Streaming Coresets for Symmetric Tensor Factorization

Rachit Chhaya [1] **Jayesh Choudhari** [1] **Anirban Dasgupta** [1] **Supratim Shit** [1]

Abstract

Factorizing tensors has recently become an important optimization module in a number of machine learning pipelines, especially in latent variable models. We show how to do this efficiently in the streaming setting. Given a set of n vectors, each in $\mathbb{R}^d$, we present algorithms to select a sublinear number of these vectors as coreset, while guaranteeing that the CP decomposition of the p-moment tensor of the coreset approximates the corresponding decomposition of the p-moment tensor computed from the full data. We introduce two novel algorithmic techniques: online filtering and kernelization. Using these two, we present four algorithms that achieve different tradeoffs of coreset size, update time and working space, beating or matching various state of the art algorithms. In the case of matrices (2-ordered tensor), our online row sampling algorithm guarantees $(1 \pm \epsilon)$ relative error spectral approximation. We show applications of our algorithms in learning single topic modeling.

1. Introduction

Much of the data that is consumed in data mining and machine learning applications arrives in a streaming manner. The data is conventionally treated as a matrix, with a row representing a single data point and the columns its corresponding features. Since the matrix is typically large, it is advantageous to be able to store only a small number of rows and still preserve some of its "useful" properties. One such abstract property that has proven useful in a number of different settings, such as solving regression, finding various factorizations, is *subspace preservation*. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, an $m \times d$ matrix $\mathbf{C}$ is its subspace preserving matrix for the ℓ_2 norm if, $\forall \mathbf{x} \in \mathbb{R}^d$,

$$\Big| \sum_{\tilde{\mathbf{a}}_j \in \mathbf{C}} (\tilde{\mathbf{a}}_j^T \mathbf{x})^2 - \sum_{\mathbf{a}_i \in \mathbf{A}} (\mathbf{a}_i^T \mathbf{x})^2 \Big| \leq \epsilon \cdot \sum_{\mathbf{a}_i \in \mathbf{A}} (\mathbf{a}_i^T \mathbf{x})^2$$

We typically desire $m \ll n$ and $\tilde{\mathbf{a}}_j$'s represent the subsampled and rescaled rows from $\mathbf{A}$. Such a sample $\mathbf{C}$ is often referred to as a coreset. This property has been used to obtain approximate solutions to many problems such as regression, low-rank approximation, etc (Woodruff et al., 2014) while having m to be at most $O(d^2)$. Such property has been defined for other ℓ_p norms too (Dasgupta et al., 2009; Cohen & Peng, 2015; Clarkson et al., 2016).

Matrices are ubiquitous, and depending on the application, one can assume that the data is coming from a generative model, i.e., there is some distribution from which every incoming data point (or row) is sampled and given to user. Many a time, the goal is to know the hidden variables of this generative model. An obvious way to learn these variables is by representing data (matrix) by its low-rank representation. However, we know that a low-rank representation of a matrix is not unique as there are various ways (such as SVD, QR, LU) to decompose a matrix. So it difficult to realize the hidden variables just by the low-rank decomposition of the data matrix. This is one of the reasons to look at higher order moments of the data i.e. tensors. Tensors are formed by outer product of data vectors, i.e. for a dataset $\mathbf{A} \in \mathbb{R}^{n \times d}$ one can use a p order tensor $\mathcal{T} \in \mathbb{R}^{d \times \ldots \times d}$ as $\mathcal{T} = \sum_{i=1}^{n} \mathbf{a}_i \otimes^p$, where p is set by user depending on the number of latent variables one is expecting in the generative model (Ma et al., 2016). The decomposition of such a tensor is unique under a mild assumption (Kruskal, 1977). Factorization of tensors into its constituent elements has found uses in many machine learning applications such as topic modeling (Anandkumar et al., 2014), various latent variable models (Anandkumar et al., 2012; Hsu et al., 2012; Jenatton et al., 2012), training neural networks (Janzamin et al., 2015) etc.

For a p-order moment tensor $\mathcal{T} = \sum_i \mathbf{a}_i \otimes^p$ created using the set of vectors $\{\mathbf{a}_i\}$ and for $\mathbf{x} \in \mathbb{R}^d$ one of the important property one needs to preserve is $\mathcal{T}(\mathbf{x}, \ldots, \mathbf{x}) = \sum_i (\mathbf{a}_i^T \mathbf{x})^p$. This operation is also called tensor contraction (Song et al., 2016). Now if we wish to "approximate" it using only a subset of the rows in $\mathbf{A}$, the above property for ℓ_2 norm subspace preservation does not suffice.

[1]Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India. Correspondence to: Supratim Shit <supratim.shit@iitgn.ac.in>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

For tensor factorization, which is performed using power iteration, a coreset $\mathbf{C} \subseteq \{\mathbf{a}_i\}$, in order to give a guaranteed approximation to the tensor factorization, needs to satisfy the following natural extension of the ℓ_2 subspace preservation condition:

$$\sum_{\mathbf{a}_i \in \mathbf{A}} ({\mathbf{a}_i}^T \mathbf{x})^p \approx \sum_{\tilde{\mathbf{a}}_j \in \mathbf{C}} (\tilde{\mathbf{a}}_j^T \mathbf{x})^p$$

Ensuring this tensor contraction property enables one to approximate the CP decomposition of $\mathcal{T}$ using only the vectors $\tilde{\mathbf{a}}_i$'s via power iteration method (Anandkumar et al., 2014). A related notion is that of ℓ_p subspace embedding where we need that $\mathbf{C}$ satisfies the following, $\forall \mathbf{x} \in \mathbb{R}^d$

$$\sum_{\mathbf{a}_i \in \mathbf{A}} |{\mathbf{a}_i}^T \mathbf{x}|^p \approx \sum_{\tilde{\mathbf{a}}_j \in \mathbf{C}} |\tilde{\mathbf{a}}_j^T \mathbf{x}|^p$$

In this work, we show that it is possible to create coresets for the above property in streaming and restricted streaming (*online*) setting. In restricted streaming setting an incoming point, when it arrives, is either chosen in the set or discarded forever. We consider the following formalization of the above two properties. Given a query space of vectors $\mathbf{Q} \subseteq \mathbb{R}^d$ and $\epsilon > 0$, we aim to choose a set $\mathbf{C}$ which contains sampled and rescaled rows from $\mathbf{A}$ to ensure that $\forall \mathbf{x} \in \mathbf{Q}$ with probability at least 0.99, the following properties hold,

$$\Big| \sum_{\tilde{\mathbf{a}}_j \in \mathbf{C}} (\tilde{\mathbf{a}}_j^T \mathbf{x})^p - \sum_{\mathbf{a}_i \in \mathbf{A}} (\mathbf{a}_i^T \mathbf{x})^p \Big| \leq \epsilon \cdot \sum_{\mathbf{a}_i \in \mathbf{A}} |\mathbf{a}_i^T \mathbf{x}|^p \quad (1)$$

$$\Big| \sum_{\tilde{\mathbf{a}}_j \in \mathbf{C}} |\tilde{\mathbf{a}}_j^T \mathbf{x}|^p - \sum_{\mathbf{a}_i \in \mathbf{A}} |\mathbf{a}_i^T \mathbf{x}|^p \Big| \leq \epsilon \cdot \sum_{\mathbf{a}_i \in \mathbf{A}} |\mathbf{a}_i^T \mathbf{x}|^p \quad (2)$$

Note that neither property follows from the other. For even values of p, the above properties are identical and imply a relative error approximation as well. For odd values of p, the ℓ_p subspace embedding as equation (2) gives a relative error approximation but the tensor contraction as equation (1) implies an additive error approximation, which becomes relative error under non-negativity constraints on $\mathbf{a}_i$ and $\mathbf{x}$. This happens, for instance, for the important use case of topic modeling, where $p = 3$ typically.

Our Contributions: We propose various methods to sample rows in streaming manner for a p order tensor, which is further decomposed to know the latent factors. For a given matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, a k-dimensional query space $\mathbf{Q} \in \mathbb{R}^{k \times d}, \epsilon > 0$ and $p \geq 2$,

- We give an algorithm (`LineFilter`) that is able to select rows, it takes $O(d^2)$ update time, and returns a sample of size $O(\frac{n^{1-2/p}dk}{\epsilon^2}(1 + \log \|\mathbf{A}\| - d^{-1} \min_i \log \|\mathbf{a}_i\|))$ such that the set of selected rows forms a coreset having the guarantees stated in equations (1) and (2) (Theorem 4.1). It is a streaming algorithm but also works well in the restricted streaming (online) setting.
- We improve the sampling complexity of our coreset to $O(d^{p/2}k(\log n)^{10}\epsilon^{-5})$ by a streaming algorithm (`LineFilter`+`StreamingLW`) with amortized update time $O(d^2)$ (Theorem 4.2). It requires slightly higher working space $O(d^{p/2}k(\log n)^{11}\epsilon^{-5})$.
- For integer value $p \geq 2$ we present a kernelization technique which, for any vector $\mathbf{v}$, creates two vectors $\grave{\mathbf{v}}$ and $\acute{\mathbf{v}}$ such that for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

  $$|\mathbf{x}^T \mathbf{y}|^p = |\grave{\mathbf{x}}^T \grave{\mathbf{y}}| \cdot |\acute{\mathbf{x}}^T \acute{\mathbf{y}}|$$

  Using this technique, we give an algorithm (`KernelFilter`) which takes $O(nd^p)$ time and samples $O(\frac{d^{p/2}k}{\epsilon^2}(1 + p(\log \|\mathbf{A}\| - d^{-p/2} \min_i \log \|\mathbf{a}_i\|))$ vectors to create a coreset having the same guarantee as (1) and (2) (Theorem 4.3) for even value p. For odd value p it takes $O(nd^{p+1})$ time and samples $O(\frac{n^{1/(p+1)}d^{p/2}k}{\epsilon^2}(1 + (p+1)(\log \|\mathbf{A}\| - d^{-\lceil p/2 \rceil} \min_i \log \|\mathbf{a}_i\|))^{p/(p+1)})$ vectors to create a coreset having the same guarantee as (1) and (2) (Theorem 4.4). Both update time and working space of the algorithm for even p is $O(d^p)$ and for odd p it is $O(d^{p+1})$. It is a streaming algorithm but also works well in the restricted streaming (online) setting.
- For integer value $p \geq 2$ we combine both online algorithms and propose another online algorithm (`LineFilter`+`KernelFilter`) which has $O(d^2)$ amortized update time and returns $O(\frac{d^{p/2}k}{\epsilon^2}(1 + p(\log \|\mathbf{A}\| - d^{-p/2} \min_i \log \|\mathbf{a}_i\|))$ for even p and $O(\frac{n^{(p-2)/(p^2+p)}d^{p/2+1/4}k^{5/4}}{\epsilon^2}(1 + (p+1)(\log \|\mathbf{A}\| - d^{-\lceil p/2 \rceil} \min_i \log \|\mathbf{a}_i\|))^{p/(p+1)})$ vectors for odd p as coreset with same guarantees as equation (1) and (2) (Theorem 4.5). Here the working space is same as `KernelFilter`. The factor $n^{(p-2)/(p^2+p)} \leq n^{1/10}$.
- For the $p = 2$ case, both `LineFilter` and `KernelFilter` translate to an online algorithm for sampling rows of the matrix $\mathbf{A}$, while guaranteeing a *relative error* spectral approximation (Theorem A.5). This is an improvement (albeit marginal) over the online row sampling result by (Cohen et al., 2016). The additional benefit of this new online algorithm over (Cohen et al., 2016) is that it does not need knowledge of $\sigma_{\min}(\mathbf{A})$ to give a relative error approximation.

The rest of this paper is organized as follows: In section 2, we look at some preliminaries for tensors and coresets. We also describe the notation used throughout the paper. Section

3 discusses related work. In section 4, we state all the four streaming algorithms along with their guarantees. We also show how our problem of preserving tensor contraction relates to preserving ℓ_p subspace embedding. In section 5, we describe the guarantees given by our algorithm and some proofs. In section 6, we describe how our algorithm can be used in case of streaming single topic modeling. We give empirical results that compare our sampling scheme with other schemes. Proofs of theorems and the supporting lemmas have been delegated to the appendix A available in the supplementary material.

2. Preliminaries

A scalar is denoted by a lower case letter, e.g., p while a vector is denoted by a boldface lower case letter, e.g., $\mathbf{a}$. By default, all vectors are considered as column vectors unless specified otherwise. Matrices and sets are denoted by boldface upper case letters, e.g., $\mathbf{A}$. Specifically, $\mathbf{A}$ denotes an $n \times d$ matrix with set of rows $\{\mathbf{a}_i\}$ and, in the streaming setting, $\mathbf{A}_i$ represents the matrix formed by the first i rows of $\mathbf{A}$ that have arrived. We will interchangeably refer to the set $\{\mathbf{a}_i\}$ as the input set of vectors as well as the rows of the matrix $\mathbf{A}$. A tensor is denoted by a bold calligraphy letter e.g. $\mathcal{T}$. Given a set of $d-$dimensional vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, from which a p-order symmetric tensor $\mathcal{T}$ is obtained as $\mathcal{T} = \sum_i^n \mathbf{a}_i{}^{\otimes p}$ i.e., the sum of the p-order outer product of each of the vectors. It is easy to see that a symmetric tensor $\mathcal{T}$ satisfies the following: $\forall i_1, i_2, \ldots, i_p; \mathcal{T}_{i_1,i_2,\ldots,i_p} = \mathcal{T}_{i_2,i_1,\ldots,i_p} = \ldots = \mathcal{T}_{i_p,i_{p-1},\ldots,i_1}$, i.e. equal for all possible permutations of $(i_1, i_2, \ldots, i_p)$. We define the scalar quantity, also known as *tensor contraction*, as $\mathcal{T}(\mathbf{x}, \ldots, \mathbf{x}) = \sum_{i=1}^n (\mathbf{a}_i^T \mathbf{x})^p$, where $\mathbf{x} \in \mathbb{R}^d$. There are three widely used tensor decomposition techniques known as CANDECOMP/PARAFAC(CP), Tucker and Tensor Train decomposition (Kolda & Bader, 2009; Oseledets, 2011). Our work focuses on CP decomposition.

We denote 2-norm for a vector $\mathbf{x}$ as $\|\mathbf{x}\|$, and any p-norm, for $p \neq 2$ as $\|\mathbf{x}\|_p$. We denote the 2-norm or spectral norm of a matrix $\mathbf{A}$ by $\|\mathbf{A}\|$ and for a tensor $\mathcal{T}$ by $\|\mathcal{T}\|$ which is $\sup_{\mathbf{x}|\mathbf{x}\neq 0} \mathcal{T}(\mathbf{x}, \ldots, \mathbf{x})/\|\mathbf{x}\|^p$.

Coreset: It is a small summary of data which can give provable guarantees for a particular optimization problem. Formally, given a set $\mathbf{X} \subseteq \mathbb{R}^d$, query set $\mathbf{Q}$, a non-negative cost function $f_{\mathbf{q}}(\mathbf{x})$ with parameter $\mathbf{q} \in \mathbf{Q}$ and data point $\mathbf{x} \in \mathbf{X}$, a set of subsampled and appropriately reweighed points $\mathbf{C}$ is called a *coreset* if $\forall \mathbf{q} \in \mathbf{Q}$, $|\sum_{\mathbf{x}\in\mathbf{X}} f_{\mathbf{q}}(\mathbf{x}) - \sum_{\tilde{\mathbf{x}}\in\mathbf{C}} f_{\mathbf{q}}(\tilde{\mathbf{x}})| \leq \epsilon \sum_{\mathbf{x}\in\mathbf{X}} f_{\mathbf{q}}(\mathbf{x})$, for some $\epsilon > 0$.

To guarantee the above approximation, one can define a set of scores, termed as sensitivities (Langberg & Schulman, 2010) corresponding to each data point. This can be used to create coresets via importance sampling. The sensitivity of a point $\mathbf{x}$ is $s_{\mathbf{x}} = \sup_{\mathbf{q}\in\mathbf{Q}} \frac{f_{\mathbf{q}}(\mathbf{x})}{\sum_{\mathbf{x}'\in\mathbf{X}} f_{\mathbf{q}}(\mathbf{x}')}$. In (Langberg & Schulman, 2010), authors show that using any upper bounds to the sensitivity scores, we can create a probability distribution, which can be used to sample a coreset. The size of the coreset depends on the sum of these upper bounds and the dimension of the query space.

3. Related Work

Coresets are small summaries of data which can be used as a proxy to the original data with provable guarantees. The term was first introduced in (Agarwal et al., 2004), where they used coresets for the shape fitting problem. Coresets for clustering problems were described in (Har-Peled & Mazumdar, 2004). In (Feldman & Langberg, 2011), authors gave a generalized framework to construct coresets based on importance sampling using sensitivity scores introduced in (Langberg & Schulman, 2010). Interested reader can check (Woodruff et al., 2014; Braverman et al., 2016; Bachem et al., 2017). Various online sampling schemes for spectral approximation are discussed in (Cohen et al., 2016; 2017).

Tensor decomposition is unique under minimal assumptions (Kruskal, 1977). Therefore it has become very popular in various latent variable modeling applications (Anandkumar et al., 2012; Hsu et al., 2012; Anandkumar et al., 2014), neural networks (Janzamin et al., 2015) etc. However, in general (i.e., without any assumption), most of the tensor problems, including tensor decomposition, are NP-hard (Hillar & Lim, 2013). There has been much work on fast tensor decomposition techniques. Various tensor sketching methods for tensor operations are discussed in (Bhojanapalli & Sanghavi, 2015; Wang et al., 2015; Song et al., 2016). For 3-order, orthogonally decomposable tensors, (Song et al., 2016) gives a sub-linear time algorithm for tensor decomposition, which requires the knowledge of norms of slices of the tensor. The area of online tensor power iterations has also been explored in (Huang et al., 2015; Wang & Anandkumar, 2016). Various heuristics for tensor sketching as well as RandNLA techniques (Woodruff et al., 2014) over matricized tensors for estimating low-rank tensor approximation have been studied in (Song et al., 2019). There are few algorithms that use randomized techniques to make CP-ALS, i.e., CP tensor decomposition based on alternating least square method are more practical (Battaglino et al., 2018; Erichson et al., 2020). Here the author shows various randomized techniques based on sampling and projection to improve the running time and robustness of the CP decomposition. In (Erichson et al., 2020), authors also show that their randomized projection based algorithm can also be used in power iteration based tensor decomposition. For many of these decomposition techniques, our algorithm can be used as a prepossessing.

In the online setting, for a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ where rows are coming in streaming manner, the guarantee achieved by

Table 1. Table comparing existing work (first four rows) and current contributions. `StreamingXX` refers to the obvious extension of the `XX` algorithm to the streaming model using merge-reduce.

ALGORITHM	SAMPLE SIZE $\tilde{O}(\cdot)$	UPDATE TIME	WORKING SPACE $\tilde{O}(\cdot)$
STREAMINGWCB (DASGUPTA ET AL., 2009)	$d^p k\epsilon^{-2}$	$d^5 p \log d$	$d^p k\epsilon^{-2}$
STREAMINGLW (COHEN & PENG, 2015)	$d^{p/2} k\epsilon^{-5}$	$d^C p \log d$	$d^{p/2} k\epsilon^{-5}$
STREAMINGFC (CLARKSON ET AL., 2016)	$d^{7p/2}\epsilon^{-2}$	d	$d^{7p/2}\epsilon^{-2}$
STREAMING (DICKENS ET AL., 2018)	$n^\gamma d\epsilon^{-2}$	$n^\gamma d^5$	$n^\gamma d$
LINEFILTER (THEOREM 4.1)	$n^{1-2/p} dk\epsilon^{-2}$	d^2	d^2
LINEFILTER+STREAMINGLW (THEOREM 4.2)	$d^{p/2} k\epsilon^{-5}$	d^2	$d^{p/2} k\epsilon^{-5}$
KERNELFILTER (THEOREM 4.3)(EVEN p)	$d^{p/2} k\epsilon^{-2}$	d^p	d^p
KERNELFILTER (THEOREM 4.4)(ODD p)	$n^{1/(p+1)} d^{p/2} k\epsilon^{-2}$	d^{p+1}	d^{p+1}
LINEFILTER+KERNELFILTER (THEOREM 4.5)(EVEN p)	$d^{p/2} k\epsilon^{-2}$	d^2	d^p
LINEFILTER+KERNELFILTER (THEOREM 4.5)(ODD p)	$n^{(p-2)/(p^2+p)} d^{p/2+1/4} k^{5/4}\epsilon^{-2}$	d^2	d^{p+1}

(Cohen et al., 2016) while preserving additive error spectral approximation $|\|\mathbf{Ax}\|^2 - \|\mathbf{Cx}\|^2| \leq \epsilon\|\mathbf{Ax}\|^2 + \delta, \forall \mathbf{x} \in \mathbb{R}^d$, with sample size $O(d(\log d)(\log \epsilon\|\mathbf{A}\|^2/\delta))$.

The problem of ℓ_p subspace embedding has been explored in both offline (Dasgupta et al., 2009; Woodruff & Zhang, 2013; Cohen & Peng, 2015; Clarkson et al., 2016) and streaming setting (Dickens et al., 2018). As any offline algorithm to construct coresets can be used as a streaming algorithm (Har-Peled & Mazumdar, 2004) using merge-reduce, we use the known offline algorithms and summarize the results of their streaming version in table[1] 1. The algorithm in (Woodruff & Zhang, 2013) samples $O(n^{1-2/p}\text{poly}(d))$ rows and gives $\text{poly}(d)$ error relative subspace embedding but in $O(\text{nnz}(\mathbf{A}))$ time. For streaming ℓ_p subspace embedding (Dickens et al., 2018), give a one pass deterministic algorithm for ℓ_p subspace embedding for $1 \leq p \leq \infty$. For some constant $\gamma \in (0, 1)$ the algorithm takes $O(n^\gamma d)$ space and $O(n^\gamma d^2 + n^\gamma d^5 \log n)$ update time to return a $1/d^{O(1/\gamma)}$ relative error subspace embedding for any ℓ_p norm.

4. Algorithms and Guarantees

In this section we propose all the four streaming algorithms which are based on two major contributions. We first introduce the two algorithmic modules–`LineFilter` and `KernelFilter`. For real value $p \geq 2$, `LineFilter`, on arrival of each row, simply decides whether to sample it or not. The sampling probability is computed based on the stream seen till now, where as `KernelFilter` works for integer value $p \geq 2$, for every incoming row $\mathbf{a}_i$, the decision of sampling it, depends on two rows $\grave{\mathbf{a}}_i$ and $\acute{\mathbf{a}}_i$ we define from $\mathbf{a}_i$ such that: for any vector $\mathbf{x}$, there is a similar transformation ($\grave{\mathbf{x}}$ and $\acute{\mathbf{x}}$) and we get, $|\mathbf{a}_i^T\mathbf{x}|^p = |\grave{\mathbf{a}}_i^T\grave{\mathbf{x}}| \cdot |\acute{\mathbf{a}}_i^T\acute{\mathbf{x}}|$. For even value p we define $|\mathbf{a}_i^T\mathbf{x}|^p = |\acute{\mathbf{a}}_i^T\acute{\mathbf{x}}|^2$ and for odd value p we define $|\mathbf{a}_i^T\mathbf{x}|^p = |\acute{\mathbf{a}}_i^T\acute{\mathbf{x}}|^{2p/(p+1)}$. We call it kernelization. A similar kernelization is also discussed in (Schechtman, 2011) for even value ℓ_p subspace embedding.

Note that both `LineFilter` and `KernelFilter` are restricted streaming algorithms in the sense that each row is selected / processed only when it arrives. This online nature of the two modules allows us to use these as modules in order to create the following algorithms

1. `LineFilter+StreamingLW`: Here, the output streams of `LineFilter` is fed to a `StreamingLW`, which is a merge-and-reduce based streaming algorithm based on Lewis Weights (Cohen & Peng, 2015). Here the `StreamingLW` outputs the final coreset.

2. `LineFilter+KernelFilter`: Here, the output streams from `LineFilter` is first kernelized. It is then passed to `KernelFilter`, which outputs the final coreset.

Note that `LineFilter+StreamingLW` is a streaming algorithm which works for any $p \geq 2$ where as the algorithm `LineFilter+KernelFilter` even works in a restricted streaming setting for integer valued $p \geq 2$.

The algorithms `LineFilter` and `KernelFilter` call a function `Score`$(\cdot)$, which computes a score for every incoming row, based on the score, the sampling probability of the row is decided. The score depends on the incoming row (say $\mathbf{x}_i$) and some prior knowledge (say $\mathbf{M}$) of the data, which it has already seen. Here, we define $\mathbf{M} = \mathbf{X}_{i-1}^T\mathbf{X}_{i-1}$ and $\mathbf{Q}$ is its orthonormal column basis. Here $\mathbf{X}_{i-1}$ represents the matrix with rows $\{\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}\}$ which have arrived so far. We present `Score`$(\cdot)$ as algorithm 1.

In the function `Score`$(\cdot)$ if the incoming row $\mathbf{x}_i$ lies in the subspace spanned by $\mathbf{Q}$ (i.e. if $\|\mathbf{Qx}_i\| = \|\mathbf{x}_i\|$), then the algorithm `Score`$(\cdot)$ does not require to compute inverse or pseudo-inverse of $\mathbf{M}$ hence it takes $O(m^2)$ time. If it does not lie in the subspace then it takes $O(m^3)$ where $\mathbf{x}_i \in \mathbb{R}^m$. Here we have used a modified version of Sherman Morrison formula (Sherman & Morrison, 1950) to compute $(\mathbf{X}_i^T\mathbf{X}_i)^\dagger = (\mathbf{X}_{i-1}^T\mathbf{X}_{i-1} + \mathbf{x}_i\mathbf{x}_i^T)^\dagger = (\mathbf{M} + \mathbf{x}_i\mathbf{x}_i^T)^\dagger$. Note

[1] `LineFilter`, `KernelFilter` and their combination can also be used as online algorithm. Update times are amortized for streaming algorithms.

Algorithm 1 `Score`$(\mathbf{x}_i, \mathbf{M}, \mathbf{M}_{inv}, \mathbf{Q})$

if $\mathbf{x}_i \in$ column-space$(\mathbf{Q})$ **then**
  $\mathbf{M}_{inv} = \mathbf{M}_{inv} - \frac{(\mathbf{M}_{inv})\mathbf{x}_i\mathbf{x}_i^T(\mathbf{M}_{inv})}{1+\mathbf{x}_i^T(\mathbf{M}_{inv})\mathbf{x}_i}$
  $\mathbf{M} = \mathbf{M} + (\mathbf{x}_i\mathbf{x}_i^T)$
else
  $\mathbf{M} = \mathbf{M} + \mathbf{x}_i\mathbf{x}_i^T$; $\mathbf{M}_{inv} = \mathbf{M}^\dagger$
  $\mathbf{Q} =$ orthonormal-column-basis$(\mathbf{M})$
end if
$\tilde{e}_i = \mathbf{x}_i^T(\mathbf{M}_{inv})\mathbf{x}_i$
Return $\tilde{e}_i, \mathbf{M}, \mathbf{M}_{inv}, \mathbf{Q}$

that in our setup $\mathbf{M}$ need not be full rank, and we use the formula $(\mathbf{X}_i\mathbf{X}_i)^\dagger = \mathbf{M}^\dagger - \frac{\mathbf{M}^\dagger\mathbf{x}_i\mathbf{x}_i^T\mathbf{M}^\dagger}{1+\mathbf{x}_i^T\mathbf{M}^\dagger\mathbf{x}_i}$. We prove the correctness of the formula as a lemma in the appendix A.2.

4.1. `LineFilter`

Here we present our first streaming algorithm which ensures equation (1) for integer valued $p \geq 2$ and equation (2) for any real $p \geq 2$. The algorithm can also be used in restricted steaming (online) settings where for every incoming row, we get only one chance to decide whether to sample it or not. Due to its nature of filtering out rows, we call it `LineFilter` algorithm. The algorithm tries to reduce the variance of the difference between the cost from the original and the sampled term. In order to achieve that, we use sensitivity based framework to decide the sampling probability of each row. The sampling probability of a row is proportional to its sensitivity scores. In some sense, the sensitivity score of a row captures the fact that how much the variance of the difference is going to get affected if that row is not present in the set of sampled rows. We discuss it in detail in section 5. Now we present the `LineFilter` algorithm.

Algorithm 2 `LineFilter`

Require: Streaming rows $\mathbf{a}_i, i = 1, \ldots n, p \geq 2, r > 1$
Ensure: Coreset $\mathbf{C}$ satisfying eqn (1) and (2) w.h.p.
$\mathbf{M} = \mathbf{M}_{inv} = \mathbf{0}^{d\times d}$, $L = 0$, $\mathbf{C} = \emptyset$
$\mathbf{Q} =$ orthonormal-column-basis$(\mathbf{M})$
while $i \leq n$ **do**
  $[\tilde{e}_i, \mathbf{M}, \mathbf{M}_{inv}, \mathbf{Q}] =$ `Score`$(\mathbf{a}_i, \mathbf{M}, \mathbf{M}_{inv}, \mathbf{Q})$
  $\tilde{l}_i = \min\{i^{p/2-1}(\tilde{e}_i)^{p/2}, 1\}$
  $L = L + \tilde{l}_i$; $p_i = \min\{r\tilde{l}_i/L, 1\}$
  Sample $\mathbf{a}_i/\sqrt[p]{p_i}$ in $\mathbf{C}$ with probability p_i
end while
Return $\mathbf{C}$

Every time a row $\mathbf{a}_i \in \mathbb{R}^d$ comes, the `LineFilter` calls the function 1 (i.e. `Score`$(\cdot)$) which returns a score $\tilde{e}_i$. Then `LineFilter` computes $\tilde{l}_i$, which is an upper bound to its sensitivity score. Based on $\tilde{l}_i$ the row's sampling probability is decided. We formally define and discuss sensitivity scores of our problem in section 5.

Now for the `Score`$(\cdot)$ function there can be at most d occasions where an incoming row is not in the row space of the previously seen rows, i.e. $\mathbf{Q}$. In these cases `Score`$(\cdot)$ takes $O(d^3)$ time and for the other, at least $n-d$, cases by Sherman Morrison formula it takes $O(d^2)$ time to return $\tilde{e}_i$. Hence the entire algorithm just takes $O(nd^2)$ time. Now we summarize the guarantees of `LineFilter`.

Theorem 4.1. *Given $\mathbf{A} \in \mathbb{R}^{n\times d}$ whose rows are coming in streaming manner, `LineFilter` selects a set $\mathbf{C}$ of size $O(\frac{n^{1-2/p}dk}{\epsilon^2}(1+\log\|\mathbf{A}\| - d^{-1}\min_i \log\|\mathbf{a}_i\|))$ using both working space and update time $O(d^2)$. Suppose $\mathbf{Q}$ is a fixed k-dimensional subspace, then with probability at least 0.99, for integer value $p \geq 2, \epsilon > 0$, $\forall \mathbf{x} \in \mathbf{Q}$, the set $\mathbf{C}$ satisfies both p-order tensor contraction and ℓ_p subspace embedding as in equations (1) and (2) respectively.*

It can be used to get an ℓ_p subspace embedding for any real $p \geq 2$. It is worth noting that `LineFilter` benefits by taking very less working space and computation time, which are independent of p. However `LineFilter` gives a coreset which is sub-linear to input size but as p increases the coreset size tends towards $O(n)$.

4.2. `LineFilter+StreamingLW`

Now we present a streaming algorithm which returns a coreset for the same problem with its coreset size smaller than that of `LineFilter`. Here first we want to point out that our coresets for tensor contraction i.e., equation (1), also preserve ℓ_p subspace embedding i.e., equation (2). This is mainly due to two reasons. First is that our coreset is a subsample of original data, and second is because of the way we define our sampling probability.

For simplicity, we show this relation in the complete offline setting, where we have access to the entire data $\mathbf{A}$. For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, we intend to preserve the tensor contraction property in equation (1). We create $\mathbf{C}$ by sampling original rows $\mathbf{a}_i$ with appropriate scaling. We analyze the variance of the difference between the tensor contraction from original and sampled term, through Bernstein inequality (Dubhashi & Panconesi, 2009) and try to reduce it. Here we use sensitivity based framework to decide our sampling probability where we know sensitivity scores are well defined for positive cost function (Langberg & Schulman, 2010). Now with the tensor contraction problem for odd p and for some $\mathbf{x}$, the cost $(\mathbf{a}_i^T\mathbf{x})^p$ could be negative. So for every row i we define the sensitivity score as follows,

$$s_i = \sup_{\mathbf{x}} \frac{|\mathbf{a}_i^T\mathbf{x}|^p}{\sum_{j=1}^n |\mathbf{a_j}^T\mathbf{x}|^p} \tag{3}$$

Here by sampling enough number of rows based on above defined sensitivity scores would ensure tensor contraction

as in (1) and it will also preserve $\sum_{i=1}^{n} |\mathbf{a}_i^T \mathbf{x}|^p = \|\mathbf{A}\mathbf{x}\|_p^p$ (Langberg & Schulman, 2010). The sampled rows creates a coreset $\mathbf{C}$ which is ℓ_p subspace embedding, i.e. $\forall \mathbf{x}$, $|\|\mathbf{A}\mathbf{x}\|_p^p - \|\mathbf{C}\mathbf{x}\|_p^p| \leq \epsilon \|\mathbf{A}\mathbf{x}\|_p^p$. We define the online version of these scores in section 5 which also preserve tensor contraction. Sampling based methods used in (Dasgupta et al., 2009; Cohen & Peng, 2015; Clarkson et al., 2016) to get a coreset for ℓ_p subspace embedding also preserve tensor contraction. This is because these sampling based methods reduces the variance of the difference between the cost function from the original and the sampled terms.

We know that any offline coreset algorithm can be made into a streaming algorithm using merge and reduce method (Har-Peled & Mazumdar, 2004). For $p \geq 2$, the sampling complexity of (Cohen & Peng, 2015) is best among all the other methods that we mentioned. Hence here we use Lewis Weights sampling (Cohen & Peng, 2015) as the offline method along with merge and reduce for its streaming version, which we call `StreamingLW`. The following lemma summarizes the guarantee of `StreamingLW`.

Lemma 4.1. *Given a set of n streaming rows $\{\mathbf{a}_i\}$, the `StreamingLW` returns a coreset $\mathbf{C}$. For integer $p \geq 2$, a fixed k-dimensional subspace $\mathbf{Q}$, with probability 0.99 and $\epsilon > 0$, $\forall \mathbf{x} \in \mathbb{R}^d$, $\mathbf{C}$ satisfies p-order tensor contraction and ℓ_p subspace embedding as in equations (1) and (2).*

It requires $O(d^C p \log d)$ amortized update time and uses $O(d^{p/2}\epsilon^{-5}\log^{11} n)$ working space to return a coreset $\mathbf{C}$ of size $O(d^{p/2}\epsilon^{-5}\log^{10} n)$ here C is a constant.

The proof is fairly straight forward which we discuss in the appendix A.2.1. It also guarantees ℓ_p subspace embedding for real $p \geq 2$. Note that in this case, both update time and working space depends on d and p.

Now we propose our second algorithm, where we feed the output of `LineFilter` to `StreamingLW` method. Here every incoming row is fed to `LineFilter`, which quickly computes a sampling probability and based on which the row gets sampled. Now, if it gets sampled, then we pass it to the `StreamingLW` method, which returns the final coreset. The entire algorithm gets an improved *amortized* update time compared to `StreamingLW` and improved sampling complexity compared to `LineFilter`. We call this algorithm `LineFilter+StreamingLW` and summarize its guarantees in the following theorem.

Theorem 4.2. *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ with rows are coming in streaming manner. `LineFilter+StreamingLW` takes $O(d^2)$ amortized update time and uses working space of $O((1-2/p)^{11} d^{p/2}\epsilon^{-5}\log^{11} n)$ to return a coreset $\mathbf{C}$ of size $O((1-2/p)^{10} d^{p/2}\epsilon^{-5}\log^{10} n)$ such that with at least 0.99 probability, $\mathbf{C}$ satisfies both p-order tensor contraction and ℓ_p subspace embedding as in equations (1) and (2).*

This is an improved streaming algorithm which gives the same guarantee as lemma 4.1 but using very less amortized update time. Hence asymptotically, we get an improvement in the overall run time of the algorithm and yet get a coreset which is smaller than that of `LineFilter`. Similar to `StreamingLW`, `LineFilter+StreamingLW` also ensures ℓ_p subspace embedding for real $p \geq 2$. It is important to note that we could improve the run time of the streaming result because our `LineFilter` can be used in an online manner, which returns a sub-linear size coreset (i.e., $o(n)$) and its update time is less than the amortized update time of `StreamingLW`. The proof of the above theorem is discussed in the appendix A.2.2. Note that `LineFilter+StreamingLW` is a streaming algorithm, whereas `LineFilter` or the next algorithm that we propose works even in a restricted streaming setting.

4.3. `KernelFilter`

Now we discuss our second module which is also a streaming algorithm for the tensor contraction guarantee as equation (1). First we give a reduction from p-order function to q-order function, where $q \leq 2$.

Lemma 4.2. *For an integer value $p \geq 2$, a vector $\mathbf{x} \in \mathbb{R}^d$ can be transformed to $(\grave{\mathbf{x}}$ and $\acute{\mathbf{x}})$ such that for any two d-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ with their similar transformations we get,*

$$|\mathbf{x}^T\mathbf{y}|^p = |\grave{\mathbf{x}}^T\grave{\mathbf{y}}| \cdot |\acute{\mathbf{x}}^T\acute{\mathbf{y}}| = \begin{cases} |\acute{\mathbf{x}}^T\acute{\mathbf{y}}|^2 & \text{if } p \text{ even} \\ |\acute{\mathbf{x}}^T\acute{\mathbf{y}}|^{2p/(p+1)} & \text{if } p \text{ odd} \end{cases}$$

For even valued p, $(\grave{\mathbf{x}}, \grave{\mathbf{y}})$ are same as $(\acute{\mathbf{x}}, \acute{\mathbf{y}})$. So we define $|\mathbf{x}^T\mathbf{y}|^p = |\acute{\mathbf{x}}^T\acute{\mathbf{y}}|^2$. For odd value p, $(\grave{\mathbf{x}}, \grave{\mathbf{y}})$ are not same as $(\acute{\mathbf{x}}, \acute{\mathbf{y}})$ and we define $|\mathbf{x}^T\mathbf{y}|^p = |\acute{\mathbf{x}}^T\acute{\mathbf{y}}|^{2p/(p+1)}$. The proof is discussed in more detail in the appendix A.4.

Now we give a streaming algorithm which is in the same spirit of `LineFilter`. For every incoming row $\mathbf{a}_i$ it computes the sampling probability based on its kernelized row $\acute{\mathbf{a}}_i$ and the counterpart of the previously seen rows. As the row $\acute{\mathbf{a}}_i$ only depends on $\mathbf{a}_i$, this algorithm can also be used in an online setting as well. Since we give a sampling based coreset, it retains the structure of the input data. So one need not require to kernelize $\mathbf{x}$ into its corresponding higher dimensional vector. Instead, one can use the same $\mathbf{x}$ on the sampled coreset to compute the desired operation. We call it `KernelFilter` and give it as algorithm 3. We summarize the guarantees of `KernelFilter` in the following two theorems.

Theorem 4.3. *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ whose rows are coming in a streaming manner and an **even** value p, the `KernelFilter` selects a set $\mathbf{C}$ of size $O(\frac{d^{p/2}k}{\epsilon^2}(1 + p(\log\|\mathbf{A}\| - d^{-p/2}\min_i \log\|\mathbf{a}_i\|))$ using working space and update time $O(d^p)$. Suppose $\mathbf{Q}$ is a fixed k-dimensional subspace, then with probability at least $0.99, \epsilon > 0, \forall \mathbf{x} \in \mathbf{Q}$*

Algorithm 3 `KernelFilter`

Require: Streaming rows $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$, $r > 1, p \geq 2$
Ensure: Coreset $\mathbf{C}$ satisfying eqn (1) and (2) w.h.p.
  $\acute{\mathbf{M}} = \acute{\mathbf{M}}_{inv} = 0^{d^{\lceil p/2 \rceil} \times d^{\lceil p/2 \rceil}}; L = 0; \mathbf{C} = \emptyset$
  $\acute{\mathbf{Q}} =$ orthonormal-column-basis($\acute{\mathbf{M}}$)
  while $i \leq n$ **do**
    $\acute{\mathbf{a}}_i = \text{vec}(\mathbf{a}_i \otimes^{\lceil p/2 \rceil})$
    $[\acute{e}_i, \acute{\mathbf{M}}, \acute{\mathbf{M}}_{inv}, \acute{\mathbf{Q}}] = \texttt{Score}(\acute{\mathbf{a}}_i, \acute{\mathbf{M}}, \acute{\mathbf{M}}_{inv}, \acute{\mathbf{Q}})$
    if (p is *even*) **then**
      $\tilde{l}_i = (\acute{e}_i)$
    else
      $\tilde{l}_i = (\acute{e}_i)^{p/(p+1)}$
    end if
    $L = L + \tilde{l}_i; p_i = \min\{r\tilde{l}_i/L, 1\}$
    Sample $\mathbf{a}_i / \sqrt[p]{p_i}$ in $\mathbf{C}$ with probability p_i
  end while

we have $\mathbf{C}$ *satisfying both* p*-order tensor contraction and* ℓ_p *subspace embedding as equations* (1) *and* (2) *respectively.*

Theorem 4.4. *Given* $\mathbf{A} \in \mathbb{R}^{n \times d}$ *whose rows are coming in a streaming manner and an* ***odd*** *integer* p, $p \geq 3$, *the algorithm* `KernelFilter` *selects a set* $\mathbf{C}$ *of size* $O\big(\frac{n^{1/(p+1)} d^{p/2} k}{\epsilon^2}(1 + (p+1)(\log \|\mathbf{A}\| - d^{-\lceil p/2 \rceil} \min_i \log \|\mathbf{a}_i\|))^{p/(p+1)}\big)$ *using working space and update time* $O(d^{p+1})$. *Suppose* $\mathbf{Q}$ *is a fixed* k*-dimensional subspace, then with probability at least* $0.99, \epsilon > 0, \forall \mathbf{x} \in \mathbf{Q}$ *we have* $\mathbf{C}$ *satisfying both* p*-order tensor contraction and* ℓ_p *subspace embedding as equations* (1) *and* (2) *respectively.*

The working space and the computation time of the above algorithm are functions of d and p. But compared to `LineFilter`, the `KernelFilter` returns an asymptotically smaller coreset. This is because the $\tilde{l}_i$ ensures a tighter upper bound of the online sensitivity score compared to what `LineFilter` gives. Note that although in the coreset size for odd value p there is factor of n but unlike `LineFilter` it decreases with increase in p.

4.4. `LineFilter+KernelFilter`

Here we briefly sketch our final algorithm. We use our `LineFilter` algorithm along with `KernelFilter` to give a streaming algorithm that benefits both in space and time. For every incoming row first the `LineFilter` quickly decides its sampling probability and samples according to it which is then passed to `KernelFilter` which returns the final coreset. Now we state the guarantee of `LineFilter+KernelFilter` in the following theorem.

Theorem 4.5. *Consider a matrix* $\mathbf{A} \in \mathbb{R}^{n \times d}$ *whose rows are coming one at a time and feed to the algorithm* `LineFilter+KernelFilter`, *which takes* $O(d^2)$ *amortized update time and uses* $O(d^{p+1})$ *working space for odd* p *and* $O(d^p)$ *for even to return* $\mathbf{C}$ *such that with at least* 0.99 *probability,* $\epsilon > 0, \forall \mathbf{x} \in \mathbf{Q}$, $\mathbf{C}$ *satisfies both* p*-order tensor contraction and* ℓ_p *subspace embedding as equations* (1) *and* (2) *respectively. With* $\mathbf{a}_{\min} = \arg\min_i \|\mathbf{a}_i\|$ *the size of* $\mathbf{C}$ *is as follows for integer* $p \geq 2$*:*

- p *even:* $O\big(\frac{d^{p/2} k}{\epsilon^2}(1 + p(\log \|\mathbf{A}\| - d^{-p/2} \log \|\mathbf{a}_{\min}\|))\big)$
- p *odd:* $O\big(\frac{n^{(p-2)/(p^2+p)} d^{p/2+1/4} k^{5/4}}{\epsilon^2}(1 + (p+1)(\log \|\mathbf{A}\| - d^{-\lceil p/2 \rceil} \log \|\mathbf{a}_{\min}\|))^{p/(p+1)}\big)$

Note that for integer $p \geq 2$, the factor $n^{(p-2)/(p^2+p)}$ in the odd p case can be upper bounded by $n^{1/10}$ and in general it is always $o(n^{1/(p+3)})$. Since the `LineFilter` only chooses a sub-linear number of samples, which are further passed to `KernelFilter`, hence the amortized update time of `LineFilter+KernelFilter` is same as the update time of `LineFilter`, whereas the working space is same as that of `KernelFilter`.

4.5. $p = 2$ case

In the case of a matrix, i.e., $p = 2$ the `LineFilter` and `KernelFilter` are just the same. This is because, for every incoming row $\mathbf{a}_i$, the kernelization returns the same row itself. Hence `KernelFilter`'s sampling process is exactly the same as `LineFilter`. While we use the sensitivity framework, for $p = 2$, our proofs are novel in the following sense:

1. When creating the sensitivity scores in the online setting, we do not need to use a regularization term as (Cohen et al., 2016), instead relying on a novel analysis when the matrix is rank deficient. Hence we get a relative error bound without making the number of samples depend on the smallest non zero singular value (which (Cohen et al., 2016) need for online row sampling for matrices).
2. We do not need to use a martingale based argument, since the sampling probability of a row does not depend on the previous samples.

Our algorithm gives a coreset which preserves relative error approximation (i.e., subspace embedding). Note that lemma 3.5 of (Cohen et al., 2016) can be used to achieve the same, but it requires the knowledge of $\sigma_{\min}(\mathbf{A})$ (smallest non zero singular value of $\mathbf{A}$). There we need to set $\delta = \epsilon \sigma_{min}(\mathbf{A})$ which gives sampling complexity as $O(d(\log d)(\log \kappa(\mathbf{A}))/\epsilon^2)$. Our algorithm gives relative error approximation even when $\kappa(\mathbf{A}) = 1$, which is not clear in (Cohen et al., 2016). We state our guarantees for $p = 2$ case in the appendix A.5.

5. Proofs

In this section, we state the supporting lemmas to prove our main theorems in the previous section. We give sketch of proof for some lemmas. The detailed proofs are discussed in appendix A.

5.1. `LineFilter`

Here we give a sketch of the proof for theorem 4.1. For ease of notation, the rows are considered numbered according to their order of arrival. The supporting lemmas are for the online setting, which also works for the streaming case. We show that it is possible to generalize the notion of sensitivity for the online setting as well as give an upper bound to it. We define the *online sensitivity* of any i^{th} row as : $\sup_{\mathbf{x}\in\mathbf{Q}} \frac{|\mathbf{a}_i^T\mathbf{x}|^p}{\sum_{j=1}^{i}|\mathbf{a}_j^T\mathbf{x}|^p} = \sup_{\mathbf{y}\in\mathbf{Q}'} \frac{|\mathbf{u}_i^T\mathbf{y}|^p}{\sum_{j=1}^{i}|\mathbf{u}_j^T\mathbf{y}|^p}$, where $\mathbf{Q}' = \{\mathbf{y}|\mathbf{y} = \Sigma\mathbf{V}^T\mathbf{x}, \mathbf{x} \in \mathbf{Q}\}$, $\text{svd}(\mathbf{A}) = \mathbf{U}\Sigma\mathbf{V}^T$ and $\mathbf{Q}$ is the query space. Notice that the denominator now contains a sum only over the rows that have arrived. We note that while online sampling results often need the use of martingales as an analysis tool, e.g., (Cohen et al., 2016), in our setting, the sampling probability of each row does depend on the previous rows, but not on whether they were sampled or not. So, the sampling decision of each row is independent. Hence, the application of Bernstein's inequality A.1. suffices.

We first show that the $\tilde{l}_i$, as defined in `LineFilter` used to compute the sampling probability p_i, are upper bounds to the online sensitivity scores.

Lemma 5.1. *Consider $\mathbf{A} \in \mathbb{R}^{n\times d}$, whose rows are provided in a streaming manner to `LineFilter`. Let $\tilde{l}_i = \min\{i^{p/2-1}(\mathbf{a}_i^T\mathbf{M}^\dagger\mathbf{a}_i)^{p/2}, 1\}$, and $\mathbf{M}$ is a $d \times d$ matrix maintained by the algorithm. Then $\forall i \in [n]$, $\tilde{l}_i$ satisfies $\tilde{l}_i \geq \sup_{\mathbf{x}} \frac{|\mathbf{a}_i^T\mathbf{x}|^p}{\sum_{j=1}^{i}|\mathbf{a}_j^T\mathbf{x}|^p}$.*

The proof of this lemma is discussed in the appendix A.1.1. Although the $\tilde{l}_i$'s are computed very quickly but the algorithm gives a loose upper bound due to the factor of $i^{p/2-1}$ in the definition of $\tilde{l}_i$. As i increases, the $\tilde{l}_i$ also increases. Now with these upper bounds we get the following.

Lemma 5.2. *Let r provided to `LineFilter` be $O(k\epsilon^{-2}\sum_{j=1}^{n}\tilde{l}_j)$. Let `LineFilter` returns a coreset $\mathbf{C}$. Then with probability at least 0.99, $\forall\mathbf{x} \in \mathbf{Q}$, $\mathbf{C}$ satisfies the p-order tensor contraction as in equation (1) and ℓ_p subspace embedding as in equation (2).*

The proof of this lemma is given in the appendix A.1.2. Now in order to bound the number of samples, we need a bound on the quantity $\sum_{j=1}^{n}\tilde{l}_j$ which we demonstrate in the following lemma.

Lemma 5.3. *The $\tilde{l}_i$ in `LineFilter` algorithm which satisfies lemma 5.1 and lemma 5.2 has $\sum_{i=1}^{n}\tilde{l}_i = O(n^{1-2/p}(d + d\log\|\mathbf{A}\| - \min_i\log\|\mathbf{a}_i\|))$.*

Proof. Here we give a sketch of the proof. The detail proof is discussed in appendix A.1.3. First we bound $\sum_{i\leq n}\tilde{e}_i$, where $\tilde{e}_i$ are the scores returned by `Score`$(\cdot)$. At any time i the algorithm gets $\mathbf{a}_i$ and it has $\mathbf{M} = \mathbf{A}_{i-1}^T\mathbf{A}_{i-1}$ using which it computes $\tilde{e}_i = \mathbf{a}_i^T(\mathbf{A}_i^T\mathbf{A}_i)^\dagger\mathbf{a}_i$. With incoming row $\mathbf{a}_i$ the rank of $\mathbf{M}$ increases from 1 to at most d. We say that the algorithm is in phase-k if the rank of $\mathbf{M}$ equals k. For each phase $k \in [1, d]$, let i_k denote the index where row $\mathbf{a}_{i_k}$ caused a phase-change in $\mathbf{M}$ i.e. rank of $(\mathbf{A}_{i_k-1}^T\mathbf{A}_{i_k-1})$ is $k-1$, but rank of $(\mathbf{A}_{i_k}^T\mathbf{A}_{i_k})$ is k. Note that for such i_k, $\tilde{e}_{i_k} = 1$. There are at most d such indices. Now for a phase k with $i \in [i_k + 1, i_{k+1} - 1]$ we bound the term $\tilde{e}_i$. Suppose the thin-SVD$(\mathbf{A}_{i_k}^T\mathbf{A}_{i_k}) = \mathbf{V}\Sigma_{i_k}\mathbf{V}^T$, all entries in Σ_{i_k} being positive. We define $\mathbf{X}_{i_k} = \mathbf{V}^T(\mathbf{A}_{i_k}^T\mathbf{A}_{i_k})\mathbf{V}$, where $\mathbf{X}_{i_k} = \Sigma_{i_k}$ is positive definite and hence full rank. Similarly, for all i in the range we define $\mathbf{X}_i = \mathbf{V}^T(\mathbf{A}_i^T\mathbf{A}_i)\mathbf{V}$ where $\mathbf{X}_i$ is positive definite. Further, for the same range of i we have $\mathbf{a}_i = \mathbf{V}\mathbf{b}_i$ with $\mathbf{X}_i = \mathbf{X}_{i-1} + \mathbf{b}_i\mathbf{b}_i^T$. So, $\tilde{e}_i = \mathbf{a}_i^T(\mathbf{A}_i^T\mathbf{A}_i)^\dagger\mathbf{a}_i = \mathbf{b}_i^T\mathbf{V}^T(\mathbf{V}(\mathbf{X}_{i-1} + \mathbf{b}_i\mathbf{b}_i^T)\mathbf{V}^T)^\dagger\mathbf{V}\mathbf{b}_i = \mathbf{b}_i^T(\mathbf{X}_{i-1} + \mathbf{b}_i\mathbf{b}_i^T)^{-1}\mathbf{b}_i$. Using matrix determinant lemma (Harville, 1997) on $\det(\mathbf{X}_{i-1} + \mathbf{b}_i\mathbf{b}_i^T)$

$$\begin{aligned} &= \det(\mathbf{X}_{i-1})(1 + \mathbf{b}_i^T(\mathbf{X}_{i-1})^{-1}\mathbf{b}_i) \\ &\geq \det(\mathbf{X}_{i-1})(1 + \mathbf{b}_i^T(\mathbf{X}_{i-1} + \mathbf{b}_i\mathbf{b}_i^T)^{-1}\mathbf{b}_i) \\ &= \det(\mathbf{X}_{i-1})(1 + \tilde{e}_i) \geq \det(\mathbf{X}_{i-1})\exp(\tilde{e}_i/2) \end{aligned}$$

So finally we get $\exp(\tilde{e}_i/2) \leq \frac{\det(\mathbf{X}_{i-1}+\mathbf{b}_i\mathbf{b}_i^T)}{\det(\mathbf{X}_{i-1})}$. Now computing other $\tilde{e}_i$'s and by a careful analysis we get $\sum_{i\leq n}\tilde{e}_i \leq \tilde{e}_1 - \tilde{e}_n$, which is then used to bound $\sum_{i\leq n}\tilde{l}_i$. □

With lemmas 5.1, 5.2 and 5.3 we prove that the guarantee in theorem 4.1 is achieved by `LineFilter`. The bound on space is evident from the fact that we are maintaining the matrix $\mathbf{M}$ in algorithm which uses $O(d^2)$ space and returns a coreset of size $O(\frac{n^{1-2/p}dk}{\epsilon^2}(1 + \log\|\mathbf{A}\| - d^{-1}\min_i\log\|\mathbf{a}_i\|))$. Further these lemma's can also be used to show that `LineFilter` ensures ℓ_p subspace embedding for any real value $p \geq 2$.

5.2. `KernelFilter`

In this section, we give a sketch of the proof of theorem 4.3. We use sensitivity based framework to decide the sampling probability of each incoming row. The novelty in this algorithm is by reducing the p order operation to a q order, where q is either 2 or less than but very close to 2. Now we give bound on sensitivity score of every incoming row.

Lemma 5.4. *Consider $\mathbf{A} \in \mathbb{R}^{n\times d}$, whose rows are provided in a streaming manner to `KernelFilter`. The term $\tilde{l}_i$ defined in the algorithm upper bounds the online sensitivity score, i.e. $\forall i \in [n], \tilde{l}_i \geq \sup_{\mathbf{x}} \frac{|\mathbf{a}_i^T\mathbf{x}|^p}{\sum_{j=1}^{i}|\mathbf{a}_j^T\mathbf{x}|^p}$.*

The proof is discussed in the appendix A.4.1. Unlike `LineFilter`, the `KernelFilter` does not use any additional factor of i. Hence it gives tighter upper bounds to the sensitivity scores compared to lemma 5.1. It will be evident when we sum these upper bounds. Next we show with these $\tilde{l}_i$'s we can claim the following,

Lemma 5.5. *In the `KernelFilter` let r is set as $O(k\sum_{i=1}^{n}\tilde{l}_i/\epsilon^2)$ then for some fixed k-dimensional subspace $\mathbf{Q}$, the set $\mathbf{C}$ with probability 0.99 $\forall \mathbf{x} \in \mathbf{Q}$ satisfies p-order tensor contraction as in equation (1) and ℓ_p subspace embedding as in equation (2).*

We discuss this in detail in the appendix A.4.2. Next we bound the sum of upper bounds, i.e. $\sum_{i\leq n}\tilde{l}_i$.

Lemma 5.6. *Let $\mathbf{a}_{\min} = arg\min_i \|\mathbf{a}_i\|$ and $\tilde{l}_i$'s used in `KernelFilter` which satisfy lemma 5.4 and 5.5 has $\sum_{i=1}^{n}\tilde{l}_i$ as*

- *p even:* $O(d^{p/2}(1+p(\log\|\mathbf{A}\| - d^{-p/2}\log\|\mathbf{a}_{\min}\|)))$
- *p odd:* $O(n^{1/(p+1)}d^{p/2}(1+(p+1)(\log\|\mathbf{A}\| - d^{-\lceil p/2\rceil}\log\|\mathbf{a}_{\min}\|))^{p/(p+1)})$

We discuss the proof in detail in the appendix A.4.3. The proof of the above lemma is similar to that of lemma 5.3. It implies that the lemma 5.4 gives tighter sensitivity bounds compared to lemma 5.1. Now with lemmas 5.4, 5.5 and 5.6 we prove that the guarantees in theorem 4.3 and theorem 4.4 is achieved by `KernelFilter`. The bound on space $O(d^p)$ for even p and $O(d^{p+1})$ for odd p is for maintaining $\acute{\mathbf{M}}$ and $\acute{\mathbf{M}}_{inv}$. Note that for even value p our coreset is independent of n, where as for odd value p we have a small factor of n. Unlike `LineFilter` this factor reduces with increases in p. Note for p as $O(\log n)$ the factor of n is $O(1)$. Hence coreset $\mathbf{C}$ will be just $\tilde{O}(d^{p/2}k\epsilon^{-2})$.

6. Applications

Here we show how our methods can also be used for learning latent variable models using tensor factorization. We use a corollary, which summarizes the guarantees we get on latent variables by learning them using tensor factorization on our coreset. We discuss it in the appendix A.6. Note that one can always use an even order tensor for estimating the latent variables in a generative model. It will only increase a factor of $O(d)$ in the working space, but by doing so, it will return a smaller coreset, which is independent of n.

Streaming Single Topic Model: We empirically show how sampling using `LineFilter`+`KernelFilter` can preserve tensor contraction as in equation (1). This can be used in single topic modeling where documents are coming in a streaming manner. We compare our method with two other sampling schemes, namely – `Uniform` and online leverage scores, which we call `LineFilter`(2).

Here we use a subset of **20Newsgroups** dataset (preprocessed). We took a subset of 10K documents and considered the 100 most frequent words. We normalized each document vector, such that its ℓ_1 norm is 1 and created a matrix $\mathbf{A} \in \mathbb{R}^{10\text{K}\times 100}$. We feed its row one at a time to `LineFilter`+`KernelFilter` with $p = 3$, which returns a coreset $\mathbf{C}$. We run tensor based single topic modeling (Anandkumar et al., 2014) on $\mathbf{A}$ and $\mathbf{C}$, to return 12 top topic distributions from both. We take the best matching between empirical topics and estimated topics based on ℓ_1 distance and compute the average ℓ_1 difference between them. Here smaller is better. We run this entire method 5 times and report the median of their ℓ_1 average differences. Here the coreset sizes are over expectation.

Table 2. Streaming Single Topic Modeling

SAMPLE	UNIFORM	LINEFILTER(2)	LINEFILTER +KERNELFILTER
50	0.5725	0.6903	**0.5299**
100	0.5093	0.6385	**0.4379**
200	0.4687	0.5548	**0.3231**
500	0.3777	0.3992	**0.2173**
1000	0.2548	0.2318	**0.1292**

From the table, it can be seen that our algorithm `LineFilter`+`KernelFilter` performs better compare to both `Uniform` and `LineFilter`(2), thus supporting our theoretical claims.

Conclusion: In this work, we presented both online and streaming algorithms to create coresets for tensor and ℓ_p subspace embedding, and showed their applications in latent factor models. The algorithms either match or improve upon a number of existing algorithms for ℓ_p subspace embedding for all integer $p \geq 2$. The core of our approach is using a combination of a fast online subroutine `LineFilter` for filtering out most rows and a more expensive subroutine for better subspace approximation. Obvious open questions include extending the techniques to $p = 1$ as well as improving the coreset size for `KernelFilter`, for odd-p. It will also be interesting to explore the connection of `KernelFilter` to Lewis weights (Cohen & Peng, 2015), since both are different ways of mapping the ℓ_p problem to ℓ_2. Further, it will also be interesting to explore both theoretically and empirically that how the randomized CP decomposition (Battaglino et al., 2018; Erichson et al., 2020) performs in various latent variable models.

Acknowledgements. We are grateful to the anonymous reviewers for their helpful feedback. Anirban acknowledges the kind support of the N. Rama Rao Chair Professorship at IIT Gandhinagar, the Google India AI/ML award (2020), Google Faculty Award (2015), and CISCO University Research Grant (2016). Supratim acknowledges the kind support of Additional Fellowship from IIT Gandhinagar.

References

Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

Anandkumar, A., Hsu, D., and Kakade, S. M. A method of moments for mixture models and hidden markov models. In *Conference on Learning Theory*, pp. 33–1, 2012.

Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M., and Telgarsky, M. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

Bachem, O., Lucic, M., and Krause, A. Practical coreset constructions for machine learning. *stat*, 1050:4, 2017.

Battaglino, C., Ballard, G., and Kolda, T. G. A practical randomized cp tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018.

Bhojanapalli, S. and Sanghavi, S. A new sampling technique for tensors. *stat*, 1050:19, 2015.

Braverman, V., Feldman, D., and Lang, H. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

Clarkson, K. L., Drineas, P., Magdon-Ismail, M., Mahoney, M. W., Meng, X., and Woodruff, D. P. The fast cauchy transform and faster robust linear regression. *SIAM Journal on Computing*, 45(3):763–810, 2016.

Cohen, M. B. and Peng, R. L p row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 183–192. ACM, 2015.

Cohen, M. B., Lee, Y. T., Musco, C., Musco, C., Peng, R., and Sidford, A. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pp. 181–190. ACM, 2015.

Cohen, M. B., Musco, C., and Pachocki, J. Online row sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1758–1777. SIAM, 2017.

Dasgupta, A., Drineas, P., Harb, B., Kumar, R., and Mahoney, M. W. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

Dickens, C., Cormode, G., and Woodruff, D. Leveraging well-conditioned bases: Streaming and distributed summaries in minkowski p-norms. In *International Conference on Machine Learning*, pp. 1243–1251, 2018.

Dubhashi, D. P. and Panconesi, A. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

Erichson, N. B., Manohar, K., Brunton, S. L., and Kutz, J. N. Randomized cp tensor decomposition. *Machine Learning: Science and Technology*, 1(2):025012, 2020.

Feldman, D. and Langberg, M. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 569–578. ACM, 2011.

Har-Peled, S. and Mazumdar, S. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pp. 291–300. ACM, 2004.

Harville, D. Matrix algebra from a statistician's perspective. Technical report, Springer-Verlag, 1997.

Haussler, D. and Welzl, E. ϵ-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.

Hillar, C. J. and Lim, L.-H. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

Hsu, D., Kakade, S. M., and Zhang, T. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Huang, F., Niranjan, U., Hakeem, M. U., and Anandkumar, A. Online tensor methods for learning latent variable models. *The Journal of Machine Learning Research*, 16 (1):2797–2835, 2015.

Janzamin, M., Sedghi, H., and Anandkumar, A. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv preprint arXiv:1506.08473*, 2015.

Jenatton, R., Roux, N. L., Bordes, A., and Obozinski, G. R. A latent factor model for highly multi-relational data. In *Advances in Neural Information Processing Systems*, pp. 3167–3175, 2012.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Kruskal, J. B. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.

Langberg, M. and Schulman, L. J. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pp. 598–607. SIAM, 2010.

Ma, T., Shi, J., and Steurer, D. Polynomial-time tensor decompositions with sum-of-squares. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 438–446. IEEE, 2016.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Schechtman, G. Tight embedding of subspaces of l_p in ℓ_p^n for even. *Proceedings of the American Mathematical Society*, 139(12):4419–4421, 2011.

Sherman, J. and Morrison, W. J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21 (1):124–127, 1950.

Song, Z., Woodruff, D., and Zhang, H. Sublinear time orthogonal tensor decomposition. In *Advances in Neural Information Processing Systems*, pp. 793–801, 2016.

Song, Z., Woodruff, D. P., and Zhong, P. Relative error tensor low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2772–2789. Society for Industrial and Applied Mathematics, 2019.

Tropp, J. et al. Freedman's inequality for matrix martingales. *Electronic Communications in Probability*, 16:262–270, 2011.

Tropp, J. A. et al. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

Vervliet, N., Debals, O., Sorber, L., Van Barel, M., and De Lathauwer, L. Tensorlab 3.0. available online. *URL: http://www. tensorlab. net*, 2016.

Wang, Y. and Anandkumar, A. Online and differentially-private tensor decomposition. In *Advances in Neural Information Processing Systems*, pp. 3531–3539, 2016.

Wang, Y., Tung, H.-Y., Smola, A. J., and Anandkumar, A. Fast and guaranteed tensor decomposition via sketching. In *Advances in Neural Information Processing Systems*, pp. 991–999, 2015.

Woodruff, D. and Zhang, Q. Subspace embeddings and ℓ_p-regression using exponential random variables. In *Conference on Learning Theory*, pp. 546–567, 2013.

Woodruff, D. P. et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.