Linear and Kernel Classification in the Streaming Model: Improved Bounds for Heavy Hitters

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study linear and kernel classification in the streaming model. For linear classification, we improve upon the algorithm of [1], which solves the ℓ_1 point query problem on the optimal weight vector $w_* \in \mathbb{R}^d$ in sublinear space. We first give an algorithm solving the more difficult ℓ_2 point query problem on w_*, also in sublinear space. We then give an algorithm which solves the related ℓ_2 heavy hitter problem on w_*, in sublinear space and running time. Finally, we give an algorithm which can *deterministically* solve the ℓ_1 point query problem on w_*, with sublinear space, improving upon that of [1]. For kernel classification, if $w_* \in \mathbb{R}^{d^p}$ is the optimal weight vector classifying points in the stream according to their p^{th}-degree polynomial kernel, then we give an algorithm solving the ℓ_2 point query problem on w_* in poly$(\frac{p \log d}{\varepsilon})$ space, and an algorithm solving the ℓ_2 heavy hitter problem in poly$(\frac{p \log d}{\varepsilon})$ space and running time. Note that our space and running time is polynomial in p, making our algorithms well-suited to high-degree polynomial kernels and the Gaussian kernel (approximated by the polynomial kernel of degree $p = \Theta(\log T)$). Our algorithms for kernels are in fact a special case of a more general algorithm we give for low-rank tensor inputs.

1 Introduction

We consider logistic regression, and more generally, linear classification, in the streaming model. In our setting, we are given a dataset consisting of T examples (x_t, y_t), where $t \in [T]$, $x_t \in \mathbb{R}^d$, $y_t \in \{-1, 1\}$. The examples arrive one by one, and moreover, the nonzero coordinates of each example x_t arrive one by one. Our goal is to estimate the weights $w_* \in \mathbb{R}^d$ of the optimal linear classifier for these examples. Here, w_* is defined to be

$$w_* := \text{argmin}_{w \in \mathbb{R}^d} \frac{1}{T} \sum_{t=1}^{T} \ell(y_t w^T x_t) + \frac{\lambda}{2} \|w\|_2^2 \tag{1}$$

where ℓ is a loss function satisfying certain conditions described in Section 1.3 — the prototypical example of ℓ that we consider is the logistic regression loss function — and λ controls the strength of the ℓ_2 regularization. Finally, we assume that d is very large, and we therefore wish to estimate the weights of w_* in space that is sublinear in d. This is important both in settings with devices with limited memory constraints, such as routers or sensors on a network, as well as in machine learning problems with many features. Machine learning problems with a very large number of features arise in many natural language processing tasks, for example — one motivation for [1], the precursor to our work, is that the use of n-gram features when analyzing text data can lead to a very large memory

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

cost. [1] Our goal, and that of [1], is to find features which are the most useful for classification — as pointed out in [1], previously known sketches for compressing classifiers do not achieve this goal.

Formally, we consider the following problems in this work:

Problem 1 (ℓ_2 Point Query)**.** *Let $\varepsilon \in (0,1)$. At any time $t \in [T]$ in the stream, given an arbitrary query $i \in [d]$, the goal is to output $\widehat{w_i}$ such that $|\widehat{w_i} - w_{*,i}| \leq \varepsilon \|w_*\|_2$.*

Problem 2 (ℓ_2 Heavy Hitters)**.** *Let $\varepsilon \in (0,1)$. At any time $t \in [T]$ in the stream, the goal is to output a list $L \subset [d]$ of size at most $O(\frac{1}{\varepsilon^2})$ such that L contains all indices $i \in [d]$ such that $|w_{*,i}| \geq \varepsilon \|w_*\|_2$.*

Problem 3 (ℓ_1 Point Query)**.** *At any time $t \in [T]$ in the stream, given an arbitrary query $i \in [d]$, the goal is to output $\widehat{w_i}$ such that $|\widehat{w_i} - w_{*,i}| \leq \varepsilon \|w_*\|_1$.*

Interpretability is one of the main motivations for the above problem formulations — as argued by [1], finding the largest weights in w_* is equivalent to determining which features are the most important in classification. Also, note that Problem 1 is strictly more difficult than Problem 3 — to see why, note that in the worst case, $\|w_*\|_1$ can be larger than $\|w_*\|_2$ by a $\sqrt{d}$ factor. Thus, for instance, if an algorithm uses poly$(\frac{\log d}{\varepsilon})$ space to solve Problem 3, then it will use at least $d^{O(1)}$ space to solve Problem 1 (by replacing ϵ with $\epsilon/\sqrt{d}$), which is far too large in streaming settings.

1.1 Our Contributions

ℓ_2 point query and heavy hitters (via a reduction to turnstile ℓ_2 point query and heavy hitters) We give efficient algorithms for solving ℓ_2 point query and ℓ_2 heavy hitters. In the approach of [1], a single so-called Countsketch matrix [3] is used to maintain a sketch z_t of the weights. z_t is updated by online gradient descent according to $z_{t+1} \leftarrow (1 - \lambda\eta_t)z_t - \eta_t \ell(y_{t+1} z_t^T R x_{t+1}) R x_{t+1}$, where R is a Countsketch matrix scaled by $\frac{1}{\sqrt{s}}$, where s is the column sparsity of R — by [4], R is a Johnson-Lindenstrauss (JL) transform. See Section 1.3 for more discussion on sketching matrices. To estimate the coordinates of w_*, the recovery procedure of [3] is applied to $\sqrt{s} z_t$. [2] In addition, [1] only obtains the ℓ_1 point query guarantee, since the JL property of R is only applied to show that R preserves the inner products between $e_1, e_2, \ldots, e_d$.

To resolve both of these issues, we decouple the JL matrix from the point query/heavy hitters sketch, i.e., we use a JL matrix R, and a separate sketch S for ℓ_2 point query/heavy hitters in the well-studied turnstile streaming model. First, we maintain z_t, which is updated using online gradient descent as in [1]. In addition, we maintain an additional vector $\widehat{w_t} \in \mathbb{R}^d$, which is updated according to $\widehat{w_{t+1}} \leftarrow (1 - \lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T R x_{t+1}) x_{t+1}$. [3] We do not have enough space to explicitly maintain $\widehat{w_t}$, but we can implicitly store it by giving it the input to a turnstile streaming algorithm for ℓ_2 point query/heavy hitters. We show that $\overline{\widehat{w_T}} = \frac{1}{T}\sum_{t=1}^T \widehat{w_t}$ is close to w_* in ℓ_2 norm, and thus it suffices to solve ℓ_2 point query and ℓ_2 heavy hitters on $\overline{\widehat{w_T}}$. In summary, we give algorithms for ℓ_2 point query and heavy hitters with $O(\frac{1}{\varepsilon^2}\log(dT/\delta))$ space and $1 - \delta$ failure probability.

Deterministic ℓ_1 point query In addition, we show that ℓ_1 point query can be solved *deterministically*, and with a space complexity $O(\frac{1}{\varepsilon^2}\log(d/\delta))$, which is smaller than the $O(\frac{1}{\varepsilon^4}\log^3(d/\delta))$ space complexity of [1]. To obtain a deterministic algorithm, we replace the Countsketch matrix used by [1] with an ε-incoherent matrix (previously applied to streaming problems by [5]). To reduce the space complexity, we change the recovery procedure: to estimate $w_{*,i}$, we multiply the compressed weight vector z on the left by R_i^T, rather than applying the Countsketch recovery procedure of [3].

ℓ_2 point query (via a combined JL/point query sketch) Inspired by our deterministic ℓ_1 point query algorithm, we provide an alternative algorithm for ℓ_2 point query. We observe that the sparse JL transform of [4] can be used not only to preserve norms with high probability (i.e., to satisfy the JL lemma) but also to provide an ℓ_2 point query sketch directly. We do this by modifying the

[1] In [1] it is mentioned that in an experiment on the dataset of [2], "we recorded 47M unique word pairs that co-occur within 5-word spans of text ... [requiring] approximately 560MB of memory."

[2] More precisely, to $\overline{z_T} = \frac{1}{T}\sum_{t=1}^T z_t$ — this averaging is needed to apply standard regret bounds.

[3] The motivation for this update is that it is essentially the OGD update $w_{t+1} \leftarrow (1 - \lambda\eta_t)w_t - \eta_t \ell'(y_{t+1} w_t^T x_{t+1}) x_{t+1}$ we would perform without any sketching, but we replace $\ell'(y_{t+1} w_t^T x_{t+1})$ with $\ell'(y_{t+1} z_t^T R x_{t+1})$ due to space constraints.

standard ℓ_2 point query recovery procedure. Our procedure does not involve any median based operations. Instead, to estimate $w_{*,i}$ given the compressed weight vector z, we simply compute $\langle R_i, z\rangle$. This procedure also avoids the $\sqrt{s}$ rescaling done by [1], and achieves a space complexity of $O(\frac{1}{\varepsilon^2}\log(d/\delta))$. One theoretical advantage of this algorithm over our other ℓ_2 point query algorithm is that the required value of T grows as $\frac{1}{\lambda^4}$, as opposed to $\frac{1}{\lambda^8}$ for our black-box reduction. [4]

Worst-case data order guarantees For all of our algorithms, we do not make assumptions on the order of the x_t, unlike [1]. In [1] the pairs in the set $\{(x_1, y_1), \ldots, (x_T, y_T)\}$ are required to arrive in the stream in a uniformly random order. The following is given in [1] as a heuristic explanation: "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 ... 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." It is perhaps surprising then that we are able to entirely remove this assumption. We do this by showing that Corollary 1 of [6], used by [1], can be replaced with an argument from first principles based on online gradient descent regret bounds.

Classification with tensor inputs We consider a variant of linear classification where the inputs x_t and the weight vector w_* are p-th order tensors (i.e., are vectors in $\mathbb{R}^{d^p}$) and moreover, the x_t have rank at most k, meaning $x_t = \sum_{i=1}^k x_t^{(i,1)} \otimes x_t^{(i,2)} \otimes \ldots \otimes x_t^{(i,p)}$, where the $x_t^{(i,j)} \in \mathbb{R}^d$. This is motivated by applications of tensor regression, for instance in neuroimaging [7, 8], where the covariates have a tensor product structure. Furthermore, the x_t may be of low rank in applications — for instance, in the case $p = 2$, [7] mentions that in [8], tensor regression is performed after principal component analysis is first performed on the x_t. In such a setting, we wish to obtain ℓ_2 point query and heavy hitters algorithms with at most a polynomial dependence on $\log d$ and ε, and moreover a polynomial dependence on p. To achieve this, we use tensor sketching techniques of [9], which develops a sketching matrix $M \in \mathbb{R}^{m\times d^p}$, where $m = \text{poly}(\frac{p\log d}{\varepsilon})$, such that M is a JL matrix, and $Mx^{\otimes p}$ can be computed very efficiently for $x \in \mathbb{R}^d$, without explicitly forming x. [5] Thus, for our ℓ_2 point query, we can use M in the same way we use the sparse JL matrix of [4] in the combined JL/point query sketch above. For ℓ_2 heavy hitters, for each $i \in [p]$, our algorithm is as follows: (1) for each mode $i \in [p]$, we determine the coordinates $j \in [d]$ which comprise more than an ε^2 fraction of the squared ℓ_2 norm of w_*, [6] giving us a list $L_i \subset [d]$ of size at most $O(\frac{1}{\varepsilon^2})$, for each $i \in [p]$, and then (2) we find the (at most $O(1/\varepsilon^2)$) indices $(i_1, \ldots, i_p)$ of w_* in $[d]^p$ such that $|w_{*,(i_1,i_2,\ldots,i_p)}| \geq \varepsilon\|w_*\|_2$. We do step (2) using the L_i, by inductively constructing *prefixes* of these coordinates, one mode at a time. For each $i \in [p]$, we build an auxiliary data structure which can estimate the value of any index prefix of length i, also using the sketching matrix of [9]. Both of these algorithms have $\text{poly}(\frac{p\log d}{\varepsilon})$ space and query time, and $\text{poly}(\frac{p\log d}{\varepsilon})\sum \text{nnz}(x_t^{(i,j)})$ update time.

Both our point query and heavy hitters algorithms give significant savings in update time when compared to just using standard ℓ_2 point query/heavy hitters algorithms. To see why, note that if we use our black-box reduction from above, then the update to $\widehat{w_t}$ is

$$\widehat{w_{t+1}} \leftarrow (1-\lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T M x_{t+1}) \sum_{i=1}^k x_t^{(i,1)} \otimes x_t^{(i,2)} \otimes \ldots \otimes x_t^{(i,p)} \tag{2}$$

If we wished to use a standard ℓ_2 heavy hitters algorithm on $\widehat{w_t}$, then this requires explicitly forming $x_t^{(i,1)} \otimes x_t^{(i,2)} \otimes \ldots \otimes x_t^{(i,p)}$ — if the $x_t^{(i,j)}$ are dense, then this would require at least d^p running time, as opposed to our algorithm, which requires $\text{poly}(\frac{p\log d}{\varepsilon}) \cdot pkd$ running time — even when $p = 2$, if k is small, then this is a significant improvement.

Kernel classification Kernel logistic regression (KLR) is a well-known classification method in the field of statistical learning, see e.g., [10] and its many citations. We obtain the first results for

[4] We use OGD regret bounds to show that the estimation error of our algorithms involves a term that is proportional to $\frac{1}{T^{1/4}}$, for both algorithms. Thus, for the error to be at most $\varepsilon\|w_*\|_2$ or $\varepsilon\|w_*\|_1$, T must be at least a certain value.

[5] The running time of this computation is $\text{nnz}(x) \cdot \text{poly}(p\frac{\log d}{\varepsilon})$.

[6] Formally, we want to find $j \in [d]$ such that the slice of w_* consisting of those coordinates, whose index in the ith mode is j, has norm at least $\varepsilon\|w_*\|_2$.

finding the large weights of a classifier in the kernel space for the polynomial and Gaussian kernels. A succinct summary of the classifier, such as its list of heavy hitters with their approximate values, is especially important for kernel classification, since the dimension of the kernel space can be much larger than d, and in the case of the Gaussian kernel, even infinite. In this setting, for the polynomial kernel, classification is done using $x_t^{\otimes p}$ to predict y_t — thus, this is a special case of the setting where x_t is a tensor of rank at most k, discussed above. As in [9], we can approximate the Gaussian kernel via a Taylor expansion, using a polynomial kernel of degree $O(\log T)$. Note that if $(i_1, i_2, \ldots, i_p)$ is an index in $[d]^p$ and $(j_1, j_2, \ldots, j_p)$ is a re-ordering of $(i_1, i_2, \ldots, i_p)$, then one may want to consider $x_{i_1} \ldots x_{i_p}$ and $x_{j_1} \ldots x_{j_p}$ to be the same feature. To get around this, suppose $(i_1, i_2, \ldots, i_p)$ has a Hamming distance of at most c from the set of indices of the form $(i, i, \ldots, i)$ and it is an ε-heavy hitter according to the new notion of heavy hitters (precisely defined later in the section on kernel classification). Then, if we apply our algorithms from the low-rank tensor setting with an accuracy of $\frac{\varepsilon}{c!}$, $(i_1, i_2, \ldots, i_p)$ will be detected as a heavy hitter. An interesting open question is whether this can be done in poly$(\frac{p \log d}{\varepsilon})$ space even when c is allowed to be equal to p — we leave this question to future work.

Experiments We empirically compare our second algorithm for ℓ_2 point query (the combined JL/point query sketch) with the WM-Sketch algorithm of [1]. [7] In addition to the theoretical advantages discussed above, our experimental results show that our combined JL/point query sketch leads to improved performance in estimating w_*.

1.2 Related work

There is a large body of work on finding the heavy hitters in a data stream. For a survey, see, e.g., [11]. Of particular relevance to this work is the CountSketch algorithm of [3] for finding ℓ_2 heavy hitters. We note that [12, 13] improve the memory of the algorithm of [3] by a logarithmic factor, but do not handle negative updates, which may arise in our setting. We also need deterministic algorithms for finding ℓ_1 heavy hitters, and we use the algorithms of [5] which use ϵ-incoherent matrices, and improve upon the earlier work of Ganguly [14]. We note that the CountMin algorithm of [15] also achieves the ℓ_1 heavy hitter guarantee, though it is randomized, while here we seek a stronger deterministic guarantee. Indeed, for randomized algorithms, we can achieve the stronger ℓ_2 heavy hitter guarantee.

There exist several works on applying sketching and dimensionality reduction techniques to compressing classifiers. One such work, as mentioned above, is [1], which solves ℓ_1 point query on w_* in the streaming model, and achieves $O(\frac{1}{\varepsilon^4} \log^3(d))$ space. Unlike [1], we give an algorithm with provable guarantees for finding the at most $\frac{1}{\varepsilon^2}$ heavy hitters in *sublinear* time, and we solve the stronger ℓ_2 point query and heavy hitters problems in addition to ℓ_1 point query. There exist several other works which have a different focus from ours. One such related work [16] proposes the MISSION algorithm. The focus of this work is somewhat different from ours. The MISSION algorithm finds a k-sparse solution for least-squares regression in low space, using Countsketch. [16] modifies the SGD algorithm: in each iteration, a uniformly random training example is selected and the SGD update is given to a Countsketch data structure — then, Countsketch is used to select the top k features, and the vector with these k non-zero coordinates is used for the next SGD update. The key differences between our ℓ_2 point query/heavy hitters algorithms and MISSION is that (1) MISSION is a modified version of SGD, while our algorithms are for the streaming model, where the x_t can arrive in a worst-case ordering, (2) MISSION focuses on convergence of the iterates β^t to a k-sparse vector β^*, while in the analyses of our ℓ_2 point query/heavy hitters algorithms, we desire/show convergence of $\widehat{w_T}$ to w_* up to additive error $\varepsilon \|w_*\|_2$, for a potentially arbitrary w_*, and (3) our gradient updates are thus different, as we update $\widehat{w_{t+1}} \leftarrow (1 - \lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T R x_{t+1}) x_{t+1}$, i.e., our update is very similar to GD and does not involve any truncation using Countsketch. Another related work [17] proposes the BEAR algorithm, in fact showing that the L-BFGS algorithm can be performed in low memory using CountSketch — the focus of [17] is similar to MISSION [16] and thus different from ours.

Two other works [18, 19] are on reducing the dimensionality of the feature space using hashing — note that these works do not consider the combination of gradient descent with dimensionality

[7] We use an implementation of their algorithm that is publicly available at `https://github.com/stanford-futuredata/wmsketch`.

reduction, and also have a different focus from our work. [20] studies a more general setting of non-convex optimization/neural networks, but the focus is on compressing the *auxiliary* variables in optimization algorithms such as ADAM, ADAGRAD, etc. [21] is concerned with machine learning problems *directly making use of the compressed weights*, rather than recovering estimates of the optimal weight vectors.

Finally, to our knowledge, our work is the first to consider the problems of linear classification with tensor inputs, and kernel classification, in the streaming models, with the focus of recovering the top weights of these classifiers.

1.3 Preliminaries

Our setting is similar to that of [1]. We are given a loss function $\ell : \mathbb{R} \to \mathbb{R}$, training examples $\{(x_t, y_t)\}_{t\in[T]}$, and $\lambda > 0$, and we define $w_* := \operatorname{argmin}_{w\in\mathbb{R}^d} \frac{1}{T}\sum_{t=1}^{T} \ell(y_t w^T x_t) + \frac{\lambda}{2}\|w\|_2^2$. We assume ℓ is convex and β-strongly smooth, and H-Lipschitz continuous. We also assume that $\|x_t\|_2 \leq 1$ for all $t \in [T]$. [8] Finally, we assume there exists a constant $\tau > 0$ such that $\|w_*\|_2 > \tau$. [9] In addition, we make use of the regret bounds for online gradient descent, specifically Theorem 3.1 of [22]. [10] Throughout the paper, we use a learning rate of $\eta_t = \frac{D}{G\sqrt{t}}$. [11]

2 ℓ_2 Point Query and Heavy Hitters

First, we define a "sketched" version of the loss function, as in [1]: define $\widehat{L}(z) = \frac{1}{T}\sum_{t=1}^{T} \ell(y_t z^T R x_t) + \frac{\lambda}{2}\|z\|_2^2$ where R is a sparse Johnson-Lindenstrauss matrix. Let us recall the properties of sparse JL matrices: [12]

Theorem 2.1 (Sparse JL Matrices [4])**.** *Let $d \in \mathbb{N}$, and $\varepsilon, \delta \in (0, 1)$. Then, there exists a distribution on matrices $S \in \mathbb{R}^{k\times d}$, where $k = \Theta(\frac{1}{\varepsilon^2}\log(1/\delta))$ and S has $s = \Theta(\frac{1}{\varepsilon}\log(1/\delta))$ nonzero entries per column, such that for any $x \in \mathbb{R}^d$, with probability $1 - \delta$, $(1-\varepsilon)\|x\|_2 \leq \|Sx\|_2 \leq (1+\varepsilon)\|x\|_2$.*

If $z_* := \operatorname{argmin} \widehat{L}(z)$, then z_* is a compressed version of w_* in the following sense:

Theorem 2.2 (Batch Setting)**.** *Let $\varepsilon, \delta \in (0, 1)$, and suppose ℓ is β-smooth and $\|x_t\|_2 \leq 1$ for all t. Define $w_* = \operatorname{argmin} L(w)$ and $z_* = \operatorname{argmin} \widehat{L}(z)$. If R is a sparse JL matrix with $O(\frac{\log(dT/\delta)}{\varepsilon^2} \cdot \frac{\beta}{\lambda})$ rows, then with probability $1 - \delta$ over R, $\|z_* - Rw_*\|_2 \leq \varepsilon\|w_*\|_2$.*

The proof of Theorem 2.2 is in the appendix. It uses a primal-dual argument that is similar to the one used to prove Theorem 1 of [1]. The main difference is that [1] only shows that $\|z_* - Rw_*\|_2 \leq \varepsilon\|w_*\|_1$. This is because they only apply the JL property of R to the vectors $e_1, e_2, \ldots, e_d$. To prove Theorem 2.2, we modify their analysis by also using the JL property of R to additionally show that the inner products $\langle x_t, w_*\rangle$ and $\langle e_i, w_*\rangle$ are well-approximated by $\langle Rx_t, Rw_*\rangle$ and $\langle Re_i, Rw_*\rangle$ respectively, for all $i \in [d]$, and $t \in [T]$.

Our algorithm proceeds by a reduction to standard ℓ_2 point query and heavy hitters, in the turnstile streaming model. Let us recall the definition of the turnstile streaming model, and the best known results for these two problems in the turnstile model.

Definition 2.3 (Turnstile Streaming Model)**.** *In the turnstile streaming model, the input is a vector $v \in \mathbb{R}^n$. Updates to this vector arrive one at a time in a stream, and are each of the form (i, Δ) for some $i \in [n]$ and $\Delta \in \mathbb{R}$ — each such update signifies that v is updated according to $v_i \leftarrow v_i + \Delta$. For ℓ_2 point query in the turnstile model, queries $i \in [d]$ should be answered with an estimate $\widehat{v}_i$ of v_i*

[8] We note that the proofs can be generalized to $\|x_t\|_2 \leq B$ for some $B > 0$, but we focus on $\|x_t\|_2 \leq 1$ for simplicity and ease of presentation.

[9] This last assumption is not explicitly made in [1], but in Theorem 2 of [1], one of the hypotheses is that T is at least $\frac{1}{\|w_*\|_1^2}$ (omitting other dependencies). Since w_* itself may depend on $x_1, x_2, \ldots, x_T$, and thus T, we explicitly make this assumption to prevent circularity. This assumption is necessary for making use of OGD regret bounds, to ensure $\|w_*\|_2$ does not decrease faster than the average regret.

[10] Available at `https://arxiv.org/pdf/1909.05207.pdf`.

[11] We do not discuss η_t further in the main paper, though it is relevant in the proofs of certain theorems. D is an upper bound on the ℓ_2 norm $\|w_*\|_2$ and G is an upper bound on the gradient norms.

[12] [4] shows that vertically stacking $O(\frac{1}{\varepsilon}\log(1/\delta))$ Countsketch matrices [3] yields a sparse JL matrix.

such that $|\widehat{v_i} - v_i| \leq \varepsilon \|v\|_2$. For ℓ_2 heavy hitters in the turnstile model, queries should be answered with a list L of length $O(\frac{1}{\varepsilon^2})$ containing all $i \in [d]$ such that $|v_i| \geq \varepsilon \|v\|_2$.

Theorem 2.4 (ℓ_2 Point Query in the Turnstile Model [3], Theorem 2 of [23], Lemma 1 of [24])**.** *There exists an algorithm for turnstile ℓ_2 point query with space complexity $O(\frac{1}{\varepsilon^2}\log(1/\delta))$, update time $O(\log(1/\delta))$ and query time $O(\log(1/\delta))$, and failure probability $1-\delta$.*

Theorem 2.5 (Turnstile ℓ_2 Heavy Hitters [25])**.** *There is an algorithm for turnstile ℓ_2 heavy hitters with $O(\frac{1}{\varepsilon^2}\log(n/\delta))$ space complexity, $O(\log(n/\delta))$ update time and $O(\frac{1}{\varepsilon^2} poly(\log(n/\delta)))$ query time, and failure probability $1-\delta$.*

For turnstile ℓ_2 point query and heavy hitters, the algorithms described above use sketching data structures S, which are linear maps: $S(cv)$ can be obtained by scaling each coordinate of Sv by c, and $S(v_1+v_2)$ can be obtained by summing the corresponding coordinates of Sv_1 and Sv_2. This is a useful property for our algorithm, as we will see.

Algorithm 1 In this algorithm, we give our black-box reduction to ℓ_2 point query or heavy hitters. In this algorithm, $\mathcal{A}$ denotes a linear sketching data structure for ℓ_2 point query or ℓ_2 heavy hitters.

function INITIALIZATION()
  $R \in \mathbb{R}^{k \times d}$ is a sparse JL matrix with $k = O(\frac{\log(d/\delta)}{\varepsilon^2} \cdot \max(1, \frac{\beta}{\lambda}))$ rows.
  $z_0 \in \mathbb{R}^k$ is set to $0 \in \mathbb{R}^k$.
  The contents of the sketch $\mathcal{A}$ are set to $0 \in \mathbb{R}^d$.
  $t \leftarrow 0$
end function

function UPDATE(x_t, y_t)
  $\widetilde{x_t} \leftarrow Rx_t$
  $\tau \leftarrow z_{t-1}^T \widetilde{x_t}$
  $z_t \leftarrow (1-\lambda\eta_t)z_{t-1} - \eta_t \ell'(y_t\tau)\widetilde{x_t}$
  Rescale the contents of $\mathcal{A}$ by $(1-\lambda\eta_t)$.
  For each nonzero coordinate $i \in [d]$ of x_t, update $\mathcal{A}$ according to $(i, -\eta_t \ell'(y_t\tau)x_{t,i})$.
end function

function QUERY()
  $\overline{\mathcal{A}_T} \leftarrow \frac{1}{T}\sum_{t=1}^T \mathcal{A}_t$
  Return QUERY($\overline{\mathcal{A}_T}$)
end function

Our algorithm is shown in Algorithm 1. Here we give further explanation. First we maintain z_t which is updated according to online gradient descent [13] using the sketched examples Rx_t, according to $z_{t+1} \leftarrow (1-\lambda\eta_t)z_t - \eta_t \ell'(y_{t+1} z_t^T Rx_{t+1})Rx_{t+1}$. In addition, we implicitly maintain a vector $\widehat{w_t} \in \mathbb{R}^d$ which is updated according to $\widehat{w_{t+1}} \leftarrow (1-\lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T Rx_{t+1})x_{t+1}$. [14] Clearly we cannot maintain $\widehat{w_t}$ explicitly, but we give it as input to the linear sketch which we refer to as $\mathcal{A}$ in Algorithm 1. To perform the online gradient descent update, given $\mathcal{A}\widehat{w_t}$, we can first rescale it by $(1-\lambda\eta_t)$, and then for each nonzero coordinate $i \in [d]$ of x_t, we can give $(i, -\eta_t \ell'(y_t z_t^T Rx_t)x_i)$ as a turnstile update to $\mathcal{A}\widehat{w_t}$. For ℓ_2 point query, $\mathcal{A}$ is a Countsketch matrix, and for ℓ_2 heavy hitters it is the Expander Sketch data structure of [25]. To justify this use of $\overline{\widehat{w_T}}$ for ℓ_2 point query heavy hitters, we show this is a good approximation of w_*:

Theorem 2.6 (Approximating w_* with $\overline{\widehat{w_T}}$)**.** *Let $\varepsilon, \delta \in (0,1)$, and suppose ℓ is β-smooth and $\|x_t\|_2 \leq 1$ for all t. Suppose ℓ is H-Lipschitz continuous. Define $w_* = \operatorname{argmin} L(w)$. Suppose R is a sparse JL matrix with $O(\frac{\log(dT/\delta)}{\varepsilon^2} \cdot \max(1, \frac{\beta}{\lambda}))$ rows. Suppose that there is a constant $\tau > 0$ such that $\|w_*\|_2 \geq \tau$. If $\widehat{w_t}$ is updated according to $\widehat{w_{t+1}} \leftarrow (1-\lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T Rx_{t+1})x_{t+1}$, and $\overline{\widehat{w_t}} = \frac{1}{T}\sum_{t=1}^T \widehat{w_t}$, then $\|\overline{\widehat{w_T}} - w_*\|_2 \leq \varepsilon \|w_*\|_2$ as long as $T \geq H^8/(\lambda^8 \varepsilon^4 \tau^4)$.*

[13] Note that we can skip the projection step used in [1], since even without projection, $\|z_{t+1}\|_2 \leq \frac{H}{\lambda}$ by the triangle inequality and induction.

[14] which is an approximation of the weight vector obtained by "unsketched" online gradient descent

Theorem 2.7 (Algorithm for ℓ_2 Point Query and Heavy Hitters for Linear Classification)**.** *Let $\varepsilon, \delta \in (0,1)$, and suppose ℓ is β-smooth and $\|x_t\|_2 \leq 1$ for all $t \in [T]$. In addition, suppose ℓ is H-Lipschitz continuous. Finally, suppose there is a constant $\tau > 0$ such that $\|w_*\|_2 \geq \tau$. Then, there is an algorithm for ℓ_2 point query on w_* which has $O(\frac{1}{\varepsilon^2}\log(dT/\delta)(1+\frac{\beta}{\lambda}))$ space complexity, $O(\frac{1}{\varepsilon}\log(dT/\delta)(1+\sqrt{\frac{\beta}{\lambda}}))$ update time and $O(\log(1/\delta))$ query time. In addition, there is an algorithm for ℓ_2 heavy hitters on w_* which has $O(\frac{1}{\varepsilon^2}\log(dT/\delta)(1+\frac{\beta}{\lambda}))$ space complexity, $O(\frac{1}{\varepsilon}\log(dT/\delta)(1+\sqrt{\frac{\beta}{\lambda}}))$ update time and poly$(\frac{\log(d/\delta)}{\varepsilon})$ query time.*

The proofs of Theorems 2.6 and 2.7 are given in the supplementary. Note that the query times are simply those of CountSketch [3] / ExpanderSketch [25] respectively.

3 Deterministic ℓ_1 Point Query

We also give a simple deterministic algorithm for ℓ_1 point query with sublinear space. In this section, as in [1], we make the additional assumption that $\|x_t\|_1 \leq \gamma$. Our algorithm for ℓ_1 point query is similar to that of [1]. However, R is now an ε-incoherent matrix:

Theorem 3.1 (ε-Incoherent Matrices [5])**.** *Let $n \in \mathbb{N}$ and $\varepsilon > 0$. Then, there exists a matrix $A \in \mathbb{R}^{m\times n}$, where $m = O(\frac{1}{\varepsilon^2}\min(\log n, (\frac{\log n}{\log\log n + \log 1/\varepsilon})^2))$, such that for all $i \in [n]$, $\|A_i\|_2 = 1$, and for all distinct $i, j \in [n]$, $\langle A_i, A_j\rangle \leq \varepsilon$. Moreover, A can be constructed deterministically in poly(n) time. A does not need to be stored explicitly, and each column can be generated on demand in low space (we describe this in the supplementary).*

Let us describe the algorithm — we provide the explicit pseudocode in the appendix. We use an incoherent matrix R as the sketch, and we store z_t as in our black-box reduction — as before z_t is updated according to $z_{t+1} \leftarrow (1-\lambda\eta_t)z_t - \eta_t \ell'(y_{t+1} z_t^T R x_{t+1}) R x_{t+1}$. To estimate the weights w_*, we directly use $\overline{z_T} = \frac{1}{T}\sum_{t=1}^T z_t$ for recovery. To estimate $w_{*,i}$, we can simply compute $\langle R_i, \overline{z_T}\rangle$ — this is the recovery procedure for standard ℓ_1 point query described in [5]. As in [1], we first analyze this algorithm in the batch setting — we bound the error when recovering w_* using z_*, the minimizer of the sketched loss mentioned in the previous section.

Theorem 3.2 (Batch Setting with Incoherent Matrices)**.** *Suppose ℓ is β-smooth, and $\|x_t\|_1 \leq \gamma$ for all $t \in [T]$. Define $w_* := \operatorname{argmin} L(w)$ and $z_* = \operatorname{argmin} \widehat{L}(z)$. Then, $\|R^T z_* - w_*\|_\infty \leq \varepsilon\|w_*\|_1$ as long as R is an incoherent matrix with $O\Big(\frac{\log d}{\varepsilon^2}\cdot\max\Big(1, \frac{2\gamma^2\beta}{\lambda}\Big)\Big)$ rows.*

Using regret bounds for online gradient descent, we are able to obtain the following result, the proof of which is in the appendix.

Theorem 3.3 (Analysis of ℓ_1 Point Query on w_* with Incoherent Matrix)**.** *Suppose $\|x_t\|_1 \leq \gamma$ and $\|x_t\|_2 \leq 1$ for all $t \in [T]$. Assume ℓ is β-smooth, and H-Lipschitz continuous. If R is an incoherent matrix with $O(\frac{\log d}{\varepsilon^2}\cdot\max(1, \frac{\gamma^2\beta}{\lambda}))$ rows, then $\|R^T\overline{z} - w_*\|_\infty \leq \varepsilon\|w_*\|_1$, as long as $T \geq O\Big(\frac{H^4(1+\sqrt{\varepsilon}\gamma)^4}{\lambda^4\varepsilon^4\|w_*\|_1^4}\Big)$.*

4 ℓ_2 Point Query: A JL matrix and Point Query Sketch in One

Inspired by the algorithm of the previous section, we propose an alternative algorithm for ℓ_2 point query. Here, we modify that algorithm so that R is a JL matrix with $O(\frac{\log(dT/\delta)}{\varepsilon^2}\cdot\max(1, \beta/\lambda))$ rows, and leave all other aspects, including the recovery procedure, the same. [15] This suffices to obtain the ℓ_2 guarantee. The theoretical guarantees are as follows — note that the space complexity is the same as Algorithm 1, but the required T is smaller by a λ^4 factor:

Theorem 4.1 (ℓ_2 Point Query using only a JL Matrix)**.** *Let $\varepsilon, \delta \in (0,1)$ and suppose ℓ is β-smooth, H-Lipschitz, and $\|x_t\|_2 \leq 1$ for all $t \in [T]$. Suppose there is some $\tau > 0$ independent of T such that $\|w_*\|_2 \geq \tau$. Suppose R is a sparse JL matrix having at least $O(\frac{\log(dT/\delta)}{\varepsilon^2}\cdot\max(1, \frac{\beta}{\lambda}))$ rows and*

[15] Algorithm is explicitly stated in the appendix.

$O(\frac{\log(d/\delta)}{\varepsilon} \cdot \max(1, \sqrt{\frac{\beta}{\lambda}}))$ *nonzero entries per column. Finally, suppose* $T \geq \Omega(\frac{H^4}{\lambda^4 \varepsilon^4 \tau^4})$*. Then, with probability* $1-\delta$*,* $\|R^T \overline{z} - w_*\|_\infty \leq \varepsilon \|w_*\|_2$*.*

Proof. We give a proof sketch — the full proof is in the appendix. Note that Theorem 2.2 can be used to show that $\|R^T z_* - w_*\|_\infty \leq \varepsilon \|w_*\|_2$, using a similar argument to that in Theorem 3.2. Using online gradient descent regret bounds then gives the desired result. □

5 Low Rank Tensor Classification and Kernel Classification

We consider ℓ_2 point query and heavy hitters in the case where the $x_t \in \mathbb{R}^{d^p}$ are p^{th} order tensors of rank at most k, provided as the sum of k rank-1 tensors. Let us recall the sketch of [9]:

Theorem 5.1 (Recursive Tensor Sketch — Follows from the Proof of Theorem 2 of [9])**.** *Let* $n, p, d \in \mathbb{N}$*, and* $\varepsilon, \delta > 0$*. Then, there is a distribution on random matrices* $M \in \mathbb{R}^{m \times d^p}$*, with* $m = \Theta(\frac{p \log(1/\varepsilon\delta)^3}{\varepsilon^2})$*, such that for all* $x, y \in \mathbb{R}^{d^p}$*,* $\Pr_M \left[|\langle Mx, My \rangle - \langle x, y \rangle| \geq \varepsilon \|x\|_2 \|y\|_2 \right] \leq 1 - \delta$ *and moreover, for all* $x_1, x_2, \ldots, x_p \in \mathbb{R}^d$*,* $M(x_1 \otimes x_2 \otimes \ldots \otimes x_p)$ *can be computed in* $O\Big(\frac{p^{1/2} \log(1/\delta)}{\varepsilon} \sum_{i=1}^p nnz(x_i) + pm \log(m)\Big)$ *time.*

Using Theorem 5.1, we can extend the techniques of the previous sections to the case where x_t are p-th order tensors of rank at most k. If $R \in \mathbb{R}^{m \times d^p}$ is the random matrix of Theorem 5.1, then we can store $z_t \in \mathbb{R}^k$, which is updated according to $z_{t+1} \leftarrow (1 - \lambda\eta_t) z_t - \eta_t \ell'(y_t z_t^T R x_t) R x_t$. Since we are given x_t as $\sum_{i=1}^k x_t^{(i,1)} \otimes x_t^{(i,2)} \otimes \ldots \otimes x_t^{(i,p)}$, Rx_t can be computed in $\text{poly}(p \log(d/\delta)/\varepsilon) \sum_{i=1}^k \sum_{j=1}^p \text{nnz}(x_t^{(i,j)})$ time, and without explicitly forming x_t. [16] Thus we obtain an algorithm for ℓ_2 point query, based on the ℓ_2 point query algorithm of the previous section. [17] (We could have also used an algorithm based on the black-box reduction — however, the point query recovery procedure would again be left multiplication by the tensor sketch described in Theorem 5.1.)

Theorem 5.2 (Tensor Classification Point Query)**.** *Let* $\varepsilon, \delta \in (0, 1)$ *and suppose* ℓ *is* β*-smooth and* H*-Lipschitz, and* $\|x_t\|_2 \leq 1$ *for all* $t \in [T]$*. Let* R *be the sketch of [9] with poly*$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta))$ *rows. Suppose* $\tau > 0$ *is a constant independent of* T *such that* $\|w_*\|_2 \geq \tau$*. Finally, suppose* $T \geq \Omega(\frac{H^4}{\lambda^4 \varepsilon^4 \tau^4})$*. Then, with probability* $1-\delta$*,* $\|R^T \overline{z_T} - w_*\|_\infty \leq \varepsilon \|w_*\|_2$*. Thus, there is an algorithm for* ℓ_2 *point query on* w_* *with the space complexity and query time poly*$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta))$*, and update time poly*$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta)) \sum_{i=1}^k \sum_{j=1}^p nnz(x_t^{(i,j)}) \leq kd \cdot poly((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta))$*.*

Next we give our algorithm for ℓ_2 heavy hitters on $w_* \in \mathbb{R}^{d^p}$. First, we make use of the black-box reduction from Section 2 to simplify the algorithm. If z_t is the compressed weight vector, then $z_{t+1} \leftarrow (1 - \lambda\eta_t) z_t - \eta_t \ell'(y_{t+1} z_t^T R x_{t+1}) R x_{t+1}$ can be easily performed since Rx_{t+1} can be quickly computed. Now we must deal with $\widehat{w_t}$, implicitly updated according to $\widehat{w_{t+1}} \leftarrow (1 - \lambda\eta_t)\widehat{w_t} - \eta_t \ell'(y_{t+1} z_t^T R x_{t+1}) y_{t+1} x_{t+1}$. We give the following algorithm which can efficiently deal with such updates and compute the heavy hitters of $\widehat{w_t}$.

We briefly describe the algorithm in words, and give more formal pseudocode in the appendix. Here our sketch is given $v \in \mathbb{R}^{d^p}$ in the form of updates of the form $\sum_{i=1}^k x_t^{(i,1)} \otimes \ldots \otimes x_t^{(i,p)}$. Our algorithm for finding the ε ℓ_2 heavy hitters of v works in two stages. In the first stage, on each mode $i \in [p]$, we find all the coordinates $j \in [d]$ on the ith mode such that the total ℓ_2 weight of all coordinates of v which have coordinate j in the ith mode is at least $\varepsilon \|w_*\|_2$. For this, we maintain two auxiliary data structures $\textsc{CompressOtherModes}^{(i)}$ and $\textsc{OneModeSketch}^{(i)}$. $\textsc{CompressOtherModes}^{(i)}$ uses the sketch of [9] to compress all modes but i into a $\text{poly}(p \log(d/\delta))$ dimensional vector

[16] $R(x_t^{(i,1)} \otimes x_t^{(i,2)} \otimes \ldots \otimes x_t^{(i,p)})$ can be computed in one pass over the nonzero entries of $x_t^{(i,1)}, x_t^{(i,2)}, \ldots, x_t^{(i,p)}$. We can see this as follows: by the construction of [9], R is essentially a tree of sketching matrices, with $2p+1$ nodes. At the base of this tree are matrices $R^1, \ldots, R^p$, which can be separately applied to $x_t^{(i,1)}, \ldots, x_t^{(i,p)}$ respectively — from this point, only $\text{poly}(\frac{p \log(d/\delta)}{\varepsilon})$ space is needed to finish the computation.

[17] We explicitly state the algorithm in the appendix.

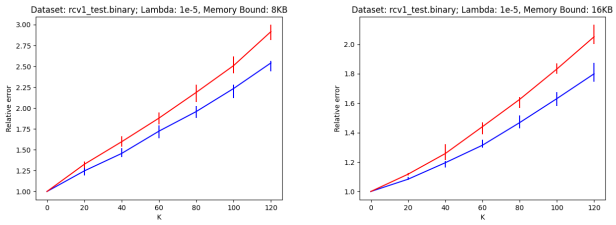

Figure 1: Blue is combined JL/HH sketch for ℓ_2 point query, red is the algorithm of [1]. We report the median across 5 trials, with the error bars denoting the smallest and highest relative errors across those trials.

(resulting in a $d \cdot \text{poly}(p \log(d/\delta))$ dimensional vector). ONEMODESKETCH$^{(i)}$ then finds all the ε heavy hitters on the ith mode using this vector as the input — the justification for this is that the sketch of [9] is a JL matrix. In the second stage, we find all heavy hitters $(j_1, j_2, \ldots, j_p)$. As mentioned before, we construct them iteratively. At the ith iteration for $i \in [p]$, we maintain a list L of all candidate prefixes $(j_1, j_2, \ldots, j_i)$. From the first stage we obtain a list L_{i+1} of the heavy coordinates on the $(i+1)$-th mode. To determine the top prefixes of length $i+1$ (out of all the prefixes in $L \times L_{i+1}$) we use two auxiliary data structures for this: COMPRESSSUFFIX$^{(i+1)}$, which compresses the modes after $i+1$ into a low dimensional vector while preserving their ℓ_2 norms, and PREFIXPOINTQUERY$^{(i+1)}$, which uses the ideas in our tensor ℓ_2 point query algorithm to estimate the value of each *prefix* of length $i+1$.

Theorem 5.3 (Tensor ℓ_2 Heavy Hitters Algorithm)**.** *Let $\varepsilon, \delta \in (0,1)$. Let $v \in \mathbb{R}^{d^p}$, which can be incrementally updated by a rank-k tensor. Then, the algorithm discussed above returns a list containing all $(i_1, i_2, \ldots, i_p) \in [d]^p$ such that $|v(i_1, \ldots, i_p)| \geq \varepsilon \|v\|_2$. The space complexity is poly$(\frac{p \log(d/\delta)}{\varepsilon})$, the query running time is poly$(\frac{p \log(d/\delta)}{\varepsilon})$, and the update time for a rank-k tensor x_t is poly$(\frac{p \log(d/\delta)}{\varepsilon}) \sum_{i=1}^{k} \sum_{j=1}^{p} nnz(x_t^{(i,j)})$.*

Theorem 5.4 (Tensor Classification ℓ_2 Heavy Hitters)**.** *Let $\varepsilon, \delta \in (0,1)$. Let $w_* \in \mathbb{R}^{d^p}$ be the optimal weight vector classifying the rank-k tensors $x_t \in \mathbb{R}^{d^p}$. Suppose ℓ is β-smooth and H-Lipschitz and $\|x_t\|_2 \leq 1$ for all t. Suppose there is a constant $\tau > 0$ such that $\|w_*\|_2 \geq \tau$. Then, there is an algorithm which, for $T \geq \frac{H^8}{\lambda^8 \varepsilon^4 \tau^4}$, finds the ε ℓ_2 heavy hitters of w_*. The space complexity and query time are poly$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta)))$. The update time is poly$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta)) \sum_{i=1}^{k} \sum_{j=1}^{p} nnz(x_t^{(i,j)}) \leq kd \cdot$ poly$((1 + \frac{\beta}{\lambda}) \frac{p}{\varepsilon} \log(dT/\delta))$.*

We give corollaries for heavy hitters for the polynomial kernel of degree p and Gaussian kernel in the supplementary.

6 Experiments

In this section, we compare our simplified ℓ_2 point query algorithm empirically with the ℓ_1 point query algorithm of [1], in terms of performance on weight recovery (how well they estimate w_*). We use a similar relative error metric to that used in Subsection 7.2 of [1] — to estimate how well the algorithms recover the top K weights, let w^K be the estimates obtained by the algorithms, we compute $\frac{\|w^K - w_*\|_2}{\|w_*^K - w_*\|_2}$ (where w^K is the K-sparse vector closest to w). We make a slight modification, using w_*^D in place of w_*, for some $D \gg K$. [18] We evaluate these algorithms on the three binary classification datasets used in [1], reporting results on the RCV1 dataset [26] here and discussing the other datasets in the supplementary materials. We also evaluate on several values of λ, and several memory limits — for each memory limit, we use the CountSketch dimensions that are noted in Table 1 of [1] as best performing configurations (and use the same dimensions for JL). We show two plots in Figure 1. Observe that for these settings, using a JL-based recovery procedure rather than taking medians led to lower error.

[18] Exact value is noted in the supplementary.

References

[1] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 757–772. ACM, 2018.

[2] Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

[3] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[4] Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014.

[5] Jelani Nelson, Huy L. Nguyên, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 627–638. Springer, 2012.

[6] Ohad Shamir. Without-replacement sampling for stochastic gradient methods: Convergence results and application to distributed optimization. *CoRR*, abs/1603.00570, 2016.

[7] Hua Zhou, Lexin Li, and Hongtu Zhu. Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502):540–552, 2013. PMID: 24791032.

[8] Brian S. Caffo, Ciprian M. Crainiceanu, Guillermo Verduzco, Suresh Joel, Stewart H. Mostofsky, Susan Spear Bassett, and James J. Pekar. Two-stage decompositions for the analysis of functional connectivity for fmri with application to alzheimer's disease risk. *NeuroImage*, 51(3):1140–1149, 2010.

[9] Thomas D Ahle, Michael Kapralov, Jakob BT Knudsen, Rasmus Pagh, Ameya Velingker, David P Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 141–160. SIAM, 2020.

[10] Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics*, 14(1):185–205, 2005.

[11] David P Woodruff. New algorithms for heavy hitters in data streams. *arXiv preprint arXiv:1603.01733*, 2016.

[12] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 740–753, 2016.

[13] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 361–376, 2017.

[14] Sumit Ganguly and Anirban Majumder. Cr-precis: A deterministic summary structure for update data streams. In *IntErnational Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 48–59. Springer, 2007.

[15] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

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

[17] Amirali Aghazadeh, Vipul Gupta, Alex DeWeese, O. Ozan Koyluoglu, and Kannan Ramchandran. Bear: Sketching bfgs algorithm for ultra-high dimensional feature selection in sublinear memory, 2020.

[18] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 1113–1120. ACM, 2009.

[19] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, Alex Strehl, and S. V. N. Vishwanathan. Hash kernels. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 496–503, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[20] Ryan Spring, Anastasios Kyrillidis, Vijai Mohan, and Anshumali Shrivastava. Compressing gradient optimizers via count-sketches. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5946–5955. PMLR, 09–15 Jun 2019.

[21] Robert Calderbank, Sina Jafarpour, and Robert Schapire. Compressed learning: Universal sparse dimensionality reduction and learning in the measurement domain. Technical report, 2009.

[22] Elad Hazan. Introduction to online convex optimization. *CoRR*, abs/1909.05207, 2019.

[23] Anna C. Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proc. IEEE*, 98(6):937–947, 2010.

[24] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011.

[25] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 61–70. IEEE Computer Society, 2016.

[26] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

Optionally include extra information (complete proofs, additional experiments and plots) in the appendix. This section will often be part of the supplemental material.