Finding Heavily-Weighted Features with the Weight-Median Sketch

Extended Abstract

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

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.

SysML'18, February 2018, Stanford, CA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

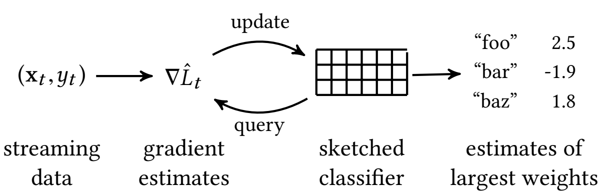

Figure 1: Streaming updates to a sketched classifier with approximations of the most heavily-weighted features.

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, $(\mathbf{x}, y)$ with $\mathbf{x} \in \mathbb{R}^d$, $y \in \{-1, +1\}$, 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 $\mathbf{z}$ 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 $\mathbf{w} \in \mathbb{R}^d$, 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.

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 $A \in \{-1, +1\}^{k \times d}$, we can then write the update $\tilde{\Delta}$ to $\mathbf{z}$ as $\tilde{\Delta} = A\Delta$. In practice, this assignment is done via hashing.

Instead of being provided the updates Δ, we must compute them as a function of the input example $(\mathbf{x}, y)$ and the sketch state $\mathbf{z}$. Given the convex and differentiable loss function ℓ, we define the update to $\mathbf{z}$ as the online gradient descent update for the *sketched* example $(A\mathbf{x}, y)$. In particular, we first make a prediction $\tau = (1/s)\mathbf{z}^T A\mathbf{x}$,

[1] The full version of this paper is available at [19].

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 Count-Sketch matrix A
- $t \leftarrow 0$

function `Update`$(\mathbf{x}, y)$

- $\tau \leftarrow \frac{1}{s}\mathbf{z}^T A\mathbf{x}$ ▷ Prediction for $\mathbf{x}$
- $\mathbf{z} \leftarrow (1 - \lambda\eta_t)\mathbf{z} - \eta_t y \nabla\ell(y\tau)\, A\mathbf{x}$ ▷ Update with ℓ_2 reg.
- $t \leftarrow t + 1$

function `Query`(i)

- **return** output of Count-Sketch retrieval on $\mathbf{z}$

and then compute the gradient update $\tilde{\Delta} = -\eta y \nabla\ell(y\tau)\, A\mathbf{x}$ with learning rate η.

For intuition, we can compare this update to the Count-Sketch update rule [4]. In the frequent-items setting, the input $\mathbf{x}$ is a one-hot encoding for the observed item. The update to the Count-Sketch state $\mathbf{z}_{\text{cs}}$ is simply $\tilde{\Delta}_{\text{cs}} = A\mathbf{x}$, where A is defined identically as above. Therefore, our update rule is simply the Count-Sketch update scaled by the constant $-\eta y \nabla\ell(y\mathbf{z}^T A\mathbf{x}/s)$. 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 $\sigma_j(i) \cdot z_{j,h_j(i)}$ 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 $O\left(\epsilon^{-4}\log^3(d/\delta)\right)$ that supports approximate recovery of the optimal weight vector $\mathbf{w}_*$ over all the examples seen so far, where the absolute error of each weight estimate is bounded above by $\epsilon\|\mathbf{w}_*\|_1$. For a given input vector $\mathbf{x}$, this structure can be updated in time $O\left(\epsilon^{-2}\log^2(d/\delta)\cdot \text{nnz}(\mathbf{x})\right)$. 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 ℓ_2-regularized online logistic regression trained on three standard binary classification datasets: Reuters RCV1 [12], malicious URL identification [13], and the KDD `Algebra` 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 4× 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%.

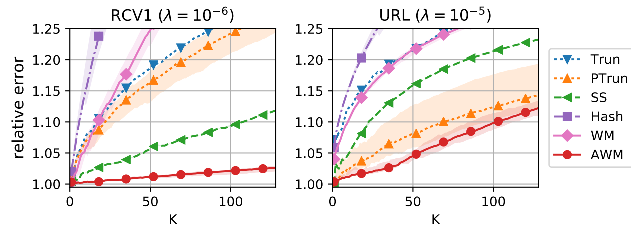

Figure 2: Relative ℓ_2 error of estimated top-K weights vs. true top-K weights for ℓ_2-regularized logistic regression under an 8KB memory budget. The AWM-Sketch achieves lower recovery error across both datasets.

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 frequency between a pair of network links [6]. The AWM-Sketch significantly outperformed a baseline using a pair of Count-Min sketches by a factor of over 4× in recall while using the same memory budget (32KB). These results indicate that linear classifiers can be used effectively to identify significant relative differences over pairs of data streams.

4 DISCUSSION AND RELATED WORK

Sparsity-Inducing Regularization. ℓ_1-regularization is a standard technique for encouraging parameter sparsity in online learning [7, 11, 14, 22]. In practice, it is difficult to *a priori* fix the regularization parameter in order to satisfy a given sparsity budget. Our approach first fixes a memory budget, with the property that the approximation is better for parameter vectors with small ℓ_1-norm.

Feature Hashing. Feature hashing [17, 21] is a technique where the classifier is trained on features that have been hashed to a fixed-width table. The AWM-Sketch with depth 1 can be seen as an interpretability-preserving extension of feature hashing.

Relation to Approximate Heavy-Hitters. The Heavy-Hitters problem, where the goal is to find frequently-occurring items above a given threshold, has been extensively studied in the streaming algorithms literature [4, 5, 16]. Our sketching algorithm for linear classifiers can be seen as a synthesis of sketching techniques for frequent items and algorithms for learning linear classifiers using online convex optimization.

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

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

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

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

[12] 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.

[13] 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.

[14] Brendan McMahan. 2011. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 525–533.

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

[16] 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.

[17] 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.

[18] 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.

[19] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. 2017. Finding Heavily-Weighted Features in Data Streams. *arXiv preprint arXiv:1711.02305* (2017).

[20] 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.

[21] 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.

[22] Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research* 11, Oct (2010), 2543–2596.

[23] 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*.

[24] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch.. In *NSDI*, Vol. 13. 29–42.