CS 15-859: Algorithms for Big Data Fall 2020

Lecture #3.2 — September 23, 2021

Prof. David Woodruff *Scribe: Yucheng Dai*

1 Low Rank Approximation

Now we want to compute SVD quickly for a special type of matrices.

For example, let A be a $(n \times d)$-matrix where $A_{i,j}$ represents how many times customer i bought item j. Since we might only focus on a few major purchase patterns (i.e. trend), we need to find the first k principal components of the SVD of the matrix.
However, each customer can be different, which means this matrix could have high rank because of noise. In this case, it costs too much to do SVD directly, so we need a new way.

Mathematically, we would like to find $A \approx L \circ R \circ X$ where L is a $(n \times k)$-matrix and R is a $(k \times d)$-matrix, and $(n+d)k << nd$.

What is a good low rank approximation?
Consider the Singular Value Decomposition(SVD):
Any matrix $A = U \cdot \Sigma \cdot V^T$.

1. U has orthonormal columns.
2. Σ is diagonal with non-increasing positive entries down the diagonal.
3. V has orthonormal columns.

Let U_k consist of the previous k columns of U, and so do Σ_k and V_k, then we can write:

$$A = U_k \Sigma_k V_k^T + E = A_k + E$$

where A_k, in fact, is the best rank-k approximation, i.e. $A_k = \arg\min_{\text{rank}(B)=k} |A - B|_F$.

However, computing A_k exactly is still expensive. Hence, we consider output a rank-k matrix A' so that:

$$|A - A'|_F \leqslant (1+\epsilon)|A - A_k|_F$$

We claim that we can find A' in $\text{nnz}(A) + (n+d) \cdot \text{poly}(k/\epsilon)$ time.
(Recall: $\text{nnz}(A)$ is the number of non-zero entries of A)

Thus, for dense matrix A, this is almost $O(nd)$, which is still better than SVD $[O(\min(n^2 d, nd^2))]$.
For sparse matrix A, this can be $O(n+d)$.

Hence, we would like to reduce the big SVD on $(n \times d)$-matrix to a smaller SVD, so we'll construct matrix SA with $k/\epsilon << n$ rows.
Then, we project rows of A onto SA, and find best rank-k approximation inside of SA.

We can choose the matrix S using the Fast Johnson Lindenstrauss Matrix ($S = PHD$), etc. But we'll use CountSketch here, so that SA can be computed in $\text{nnz}(A)$ time.

But why do these matrices work?
Consider the hypothetical regression problem: $\min_X |A_k X - A|_F$.
Obviously, $X =$ the identity is the best answer, but we're not solving it here.
Let S be an affine embedding, then $|SA_k X - SA|_F = (1 \pm \epsilon)|A_k X - A|_F$ for all X.
By normal equation(See Lec.#1), $\arg\min_X |SA_k X - SA|_F = (SA_k)^- SA$.
Hence, $|A_k(SA_k)^- SA - SA|_F \leqslant (1+\epsilon)|A_k - A|_F$.
Now note that $(A_k(SA_k)^-)\,SA$ is in the row span of SA, so we proved that there must be an answer in the row space of SA.

Therefore, we can start constructing the algorithm:

$$\min_{\text{rank-k } X} |XSA - A|_F^2 \leqslant |A_k(SA_k)^- SA - A|_F^2 \leqslant (1+\epsilon)|A - A_k|_F^2$$

By the normal equation,

$$|XSA - A|_F^2 = |XSA - A(SA)^- SA|_F^2 + |A(SA)^- SA - A|_F^2$$

Hence,

$$\min_{\text{rank-k } X} |XSA - A|_F^2 = |A(SA)^- SA - A|_F^2 + \min_{\text{rank-k } X} |XSA - A(SA)^- SA|_F^2$$

We write $SA = U\Sigma V^T$ in its **thin** SVD.
This works since SA is of dimension $poly(k/\epsilon) \times d$, which is small.

Then,

$$\begin{aligned}
&\min_{\text{rank-k } X} |XSA - A(SA)^- SA|_F^2 \\
=\ &\min_{\text{rank-k } X} |XU\Sigma V^T - A(SA)^- U\Sigma V^T|_F^2 \\
=\ &\min_{\text{rank-k } X} |XU\Sigma - A(SA)^- U\Sigma|_F^2 \\
=\ &\min_{\text{rank-k } Y} |Y - A(SA)^- U\Sigma|_F^2
\end{aligned}$$

Where at the last step we substitute $Y = XU\Sigma$. Solving X is equivalent to solving Y.

Note that $\min_{\text{rank-k } Y} |Y - A(SA)^- U\Sigma|_F^2$ is a small rank-k approximation problem so we can solve directly.
Moreover, it is very efficient to calculate $(SA)^- U\Sigma$, but there is no way to multiply it by A, because A has dimension $(n \times d)$, and we don't want any kind of $O(nd)$.

Now the current algorithm structure is like this:

1. Compute SA.
2. Project each row onto SA.
3. Find best $rank - k$ approximation of project points in the row space of SA.

The previous bottleneck is exactly step 2.

Observe that $\min_{\text{rank-k } X}|XSA - A|_F^2 = \min_{\text{rank-k } X}|(SA)^T X^T - A^T|_F^2$.
Note that $(SA)^T$ is of dimension $(k/\epsilon \times d)$, which is tall and thin.
Hence, we can apply affine embedding, so that we compute multiplications related to AR of dimension $n \times \text{poly}(k/\epsilon)$, which is acceptable.

We choose an affine embedding R s.t.

$$|XSAR - AR|_F^2 = (1 \pm \epsilon)|XSA - A|_F^2$$

for all X.

Note we can compute AR and SAR in $\text{nnz}(A)$ time.
Then we just solve $\min_{\text{rank-k } X}|XSAR - AR|_F^2$.

Similarly, by normal equation,

$$\min_{\text{rank-k } X}|XSAR - AR|_F^2 = |AR(SAR)^- SAR - AR|_F^2 + \min_{\text{rank-k } X}|XSAR - AR(SAR)^- SAR|_F^2$$

Substitute $Y = XSAR$, computing $\min_{\text{rank-k } Y}|Y - AR(SAR)^- SAR|_F^2$ using SVD needs $n \cdot \text{poly}(k/\epsilon)$ time. Then, $X = Y(SAR)^-$, and our rank-k approximation $XSA = Y(SAR)^- SA$.
In practice, give XSA in factored form is better, so we don't bother using $O(nd)$ time just to output the matrix.

2 High Precision Regression

Goal: Output x' for which $|Ax' - b|_2 \leqslant (1+\epsilon)\min_x|Ax - b|_2$.
Our old algorithms all run in $\text{poly}(d/\epsilon)$ time.
This becomes unacceptable when we need high precision, i.e. ϵ is very small.

A new algorithm can resolve this by reducing it to $\text{poly}(d) \cdot \log(1/\epsilon)$.
It is easy to deduce from the time complexity that we should start from a constant ϵ_0 and repeat some process. For each step, the precision raises by a factor of ϵ_0. Here's how:

We want to make an well-conditioned:

$$\kappa(A) = \sup_{|x|_2=1}|Ax|_2 / \inf_{|x|_2=1}|Ax|_2$$

The speed of many algorithms including Gradient Descend relies on this constant.
Now we want to use sketching to reduce $\kappa(A)$ to $O(1)$.

Let S be a $(1+\epsilon_0)$ subspace embedding of A.
Compute the QR-factorization $SA = QR^{-1}$.

Note that Q has orthonormal columns, so $\kappa(Q) = 1$.

Claim: $\kappa(SA) \leqslant \frac{1+\epsilon_0}{1-\epsilon_0}$.

Proof. For all units x, $(1-\epsilon_0)|ARx|_2 \leqslant |SARx| = 1$.
For all units x, $(1+\epsilon_0)|ARx|_2 \geqslant |SARx| = 1$.
So $\kappa(SA) \leqslant \frac{1/(1-\epsilon_0)}{1/(1+\epsilon_0)} = \frac{1+\epsilon_0}{1-\epsilon_0}$. ■

Now we let S be a $(1+\epsilon_0)$ embedding for AR.

Solve $x_0 = \arg\min_x |SARx - Sb|_2$.

Time needed to compute R and x_0 is $\text{nnz}(A) + \text{poly}(d)$ for constant ϵ_0.

At step m, we do $x_{m+1} \leftarrow x_m + R^T A^T (b - ARx_m)$. Note that:

$$\begin{aligned}
&AR(x_{m+1} - x^*) \\
=&AR(x_m + R^T A^T (b - ARx_m) - x^*) \\
=&AR(x_m - x^*) - ARR^T A^T ARx_m + ARR^T A^T b \\
=&(AR - ARR^T A^T AR)(x_m - x^*) \\
=&U(\Sigma - \Sigma^3)V^T (x_m - x^*)
\end{aligned}$$

Where $AR = U\Sigma V^T$ is the SVD of AR.

Here in the middle step we used the normal equation:

$$(AR)^T ARx^* = (AR)^T b$$

And in the last step, note $ARR^T A^T AR = AR(AR)^T AR = U\Sigma^3 V^T$.

Hence,

$$\begin{aligned}
&|AR(x_{m+1} - x^*)|_2 \\
=&|U(\Sigma - \Sigma^3)V^T (x_m - x^*)|_2 \\
=&|(\Sigma - \Sigma^3)V^T (x_m - x^*)|_2 \\
=&O(\epsilon_0)|V^T (x_m - x^*)|_2 \\
=&O(\epsilon_0)|(x_m - x^*)|_2 \\
=&O(\epsilon_0)|AR(x_m - x^*)|_2
\end{aligned}$$

In the last step, it is fine to multiply AR back because:

1. Multiplying V^T doesn't matter.
2. To multiply Σ, we need to divide $1 - \epsilon_0$, but $O(\epsilon_0)/(1 - \epsilon_0) = O(\epsilon_0)$.
3. Multiplying U doesn't matter.

Therefore,

$$\begin{aligned}
&|ARx_m - b|_2^2 \\
=&|AR(x_m - x^*)|_2^2 + |ARx^* - b|_2^2 \\
=&O(\epsilon_0^{2m})|AR(x_0 - x^*)|_2^2 + |ARx^* - b|_2^2 \\
\leqslant&O(\epsilon_0^{2m})|ARx_0 - b|_2^2 + |ARx^* - b|_2^2 \\
\leqslant&(1+\epsilon_0)O(\epsilon_0^{2m})|ARx^* - b|_2^2 + |ARx^* - b|_2^2 \\
=&(1 + O(\epsilon_0^{2m}))|ARx^* - b|_2^2
\end{aligned}$$

Hence, we repeat for $O(\log(1/\epsilon))$ times, we get $1 + O(\epsilon_0^{2m}) \to 1 + \epsilon$.