1 Projection Algorithm

Johnson-Lindenstrauss lemma (JL) addresses how well a metric can be embedded in \( L^2 \). If the metric is Euclidean, it can be done with an \( \epsilon \)-distortion for every \( \epsilon \). For other metrics, it is usually worse: maybe with a constant or logarithmic distortion.

**Question:** Which metric is the least like \( L^2 \)? An expander.

Papadimitriou, Raghavan, Tamaki, and Vempala (PRTV) Algorithm
LSI is a way to represent documents in a vector space. From last time, we know that for both SVD and PCA, if you keep a few directions, you can capture most of the norm.

**Recall:** Frobenius Norm: 
\[
\|\Omega\|_F = \text{tr}(\Omega^T \Omega) = \sum_{i,j} |\Omega_{ij}|^2 = \sum_i \sigma_i^2 \quad \text{and} \quad \|\Omega\| = \min_x \frac{\|\Omega x\|_2}{\|x\|_2}
\]

Random Projection Algorithm

**INPUT:** \( A \in \mathbb{R}^{m \times n} \)

1) \( R \in \mathbb{R}^{m \times l} \) where \( R \) is a random matrix whose entries are distributed as \( R_{ij} \sim N(0,1) \) and \( l \geq \frac{\log(n)}{\epsilon^2} \)

2) Construct \( B = \frac{1}{\sqrt{l}} R^T A \in \mathbb{R}^{l \times n} \)

3) Compute the SVD of \( B \), \( B = \sum_{i=1}^l \lambda_i a_i b_i^T \), where \( b_i \) are the right singular vectors and \( a_i \) are the left singular vectors.

4) Return \( B_k \) or \( \tilde{A} = A \sum_{i=1}^k b_i b_i^T \)

In the last step, we either return \( B_k \), the basis or \( \tilde{A} \) the actual projection. The runtime of step 1 is \( O(ml) \), step 2 is \( O(mnl) \), step 3 is \( O(nl^2) \) and the total runtime is \( O\left( \frac{mn \log(n)}{\epsilon^2} \right) \).

**Claim 1.1.** \( ||A - \tilde{A}||_F \leq ||A - A_k||_F^2 + \epsilon ||A_k||_F^2 \) with probability \( 1 - 4n^{-\frac{(\epsilon^2-\epsilon^3)}{4}} \).
Proof Recall \( A = \sum_i \sigma_i u_i v_i^T \), \( B = \sum_i \lambda_i a_i b_i^T \), and \( \tilde{A} = A \sum_i b_i b_i^T \)

\[
||A - \tilde{A}||_F^2 = \sum_{i=1}^n ||(A - \tilde{A}) b_i||_2^2 = \sum_i ||Ab_i - \tilde{Ab}_i||_2^2 \\
= \sum_i \left|\left| Ab_i - A \left( \sum_{j=1}^k b_j b_j^T \right) b_i \right|\right|^2 = \sum_{i=k+1}^n ||Ab_i||_2^2 \\
= ||A||_F^2 - \sum_{i=1}^k ||Ab_i||_2^2 = ||A - A_k||_F^2 + ||A_k||_F^2 - \sum_{i=1}^k ||Ab_i||_2^2
\]

We want to relate \( \sum_{i=1}^k ||Ab_i||_2^2 \) to the singular values of \( B \).

\[
\sum_{i=1}^k \lambda_i^2 = \sum_{i=1}^k ||Bb_i||_2^2 = \sum_{i=1}^k \frac{1}{k} ||R^T Ab_i||_2^2 \\
= \left( 1 + \frac{\epsilon}{k} \right) \sum_{i=1}^k ||Ab_i||_2^2 \quad \text{by JS}
\]

Now, \( \sum_{i=1}^k ||Ab_i||_2^2 \leq \frac{1}{1 + \frac{\epsilon}{2}} \sum_{i=1}^k \lambda_i^2 \)

Since \( v_i \) are the basis vectors for \( A \),

\[
\sum_{i=1}^k \lambda_i^2 \geq \sum_i v_i^T B^T B v_i = \sum_{i=1}^k \frac{1}{k} v_i^T A^T R R^T A v_i \\
= \sum_{i=1}^k \frac{1}{k} ||R^T A v_i||_2^2 \geq \left( 1 - \frac{\epsilon}{k} \right) \sum_{i=1}^k ||Av_i||_2^2 \\
= \left( 1 - \frac{\epsilon}{k} \right) ||A_k||
\]

Combining gives

\[
\sum_{i=1}^k ||Ab_i||_2^2 \geq \frac{1 - \frac{\epsilon}{k}}{1 + \frac{\epsilon}{2}} ||A_k||_F^2 \geq (1 - \epsilon) ||A_k||^2
\]

\[\square\]

2 Approximating Matrix Products and Random Sampling for Low Rank Approximation

**Streaming model**: assume that we can only take passes on the data, no random access is allowed. The resources required for a streaming algorithm are thus the number of passes, the additional time, and additional space required.

**Concept**: take a pass over the data, keep a sample, and return output based on that sample. If all data is similar, uniform random sampling should be sufficient. If not, we should change our probability distribution to be non-uniform. Also, we should analyze how much worse this technique will be compared to operating on all of our data.

**SELECT algorithm**

1) Set \( D = 0 \)
2) while (more data exists in the stream)
3) read item \( \{i, a_i\} \)
4) Set \( D = D + a_i \)
5) with probability \( \frac{a_i}{D} \), set \( i^* = i \) and \( a^* = a_i \)
5) output \( i^* \) and \( a^* \)

**Lemma:** in one pass and \( O(1) \) space and time, SELECT returns \( i^* \) and \( a^* \) such that \( P[i = i^*] = \frac{a^*}{\sum_{j=1}^{n} a_j} \)
(proof by induction).

**Lemma:** If you run SELECT on \( (i, j, A_{ij}) \), then \( P(\{i, j\} = \{i^*, j^*\}) = \frac{A_{i^*j^*}}{||A||_F} \)

**Basic Matrix Multiplication Algorithm:**
Inputs: \( A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}, c \geq 0, \) probability distribution \( P_i \) defined for \( i = 1 \rightarrow n \)
Output: \( C \), an \( m \times c \) matrix of the sampled columns of \( A \), and \( R \), a \( c \times n \) matrix of the sampled rows of \( B \).

1) for \( t = 1 \rightarrow c \\
2) \text{pick } i_t \in [n] \text{ with } P(i_t = k) = P_k \\
4) \quad C^{(t)} = \frac{A^{(i_t)}}{\sqrt{C \cdot P_{i_t}}} \\
5) \quad R_t = \frac{B(i_t)}{\sqrt{C \cdot P_{i_t}}} \\
5) \text{return } C, R \\

We thus have \( CR = \sum_{t=1}^{c} C^{(t)} R_t = \sum_{t=1}^{c} \frac{1}{\sqrt{C \cdot P_{i_t}}} A^{(i_t)} B_{(i_t)} \approx \sum_{t=1}^{n} A^{(t)} P_t \)