly-accurate estimation is not essential and practitioners wish to trade off between memory usage and approximation accuracy [3, 24].

Simultaneously, a wide range of streaming analytics workloads can be formulated as learning problems over streaming data. In streaming data explanation [1, 15], analyses seek to explain the difference between subpopulations in the data (e.g., between an inlier class and an outlier class) using a small set of discriminative features. In network monitoring, analyses seek to identify sets of features (e.g., source/destination IP addresses) that exhibit the most significant relative differences in occurrence between streams of network traffic [6]. In natural language processing on text streams, several applications require the identification of strongly-associated groups of tokens [8]; this pertains more broadly to the problem of identifying groups of events which tend to co-occur.

These tasks can all be framed as instances of streaming classification between two or more classes of interest, followed by the interpretation of the learned model in order to identify the features that are the most discriminative between the classes. Further, as is the case with classical tasks like identifying frequent items or counting distinct items, these use cases typically allow for some degree of approximation error in the returned results. We are therefore interested in algorithms that offer similar memory-accuracy tradeoffs to classical sketching algorithms, but in the context of online learning for linear classifiers.

Finding Heavily-Weighted Features with the Weight-Median Sketch
Extended Abstract
Kai Sheng Tai, Vatsal Sharan, Peter Bailis, Gregory Valiant
Stanford University

Figure 1: Streaming updates to a sketched classifier with approximations of the most heavily-weighted features.

We introduce a new sketching algorithm—the Weight-Median Sketch (WM-Sketch)—for binary linear classifiers trained on streaming data.1 As each new labeled example, \((x,y)\) with \(x \in \mathbb{R}^d\), \(y \in \{-1, +1\}\), is observed in the stream, we update a fixed-size data structure in memory with online gradient descent. This data structure represents a compressed version of the full \(d\)-dimensional classifier trained on the same data stream. This structure can be efficiently queried for estimates of the top-\(K\) highest-magnitude weights in the classifier. Importantly, we show that it suffices that this compressed data structure is of size polylogarithmic in the feature dimension \(d\). Therefore, the WM-Sketch can be used to realize substantial memory savings in high dimensional classification problems where the user is interested in obtaining a \(K\)-sparse approximation to the classifier, where \(K \ll d\).

2 THE WEIGHT-MEDIAN SKETCH
The main data structure in the WM-Sketch is identical to that used in the Count-Sketch [4]. The sketch is parameterized by size \(k\), depth \(s\), and width \(k/s\). We initialize with a size-\(k\) array set to zero. We view this array \(z\) as being arranged in \(s\) rows, each of width \(k/s\).

The high-level idea is that each row of the sketch is a compressed version of the model weight vector \(w \in \mathbb{R}^d\), 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.

**Updates.** For each feature \(i\), we assign a uniformly random index \(h_j(i)\) in each row \(j \in [s]\) and a uniformly random sign \(\sigma_j(i)\). Given an update \(\Delta_i \in \mathbb{R}\) to weight \(w_i\), we apply the update by adding \(\sigma_j(i)\Delta_i\) index \(h_j(i)\) of each row \(j\). Denoting this random projection by the matrix \(A \in [-1, +1]^{s \times d}\), we can then write the update \(\Delta\) to \(z\) as \(\Delta = AZ\). In practice, this assignment is done via hashing.

Instead of being provided the updates \(\Delta\), we must compute them as a function of the input example \((x,y)\) and the sketch state \(z\). Given the convex and differentiable loss function \(\ell\), we define the update to \(z\) as the online gradient descent update for the sketched example \((Ax,y)\). In particular, we first make a prediction \(\tau = (1/s)z^TAx\),

---

1The full version of this paper is available at [19].
### Algorithm 1: Weight-Median (WM) Sketch

**input:** size \( k \), depth \( s \), loss function \( \ell \), \( \ell_2 \)-regularization parameter \( \lambda \), learning rate schedule \( \eta_t \)

**initialization**
- \( z \leftarrow s \times k \) array of zeroes
- Sample Count-Sketch matrix \( A \)
- \( t \leftarrow 0 \)

**function Update(x, y)**
- \( \tau \leftarrow \frac{1}{2}z^T A x \quad \text{\( \triangleright \)} \text{ Prediction for } x \)
- \( z \leftarrow (1 - \lambda \eta_t)z - \eta_t y \ell(y \ell(x)) A x \quad \text{\( \triangleright \)} \text{ Update with } \ell_2 \text{ reg.} \)
- \( t \leftarrow t + 1 \)

**function Query(i)**
- **return** output of Count-Sketch retrieval on \( z \)

and then compute the gradient update \( \hat{\lambda} = -\eta y \nabla \ell(y \ell(x)) A x \) with learning rate \( \eta \).

For intuition, we can compare this update to the Count-Sketch update rule [4]. In the frequent-items setting, the input \( x \) is a one-hot encoding for the observed item. The update to the Count-Sketch state \( z_{cs} \) is simply \( \hat{z}_{cs} = Ax \), where \( A \) is defined identically as above. Therefore, our update rule is simply the Count-Sketch update scaled by the constant \(-\eta y \nabla \ell(yz^T A x/s)\). However, an important detail to note is that the Count-Sketch update is independent of the sketch state \( z_{cs} \); whereas the WM-Sketch update does depend on \( z \). 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 \( i \)th weight, we return the median of the values \( \sigma_j(i) \cdot z_i, h_j(i) \) for \( j \in [s] \). This is identical to the query procedure for the Count-Sketch.

**Analysis.** We show that for feature dimension \( d \) and with success probability \( 1 - \delta \), we can learn a compressed model of dimension \( O\left(e^{-9} \log^3 (d/\delta)\right) \) that supports approximate recovery of the optimal weight vector \( \omega \), over all the examples seen so far, where the absolute error of each weight estimate is bounded above by \( \epsilon \|w_i\|_1 \); For a given input vector \( x \), this structure can be updated in time \( O\left(e^{-2} \log^2 (d/\delta) \cdot \frac{\text{nlex}}{x}\right) \). For formal statements of our theorems, proofs, and additional discussion of the conditions under which this result holds, see the full version of the paper [19].

**Active-Set Weight-Median Sketch (AWM-Sketch).** We can significantly improve the recovery accuracy of the WM-Sketch in practice using a simple, heuristic extension. To efficiently track the top elements across sketch updates, we can use a size-\( K \) min-heap ordered by the absolute value of the estimated weights. Weights that are already stored in the heap need not be tracked in the sketch; instead, the sketch is updated lazily only when the weight is evicted from the heap. Additionally, the heap estimates are used to compute the sketch updates instead of querying values from the sketch. In practice, this trick can improve recovery accuracy by an order of magnitude without any additional space usage.

### 3 EVALUATION

**Classification Benchmarks.** We evaluated the recovery error on \( \ell_2 \)-regularized online logistic regression trained on three standard binary classification datasets: Reuters RCV1 [12], malicious URL identification [13], and the KDD A1gebra dataset [18, 23] (see Fig. 2 for results on RCV1 and URL). We compared against several memory-budgeted baselines: hard thresholding and a probabilistic variant (Trun, PTrun), tracking frequent features with the Space Saving algorithm (SS), and feature hashing (Hash). We found that the AWM-Sketch achieved lower recovery error across all three baselines. On RCV1, the AWM-Sketch achieved 4x lower error relative to the baseline of only learning weights for frequently-occurring features. Compared to an uncompressed classifier, the AWM-Sketch uses two orders of magnitude less space at the cost of 10% relative error in the recovered weights. In terms of classification accuracy, the AWM-Sketch with an 8KB budget achieved 1% higher error rate than an uncompressed classifier on RCV1, while outperforming all the baseline methods by at least 1%.

**Application: Network Monitoring.** IP network monitoring is one of the primary application domains for sketches and other small-space summary methods [2, 20, 24]. Here, we focus on finding source/destination IP addresses that differ significantly in relative magnitude without any additional space usage.

![Figure 2: Relative \( \ell_2 \) error of estimated top-\( K \) weights vs. true top-\( K \) weights for \( \ell_2 \)-regularized logistic regression under an 8KB memory budget. The AWM-Sketch achieves lower recovery error across both datasets.](image-url)
ACKNOWLEDGMENTS
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 DARPA under No. FA8750-17-2-0095 (D3M) and industrial gifts and support from Toyota Research Institute, Juniper Networks, Keysight Technologies, Hitachi, Facebook, Northrop Grumman, and NetApp.

REFERENCES