CS 15-859: Algorithms for Big Data Fall 2020

Lecture 4.1 — October 1

Prof. David Woodruff *Scribe: Lichen Zhang*

1 Recap: Low Rank Approximation via sketching

Recall the *low rank approximation* problem: given $A \in \mathbb{R}^{n\times d}$, find a matrix A_k of rank at most k, such that

$$\|A - A_k\|_F$$

is minimized. For interpretation purpose, one thinks of A is n data points, each in $\mathbb{R}^d$. In last lecture, we have shown that A_k can be found by finding SVD of A, but it takes $O\left(nd^2\right)$ time to compute, so we turn our attention to sketching-based method: consider a CountSketch matrix with $\text{poly}\left(k/\epsilon\right) \ll n$ rows, then we do the following:

- Compute SA, which takes $\text{nnz}\left(A\right)$ time.
- Project rows of A onto SA.
- Find best rank k approximation of projected rows using method such as SVD.

In last lecture, we have shown it is equivalent to solve the following minimization problem:

$$\min_{\text{rank}(Y)\leq k} \|Y - A\left(SA\right)^\dagger U\Sigma\|_F$$

where $SA = U\Sigma V^\top$ is the thin SVD of A, meaning that we remove all 0 singular values from Σ, and remove corresponding columns and rows from U and $V^\top$. One can solve this problem by computing SVD of matrix $A\left(SA\right)^\dagger U\Sigma$, however, to compute the product of A and $\left(SA\right)^\dagger U\Sigma$, it might take up to $\text{nnz}\left(A\right)\cdot\text{poly}\left(k/\epsilon\right)$, which is too expensive. We will introduce an algorithm that takes only $\text{nnz}\left(A\right) + \left(n+d\right)\cdot\text{poly}\left(k/\epsilon\right)$ to solve this problem.

2 Approximate projection via Affine Embedding

The computation bottleneck of our proposed scheme is to project rows of A onto SA, notice the projection problem is just regression, so the problem reduces to find fast algorithms for approximating regression:

$$\min_{\text{rank}(X)\leq k} \|X\left(SA\right) - A\|_F^2$$

Readers might identify this is exactly the affine embedding problem:

$$\min_X \|AX - B\|_F^2$$

with SA being identified as A, and A being identified as B. So we might just choose an affine embedding R and solve the minimization problem

$$\min_{\text{rank}(X)\leq k} \|X(SA)R - AR\|_F^2 \tag{1}$$

instead. We remind readers that in order for R to be an affine embedding, we require R has $\text{poly}(\text{rank}(SA)/\epsilon)$ columns, and $\text{rank}(SA) \leq \text{poly}(k/\epsilon)$ since it has $\text{poly}(k/\epsilon)$ rows, therefore, R has $\text{poly}(k/\epsilon)$ columns. Since R is an affine embedding, we have

$$\forall X \in \mathbb{R}^{n\times \text{poly}(k/\epsilon)}, \|XSAR - AR\|_F^2 = (1\pm\epsilon)\|XSA - A\|_F^2$$

We know try to analyze the new optimization problem, recall by normal equation, the optimal solution for (1) is

$$X^* = AR(SAR)^\dagger$$

and X^*SAR is the projection of rows of AR onto row span of SAR, so it's orthogonal to $AR - X^*SAR$. By Pythagorean Theorem, for arbitrary X, we have

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

Expand $\|XSAR - X^*SAR\|_F^2$, we get $\|XSAR - AR(SAR)^\dagger SAR\|_F^2$, which is equivalent to $\|Y - AR(SAR)^\dagger SAR\|_F^2$, for some $Y = XSAR$. We have reduced the problem to

$$\min_{\text{rank}(Y)\leq k} \|Y - AR(SAR)^\dagger SAR\|_F^2$$

which can be done via computing SVD for matrix $AR(SAR)^\dagger SAR$. We claim Y necessarily lies on the row span of SAR, otherwise extra cost will be incurred due to Pythagorean Theorem. Therefore, $Y = XSAR$ for some X. Recall our goal is to compute XSA, so we can first compute $X = Y(SAR)^\dagger$, then $XSA = Y(SAR)^\dagger SA$.

We know consider the run time of above algorithm. We can break it into following steps:

- Compute $AR, SAR, (SAR)^\dagger$.
- Compute $AR(SAR)^\dagger SAR$.
- Compute thin SVD of $AR(SAR)^\dagger SAR$, find Y.
- Compute $Y(SAR)^\dagger SA$.

Notice since R is a CountSketch matrix, compute AR takes only $\text{nnz}(A)$ time and compute SAR takes $\text{nnz}(SA) \leq d\cdot\text{poly}(k/\epsilon)$ time. In order to compute $(SAR)^\dagger$, we can compute its SVD and multiply respective parts, which takes $\text{poly}(k/\epsilon)$ time since $SAR \in \mathbb{R}^{\text{poly}(k/\epsilon)\times\text{poly}(k/\epsilon)}$. To compute $AR(SAR)^\dagger SAR$, it takes at most $n\cdot\text{poly}(k/\epsilon)$ time, and compute its SVD takes the same amount of time. Finally, we output $Y(SAR)^\dagger SA$ in factored form, and we are done. The total running time is $\text{nnz}(A) + d\cdot\text{poly}(k/\epsilon) + n\cdot\text{poly}(k/\epsilon) = \text{nnz}(A) + (n+d)\cdot\text{poly}(k/\epsilon)$.

We give a summarization of above algorithm:

- Compute SA.
- Compute SAR and AR.
- Compute $\min_{\text{rank}(Y)\leq k} \|Y - AR\,(SAR)^{\dagger}\,SAR\|_F^2$ using SVD.
- Output $Y\,(SAR)^{\dagger}\,SA$ in factored form.

Overall run time is $\text{nnz}\,(A) + (n+d)\cdot \text{poly}\,(k/\epsilon)$.

3 High Precision Regression

Recall the approximate regression problem: given $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$, output x' such that

$$\|Ax' - b\|_2 \leq (1 \pm \epsilon)\min_x \|Ax - b\|_2$$

with high probability. Use sketch-and-solve paradigm and choose CountSketch matrix as subspace embedding, we achieve a run time of $\text{nnz}\,(A) + \text{poly}\,(d/\epsilon)$. However, if ϵ is very small, the $1/\epsilon$ can be unacceptably large. Our goal is to get an algorithm with run time $O\,(\text{nnz}\,(A) + \text{poly}\,(d))\log\,(1/\epsilon)$.

To do so, we introduce a gradient-based algorithm, which relies heavily on the following definition of a matrix:

Definition. Let $A \in \mathbb{R}^{n\times d}$, the *condition number* of A, denoted by $\kappa\,(A)$, is defined as

$$\kappa\,(A) = \frac{\sup_{\|x\|_2=1} \|Ax\|_2}{\inf_{\|x\|_2=1} \|Ax\|_2} = \frac{\sigma_{\max}\,(A)}{\sigma_{\min}\,(A)}$$

The run time of our algorithm will depend on $\kappa\,(A)$, a large condition number implies a large run time. We will use sketching to reduce $\kappa\,(A)$ to $O\,(1)$.

Pick $\epsilon_0 < 1$ be some constant, let S be a $(1+\epsilon_0)$ subspace embedding for A. We will compute a QR factorization of SA so that $SA = QR^{-1}$, where Q has orthonormal columns and R is an invertible matrix. One way to compute QR factorization is via thin SVD, $SA = U\Sigma V^{\top}$, where U has orthonormal columns, and $\Sigma V^{\top}$ has inverse $V\Sigma^{-1}$. Notice since it's a thin SVD, Σ does not contain 0 entry on its diagonal, so it does admit an inverse.

We make two remarks: 1). S has $\text{poly}\,(d/\epsilon_0) = \text{poly}\,(d)$ rows, so SA is a $\text{poly}\,(d)\times d$ matrix. 2). $\kappa\,(SAR) = 1$, to see it, notice $SAR = Q$, and Q has orthonormal columns, and all singular values of such a matrix are 1. This means by right multiplying R, we squeeze the condition number of SA to the smallest possible value. This motivates the following claim:

Claim 1.

$$\kappa\,(AR) \leq \frac{1+\epsilon_0}{1-\epsilon_0}$$

Proof. Since S is a subspace embedding, we have the following facts:

- For all unit x,

$$\begin{aligned}(1-\epsilon_0)\|ARx\|_2 &\leq \|SARx\|_2 \\ &= \|Qx\|_2 \\ &= \|x\|_2 \\ &= 1\end{aligned}$$

- Similarly, for all unit x,

$$(1+\epsilon_0)\|ARx\|_2 \geq \|SARx\|_2 = 1$$

We hence obtain the following inequality:

$$\frac{1}{1+\epsilon_0} \leq \|ARx\|_2 \leq \frac{1}{1-\epsilon_0}$$

Thus, the condition number of AR is

$$\kappa(AR) = \frac{\sup\limits_{\|x\|_2=1}\|ARx\|_2}{\inf\limits_{\|x\|_2=1}\|ARx\|_2} = \frac{\frac{1}{1-\epsilon_0}}{\frac{1}{1+\epsilon_0}} = \frac{1+\epsilon_0}{1-\epsilon_0}$$ ■

By above claim, we have shown R is a good *preconditioner* for A, in the sense that it squeezes the condition number of AR to a much smaller constant.

With S in hand, a natural thing to do here is to compute

$$x_0 = \underset{x}{\operatorname{argmin}}\|SARx - Sb\|_2$$

Notice x_0 gives a $1 \pm O(\epsilon_0)$ approximation to $\|ARx^* - b\|_2$ and subsequently, Rx_0 gives a $1 \pm O(\epsilon_0)$ approximation to $\|Ax^* - b\|_2$. Starting from x_0, considering the following iterative scheme:

$$x_{m+1} \leftarrow x_m + R^\top A^\top (b - ARx_m)$$

We will show $\|AR(x_m - x^*)\|_2$ decays at a rate of $O(\epsilon_0)$. Combining this fact with the initial gap $\|AR(x_0 - x^*)\|_2$, we will ultimately show that $\log(1/\epsilon)$ suffices to get desired error bound. We shall first examine the vector $AR(x_{m+1} - x^*)$, let $U\Sigma V^\top = AR$ denote the SVD of AR:

$$\begin{aligned}AR(x_{m+1} - x^*) &= AR\left(x_m + R^\top A^\top (b - ARx_m) - x^*\right) \\ &= ARx_m - ARx^* + ARR^\top A^\top b - ARR^\top A^\top ARx_m \\ &= AR(x_m - x^*) + ARR^\top A^\top ARx^* - ARR^\top A^\top ARx_m \quad \text{since } x^* \text{ satisfies normal equation} \\ &= \left(AR - ARR^\top A^\top AR\right)(x_m - x^*) \\ &= \left(U\Sigma V^\top - U\Sigma V^\top V\Sigma U^\top U\Sigma V^\top\right)(x_m - x^*) \\ &= \left(U\Sigma V^\top - U\Sigma^3 V^\top\right)(x_m - x^*) \\ &= U\left(\Sigma - \Sigma^3\right)V^\top (x_m - x^*)\end{aligned}$$

What are singular values of AR? Recall in 1, we in fact show that all singular values of AR lie in $1 \pm O(\epsilon_0)$, and $(1 \pm O(\epsilon_0))^3 = 1 \pm O(\epsilon_0)$, this means each entry of $\Sigma - \Sigma^3$ is a small value in the range $\pm O(\epsilon_0)$. Now we are ready to present the ℓ_2 norm of this vector:

$$\begin{aligned}
\|AR(x_{m+1} - x^*)\|_2 &= \|U\left(\Sigma - \Sigma^3\right)V^\top (x_m - x^*)\|_2 \\
&= \|\left(\Sigma - \Sigma^3\right)V^\top (x_m - x^*)\|_2 \\
&\leq O(\epsilon_0)\|V^\top (x_m - x^*)\|_2 && \text{diagonal matrix scales entry of vector} \\
&\leq \frac{O(\epsilon_0)}{1 - O(\epsilon_0)}\|\Sigma V^\top (x_m - x^*)\|_2 \\
&\leq (1 + O(\epsilon_0))\, O(\epsilon_0)\|\Sigma V^\top (x_m - x^*)\|_2 \\
&= \left(O(\epsilon_0)^2 + O(\epsilon_0)\right)\|U\Sigma V^\top (x_m - x^*)\|_2 \\
&= O(\epsilon_0)\|AR(x_m - x^*)\|_2 \\
&\leq \frac{1}{2}\|AR(x_m - x^*)\|_2 && \text{pick } \epsilon_0 \text{ such that the factor is less than } 1/2
\end{aligned}$$

We get a decay rate of $O(\epsilon_0) \leq \frac{1}{2}$, by a telescoping argument, we have

$$\|AR(x_m - x^*)\|_2 \leq O(\epsilon_0)^m \|AR(x_0 - x^*)\|_2$$

Pick $m = O(\log(1/\epsilon))$, the discrepancy becomes

$$\|AR(x_m - x^*)\|_2 \leq O(\epsilon)\|AR(x_0 - x^*)\|_2$$

Finally, let's compute $\|ARx_m - b\|_2^2$:

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

This shows our iterative scheme finds a $1 \pm O(\epsilon)$ approximation to the optimal solution. We now analyze the run time of the algorithm:

- Compute SA, picking S as a CountSketch matrix, it takes $\text{nnz}(A)$ time.
- Compute R, which can be done by SVD on SA, $\text{poly}(d)$ time. Notice R is a $d \times d$ matrix.
- Compute SAR, it takes at most $\text{poly}(d)$ time.
- Compute x_0, one can simply solve the normal equation, which takes $\text{poly}(d)$ time.
- Iterative compute $x_{m+1} \leftarrow x_m + R^\top A^\top (b - ARx_m)$, we have upper bounded the number of iterations by $O(\log(1/\epsilon))$, so it suffices to analyze the run time of a single update: the idea is to perform matrix-vector product from Rx_m, then $A(Rx_m)$, so on and so forth, so each matrix-vector product takes either $\text{nnz}(A)$ or $\text{poly}(d)$ time, and two vector subtraction takes $n \leq \text{nnz}(A)$ time. In total, this process takes $O(\text{nnz}(A) + \text{poly}(d))\log(1/\epsilon)$ time.

Thus, our algorithm takes $O\left(\mathrm{nnz}\left(A\right)+\mathrm{poly}\left(d\right)\right)\log\left(1/\epsilon\right)$ in total, as desired.

A Appendix: An alternative perspective on algorithm for High Precision Regression

In this appendix, we introduce an interpretation of algorithm in high precision regression section, which motivates the design of algorithm. The idea is to interpret the objective function in minimization problem as a *cost function*:

$$F\left(x\right)=\frac{1}{2}\|Ax-b\|_2^2$$

So the problem reduces to minimize the cost function. We remark that F is in fact a convex function, so a standard algorithm is to run *gradient-descent* to optimize F. The minimum solution satisfies the relation that the first derivative is 0:

$$\frac{dF}{dx}=\frac{1}{2}\frac{d\left(x^\top A^\top Ax+b^\top b-x^\top A^\top b\right)}{dx}=A^\top Ax-A^\top b=0$$

This gives the normal equation, and yields an alternative proof of optimality of normal equation. Instead solve the normal equation directly, we iteratively improve the quality of x by moving in the negative direction of the gradient:

$$x_{m+1}=x_m-\gamma\frac{dF}{dx}\left(x_m\right)$$

where γ is the step size of descent. Plug in the expression of $\frac{dF}{dx}$, we get

$$x_{m+1}=x_m-\gamma A^\top\left(Ax-b\right)$$

A quick fact: if γ is sufficiently small, then this algorithm converges at a rate of $1-1/\kappa\left(A^\top A\right)$. Notice this makes the algorithm almost impractical to run: since $\left(1-1/\kappa\left(A^\top A\right)\right)^{\kappa\left(A^\top A\right)}\approx 1/e$, it requires around $\kappa\left(A^\top A\right)=\kappa\left(A\right)^2$ iterations to improve the solution by a constant factor, if $\kappa\left(A\right)$ is relatively large, then the scheme will take too long to converge. This is where we bring in the preconditioner R: the map AR has a condition number of size only a constant, this means the quality of solution improves by a constant fraction, after only a constant number of iterations. Moreover, by using sketching, we find a good initial solution, which is only a constant factor away from the optimal solution, this further reduces the number of iterations required.

To summarize, our sketching-based high precision regression algorithm can be viewed as a preconditioned gradient-descent algorithm. One big win by using sketching, especially CountSketch, is that we can efficiently find a high quality preconditioner and compute a good initial point. It also allows us to step size $\gamma=1$, which significantly improves run time.