Matrix Sketching in Data Streams

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu
In many applications, we can represent data as a matrix: e.g. text analysis, recommendation.
Data as a Matrix

- Think of data as \( A \in \mathbb{R}^{n \times d} \) containing \( n \) row vectors in \( \mathbb{R}^d \), and typically \( n \gg d \)

- Some examples of typical web-scale data:

<table>
<thead>
<tr>
<th>Data</th>
<th>Rows</th>
<th>Columns</th>
<th>( n )</th>
<th>( d )</th>
<th>sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual</td>
<td>Documents</td>
<td>Words</td>
<td>( &gt; 10^{10} )</td>
<td>( 10^5 - 10^7 )</td>
<td>yes</td>
</tr>
<tr>
<td>Visual</td>
<td>Images</td>
<td>Pixels, SIFT</td>
<td>( &gt; 10^8 )</td>
<td>( 10^5 - 10^6 )</td>
<td>no</td>
</tr>
<tr>
<td>Audio</td>
<td>Songs</td>
<td>Frequencies</td>
<td>( &gt; 10^8 )</td>
<td>( 10^5 - 10^6 )</td>
<td>no</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Examples</td>
<td>Features</td>
<td>( &gt; 10^6 )</td>
<td>( 10^2 - 10^4 )</td>
<td>yes/no</td>
</tr>
<tr>
<td>Financial</td>
<td>Prices</td>
<td>Items, Stocks</td>
<td>( &gt; 10^6 )</td>
<td>( 10^3 - 10^5 )</td>
<td>no</td>
</tr>
</tbody>
</table>
Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A.

**Rank-k Approximation**

Given $A \in \mathbb{R}^{n \times d}$ with $\text{rank}(A) = r$, compute a concise matrix $B$ with rank $k \ll r$ such that it approximates $A$ "accurately".
Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

- B is much smaller than A that it fits in memory
- Rank(B) << rank(A)
  - If A is a document-term matrix with 10 billion documents and 1 million words $A \in \mathbb{R}^{10^{10}\times10^6}$ then B would probably be $B \in \mathbb{R}^{1000\times106}$
**Review: rank-k approximation**

- **Rank-k approximation** to $A$ computes a smaller matrix $B$ of rank $k$ such that $B$ approximates $A$.

<table>
<thead>
<tr>
<th>Rank-$k$ Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given $A \in \mathbb{R}^{n \times d}$ with $\text{rank}(A) = r$, compute a concise matrix $B$ with rank $k \ll r$ such that it approximates $A$ &quot;accurately&quot;.</td>
</tr>
</tbody>
</table>
Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A.

<table>
<thead>
<tr>
<th>Rank-k Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given $A \in \mathbb{R}^{n \times d}$ with $\text{rank}(A) = r$, compute a concise matrix $B$ with rank $k \ll r$ such that it approximates $A$ &quot;accurately&quot;.</td>
</tr>
</tbody>
</table>

- Error difference between A and B is small:
**Review: rank-k approximation**

- Rank-k approximation to $A$ computes a smaller matrix $B$ of rank $k$ such that $B$ approximates $A$

**Rank-k Approximation**

Given $A \in \mathbb{R}^{n \times d}$ with $\text{rank}(A) = r$, compute a concise matrix $B$ with rank $k \ll r$ such that it approximates $A$ "accurately".

- Error difference between $A$ and $B$ is small:
  - The covariance error $\|A^T A - B B^T\|_{2,F}$ is small
**Review: rank-k approximation**

- **Rank-k approximation** to $A$ computes a smaller matrix $B$ of rank $k$ such that $B$ approximates $A$.

  \[
  \text{Given } A \in R^{n \times d} \text{ with } \text{rank}(A) = r, \text{ compute a concise matrix } B \text{ with }
  \text{rank } k \ll r \text{ such that it approximates } A \text{ "accurately".}
  \]

- **Error difference between $A$ and $B$ is small:**
  - The **covariance error** $\|A^T A - BTB\|_{2,F}$ is small
  - The **projection error** $\|A - \Pi_B(A)\|_{2,F}$ is small
    - $\Pi_B A :=$ projecting rows of $A$ onto the subspace of $B$
    - If $B = USV^T$ then, the subspace of $B$ is $VV^T$
    - Therefore $\Pi_B A = AVV^T$
Best Rank-k Approximation

- We saw that SVD computes the best rank-k approximation to A

\[
A = U \Sigma V^T
\]

- left singular vectors
- singular values
- right singular vectors
SVD computes the **best** rank-k approximation

\[ A_k = \arg \min_{\text{rank}(B) \leq k} \| A - B \|_{F,2} \]

So the desirable approximation error is

\[
\| A - \Pi_B(A) \|_{2, F} \leq c \| A - A_k \|_{2, F} \quad \text{or} \quad \| A^T A - B T B \|_{2, F} \leq c \| A - A_k \|_{2, F}
\]
Best Rank-k Approximation

- SVD computes the **best** rank-k approximation to $A$
  - SVD requires $O(nd^2)$ time and $O(nd)$ space
  - Not applicable in streaming, or distributed settings
  - Not efficient for sparse matrices
Can we compute rank-k approximation in streaming setting?
Streaming matrix sketching
Streaming data matrix

- Every element of the stream is a row vector of fixed $d$-dimension.
  - We’d like to process $A$ in one pass and using a small amount of memory (sublinear in $n$)
Streaming data such as any time series data:

- ecommerce purchases
- Traffic sensors
- Activity logs

We can not store the entire data
A large set of data analysis tasks rely on obtaining a **low rank approximation**:

- Dimension reduction
- Anomaly detection
- Data denoising
- Clustering
- Recommendation systems
B is a **sketch** of a streaming matrix A iff

- B is of a fixed **small size** that fits in memory
- At any point in stream, B approximates A
Almost any matrix sketching methods in streaming setting falls into one of these categories:

1. Row sampling based
2. Random projection based and Hashing
3. Iterative sketching
Row Sampling Methods
They select a subset of “important” rows
- Sample w.r.t a well-defined probability distribution
- Often sampling is done with replacement

Methods differ in how they define “importance”
They construct sketch B by:

- assign a probability $p_i$ to each row $a_i$
- sample $l$ rows from A to construct B
- rescale B appropriately to make it unbiased
An Intuitive way to define “importance” of an item:
- the weight associated to the item, e.g.
  - file records → weights as size of the file,
  - IP addresses → weights as number of times the IP address makes a request

why it is necessary to sample important items?
- Consider a set of weighted items $S = \{(a_1, w_1), (a_2, w_2), \ldots, (a_n, w_n)\}$ that we want to summarize with a small & representative sample.

We define a representative sample as the one estimates total weight of $S$ (i.e. $W_S = \sum_{i=1}^{n} w_i$) in expectation.
Intuition: Row Sampling Methods

- This is achievable with a sample set of size one!
  - Sample any item \((a_j, w_j)\) with an arbitrary fixed probability \(p\), and rescale its weight to \(W_s/p\).
  - Then \(E[\text{weight of the sample}] = p. \ W_s/p = W_s\)

- High variance issue:
  - To lower down the variance, (1) sample heavy items (i.e. important items) with higher prob., and (2) sample more items
  - So sample item \(a_j\) with prob. \(p = w_j/W_s\) and rescale it to \(W_s/p\)
  - If we sample \(l\) items, then rescale items to rescale it to \(W_s/(lp)\)
Row Sampling algorithms

- In matrices,
  - Each item $a_j$ is a row vector
  - Each weight $w_j = \|a_j\|^2$
  - And $\sum_{j=1}^{n} \|a_j\|^2 = \|A\|^2_F$

- Row sampling algorithm based on L2 norm:
  - Let sample size = $l$, i.e. the sketch $B$ is $l \times d$
  - For every row $a_i$ arriving in the stream,
    - Update $\|A\|^2_F$ by adding $\|a_j\|^2_F$
    - Compute its sampling probability $p_i = \|a_i\|^2 / \|A\|^2_F$
    - Sample it $l$ times (one for each row of $B$. If it is sampled, replace the corresponding row in $B$ with $a_i$)
    - Rescale $a_i$ where it is sampled by $1/\sqrt{l \cdot p_i}$
Row Sampling algorithms

- We can show that

\[ E[\|B\|_F] = \|A\|_F \]

- If we sample \( \ell = O(k/\varepsilon^2) \) rows, then:

\[ \|A - \pi_B(A)\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon\|A\|_F^2 \]
Row sampling based on L2 norm:

- CUR method: samples rows/columns with probability = squared norm of rows/columns

\[
\begin{pmatrix}
A \\
\end{pmatrix}
\approx
\begin{pmatrix}
\begin{pmatrix}
\end{pmatrix}
\end{pmatrix}
\cdot
\begin{pmatrix}
U \\
\end{pmatrix}
\cdot
\begin{pmatrix}
R
\end{pmatrix}
\]
CUR: Row/column sampling

- Row sampling based on L2 norm:
  - CUR method: samples rows/columns with probability = squared norm of rows/columns

\[
\begin{pmatrix}
A \\
C \\
\end{pmatrix}
\approx
\begin{pmatrix}
C \\
U \\
\end{pmatrix}
\cdot
\begin{pmatrix}
R \\
\end{pmatrix}
\]

Pseudo-inverse of the intersection of \(C\) and \(R\)
CUR: Row/column sampling

- Row sampling based on L2 norm:
  - CUR method: samples rows/columns with probability = squared norm of rows/columns

- Error guarantee: If we sample $c = O\left(\frac{k \log k}{\varepsilon^2}\right)$ columns and $r = O\left(\frac{k \log k}{\varepsilon^2}\right)$ rows, then

  $$\|A - CUR\|_F \leq (2 + \varepsilon)\|A - A_K\|_F$$

  With probability $\geq 98\%$
Row Sampling Methods

+ **Easy interpretation of basis**
  - Since the basis vectors are actual rows/columns

+ **Suitable for Sparse data**
  - Since the basis vectors are actual rows/columns

- **Duplicate columns and rows**
  - Columns of large norms will be sampled multiple times
Random Projection Methods
Key idea: if points in a vector space are projected onto a randomly selected subspace of suitably high dimension, then the distances between points are approximately preserved.

Johnson-Lindenstrauss Transform (JLT): \(d\) datapoints in any dimension (\(\mathbb{R}^n\) for \(n \gg d\)) can get embedded into roughly \(\log d\) dimensional space, such that their pair-wise distances are preserved to some extent.
We define JLT more precisely:

- A random matrix $S \in \mathbb{R}^{r \times n}$ has JLT property if for all vectors $v, v' \in \mathbb{R}^n$,
  \[ \|Sv - Sv'\|^2 = (1 \pm \epsilon)\|v - v'\|^2 \]
  with probability at least $1 - \delta$

- There are many ways to construct a matrix $S$ that preserve pair-wise distances.
  - All such matrices are called to have the Johnson-Lindenstrauss Transform (JLT) property
How to construct a JLT matrix

One simple construction of $S$:

- Pick matrix $S \in \mathbb{R}^{r \times n}$ as an orthogonal projection on a random $r$-dimensional subspace of $\mathbb{R}^n$ with $r = O(\varepsilon^{-2} \log d)$
  - Rows of $S$ are orthogonal vectors

- Then for any matrix $A \in \mathbb{R}^{n \times d}$, $SA$ preserves pair-wise distances between $d$ datapoints in $A$
A simpler construction for $S \in \mathbb{R}^{r \times n}$ is:
- to have entries as independent random variables with the standard normal distribution

$$S = \sqrt{\frac{1}{r}} \left[ \text{matrix with entries draw from } N(0,1) \right]$$
Another construction for $S \in \mathbb{R}^{r \times n}$ is:

$$S = \sqrt{\frac{1}{r}} \text{[entries as independent +/-1 random var]}$$

This is computationally simpler to construct.
Random Projection Methods

- They use a JLT matrix $S \in \mathbb{R}^{r \times n}$
- Construct the sketch as $B = SA \in \mathbb{R}^{r \times d}$
  - this projects datapoints from a high-dim space $\mathbb{R}^n$ onto a lower-dim subspace $\mathbb{R}^r$
- They show $\mathbb{E}[B^T B] = A^T \mathbb{E}[S^T S] A = A^T A$

$E[S^T S] = I_n$
Random Projection Methods

- Depending on JLT construction, we achieve different error bounds:
  - If $S \in \mathbb{R}^{r \times n}$ has iid zero-mean $\pm 1$ entries and $r = O\left(\frac{k}{\varepsilon} + k \log k\right)$ and, then

\[
\|A - \pi_{SA}(A)\|_F \leq (1 + \varepsilon)\|A - A_k\|_F
\]
Random Projection Methods

- Computationally efficient
- Sufficiently accurate in practice
- A great pre-processing step in applications

- **Data-oblivious** as their computation involves only a random matrix $S$
  - Compare to row sampling methods that need to access data to form a sketch
Matrix Hashing Techniques

- Use matrix $S$ that contains one $\pm 1$ per column

Only one non-zero entry in each column of $S$. The rest of entries are zero

- To build $S$, use two hash functions:
  - $h: [n] \rightarrow [r]$, and $g:[n] \rightarrow \{-1, +1\}$
Matrix Hashing Techniques

- Very efficient for sparse matrices $A$
  - can be applied in $O(\text{nnz}(A))$ operations
  - $\text{nnz}(A) = \text{number of non-zeros of } A$

$S$

$h(i)$

$A$

$B$

set $S_{h(i),i} = \pm 1$
Iterative Sketching
They work over a stream $A = \langle a_1, a_2, \ldots, a_n \rangle$

- each $a_i$ is read once, get processed quickly and not read again
- with only a small amount of memory available
State of the art method in this group is called “Frequent Directions”

It is based on Misra-Gries algorithm for finding frequent items in a data stream

We first see how Misra-Gries algorithm for finding frequent items work
  - Then we extend it to matrices
Suppose there is a stream of items, and we want to find frequency $f(i)$ of each item.
If we keep $d$ counters, we can count frequency of every item...

- But it’s not good enough (IP addresses, queries,...)
Let’s keep $l$ counters where $l \ll d$
If a new item arrives in the stream that is already in the counters, we add 1 to its count.
If the new item is not in the counters and we have space, we create a counter for it and set it to 1.
Frequent Items: Misra-Gries

- But what if we don’t have space for it?
Let $\delta$ be the median counter at time $t$. 

\[ \delta = \ell/2 = 2 \]
Decrease all counts by $\delta$ (set it to 0 if less than $\delta$)
Now we have space for new item, so we continue...
Frequent Items: Misra-Gries

- At any time in the stream, the approximated counts for items are what we have kept so far.
Frequent Items: Misra-Gries

- This method undercounts so for any item $i$
  
  \[ 0 \leq f'(i) \leq f(i) \]

- We decrease each count by at most $\delta_t$

  \[
  f'(i) \geq f(i) - \sum \delta_t
  \]

- At any point that we have seen $n$ elements in stream:

  \[
  \frac{l}{2} \sum \delta_t \leq n
  \]

- The error guarantee: \( 0 \leq f(i) - f'(i) \leq 2n/l \)
Misra-Gries produces a non-zero approximated frequency $f'(i)$ for all items that their true frequency $f(i) > 2n/l$

$$f(i) - 2n/l \leq f'(i)$$

To find items that appear more than 20% of the time i.e. $f(i) > n/5$, take $l = 10$ counters and run Misra-Gries algorithm
Let’s extend it to vectors and matrices

Stream items are row vectors in $d$ dimension

At any time $n$ in the stream, they form a tall matrix $A \in \mathbb{R}^{n \times d}$

The goal is to find the most frequent directions of $A$
Frequent Directions

**Input:** $A \in \mathbb{R}^{n \times d}$, and an integer $\ell$

$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$

**for** $(a_i \in A)$

- Insert $a_i$ into $B$

  **if** $(B$ is full)$

  $[U, S, V] \leftarrow \text{svd}(B)$

  $\tilde{S} \leftarrow [\sqrt{S_1^2-S_{\ell/2}^2}, \sqrt{S_2^2-S_{\ell/2}^2}, \ldots, 0, \ldots, 0]$}

  $B \leftarrow \tilde{S} V^T$

**return** $B$
Frequent Directions (Lib’13)

**Input:** \( A \in \mathbb{R}^{n \times d} \), and an integer \( \ell \)

\( B \leftarrow \) empty matrix \( \in \mathbb{R}^{\ell \times d} \)

for \( (a_i \in A) \)

- Insert \( a_i \) into \( B \)

if \( (B \) is full) 

\[
[U, S, V] \leftarrow \text{svd}(B)
\]

\[
\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{i/2}^2}, \sqrt{S_2^2 - S_{i/2}^2} \ldots 0, \ldots, 0]
\]

\( B \leftarrow \tilde{S}V^T \)

return \( B \)
Frequent Directions

\textbf{Input:} \( A \in \mathbb{R}^{n \times d} \), and an integer \( \ell \)
\( B \leftarrow \) empty matrix \( \in \mathbb{R}^{\ell \times d} \)
\textbf{for} \( (a_i \in A) \)
\begin{itemize}
  \item Insert \( a_i \) into \( B \)
  \item \textbf{if} \ (B \text{ is full})
  \begin{align*}
    \begin{bmatrix} U, S, V \end{bmatrix} &\leftarrow \text{svd}(B) \\
    \tilde{S} &\leftarrow \begin{bmatrix} \sqrt{S_1^2 - S_{\ell/2}^2}, \sqrt{S_2^2 - S_{\ell/2}^2}, \ldots, 0, \ldots, 0 \end{bmatrix}
  \end{align*}
  \item \( B \leftarrow \tilde{S}V^T \)
\end{itemize}
\textbf{return} \( B \)
### Frequent Directions (Lib’13)

**Input:** \( A \in \mathbb{R}^{n \times d} \), and an integer \( \ell \)

\[ B \leftarrow \text{empty matrix} \in \mathbb{R}^{\ell \times d} \]

**for** \( (a_i \in A) \)
- Insert \( a_i \) into \( B \)
- **if** \( (B \text{ is full}) \)
  - \( [U, S, V] \leftarrow \text{svd}(B) \)
  - \( \tilde{S} \leftarrow \begin{bmatrix} \sqrt{S_1^2 - S_{l/2}^2} & \sqrt{S_2^2 - S_{l/2}^2} & \ldots & 0, \ldots, 0 \end{bmatrix} \)
  - \( B \leftarrow \tilde{S} V^T \)

**return** \( B \)
Frequent Directions

**Input:** $A \in \mathbb{R}^{n \times d}$, and an integer $\ell$

$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$

for ($a_i \in A$)

- Insert $a_i$ into $B$

if ($B$ is full)

$$[U, S, V] \leftarrow \text{svd}(B)$$

$$\tilde{S} \leftarrow \begin{bmatrix} \sqrt{S_1^2 - S_{i/2}^2}, & \sqrt{S_2^2 - S_{i/2}^2}, & \ldots & 0, & \ldots & 0 \end{bmatrix}$$

$B \leftarrow \tilde{S} V^T$

return $B$
Frequent Directions

- Similar to the frequent items case, this method has the following error guarantee:

\[ \| A^T A - B T B \| \ll \frac{2}{l} \| A \|_F^2 \]

- And if using \( l = k + k/\epsilon \)

\[ \| A - \Pi_B(A) \|_F^2 \ll (1 + \epsilon) \| A - A_k \|_F^2 \]
Sketching in Experiment

$\text{cov-err} := \frac{\|A^T A - B^T B\|_F^2}{\|A\|_F^2}$

- Random Projections
  - [Sarlos FOCS06]
- Hashing
  - [Clarkson+Woodruff STOC13]
- Sampling
  - [Drineas, Kannan, Mahoney SIAM JoC06]
- FrequentDirections
  - [all 0s]
- Naive
  - [SVD]
- Brute Force

Sketch Size

Covariance Error
Sketching in Experiment

Projection Error vs. Sketch Size

- Random Projections
- Hashing
- Sampling
- FrequentDirections
- Naive
- Brute Force

Projection Error: \( \frac{\|A - \pi_B(A)\|_2^2}{\|A - A_k\|_F^2} \), \( k = 10 \)

- [Sarlos FOCS06]
- [Clarkson+Woodruff STOC13]
- [Drineas, Kannan, Mahoney SIAM JoC06]
- [all 0s]
- [SVD]
Matrix Sketching in Streams:

- Row sampling methods
  - CUR
  - L2 norm based sampling
- Random projection methods
  - Johnson Lindenstrauss Transform (JLT)
  - Different ways to construct a JLT matrix
- Iterative sketching methods
  - Misra-Gries algorithm for frequent items
  - Frequent Directions method (state of the art)