CS7880: Faster Regression via Gradient Descent and Subspace Embedding

Scribe: Niklas Smedemark-Margulies

March 14, 2019

Contents

1 Last time

1.1 Least Squares Regression

Recall the Least Squares Regression problem. We have a data matrix $A \in \mathbb{R}^{n \cdot d}$, containing our n data examples, each with d features. We want to find a vector x satisfying:

$$\min_x \|Ax - b\|_2^2 \tag{1}$$

Note - the typical regime is with $n >> d$; if $d > n$, the system is under-determined, and we would need to apply additional constraints to find a unique solution.

1.2 Analytical Solution by Projection

The vector of labels b may or may not be in the column space of A. In either case, our objective is to find a vector x whose product with A has the minimum euclidean distance. This is exactly the projection of b onto the column space of A. Notice if b does lie in the column space, we can find a vector with 0 error.

We can derive the analytical solution by simply considering our error vector. We observe that the error vector, $Ax^* - b$ is perpendicular to the column space of A. It must have a 0 dot product with each column of A, or in other words, this error vector is in the **left nullspace** of the matrix A:

$$\begin{aligned} A^T(Ax^* - b) &= 0 \\ A^TAx^* &= A^Tb \\ x^* &= (A^TA)^{-1}A^Tb \end{aligned}$$

Recall that the running time of this analytical solution is dominated by the multiplication A^TA, because $n >> d$.

1.3 Faster Solution using Subspace Embedding

Last time, we saw that we can compute an approximation to the least squares solution by projecting the whole problem to a lower-dimensional space using a sketching matrix Π.

Intuitively, if we perform this projection using a matrix that satisfies the ε-subspace embedding property for the column space of A, which nearly preserves lengths for all the vectors in this subspace, we will also find a good approximation to the true solution.

For a sketching matrix $\Pi \in \mathbb{R}^{m \cdot n}$, which projects vectors from its rowspace in $\mathbb{R}^n$ into its lower-dimensional column space in $\mathbb{R}^m$, we search for the approximate solution $\tilde{x}$ as follows:

$$\min_x \|\Pi Ax - \Pi b\|_2^2 \tag{2}$$

We discussed two sketching matrix constructions, with different runtimes. Here, $nnz(A)$ represents the number of non-zero entries of A. In general, A might be very sparse, and could be represented in the memory of a program as just a set of non-zero locations; therefore, we try to be a bit careful about the time required to multiply by A.

- Using a dense Gaussian matrix, we can get away with few rows ($m = \Theta(\frac{d}{\varepsilon})$, but multiplication by this dense matrix is relatively slower. We have an overall runtime of $O(\frac{d}{e} \cdot nnz(A) + poly(d, 1/\varepsilon)$
- Using a matrix containing only a single non-zero entry per column, each of which is just $+/-1$, we needed $m = \Theta(\frac{d^2}{\varepsilon^2 \delta})$ rows. Recall that:
  - δ was the failure probability that this matrix actually fails to preserve lengths, d is our number of features, and ε is the distortion factor.
  - Because of the special way we represented this matrix using hash functions, we were also able to multiply with it quickly, and our overall least squares runtime was $O(nnz(A) + poly(d, 1/\varepsilon))$. Notice this runtime has a larger polynomial in d and $1/\varepsilon$ than for the dense Gaussian case.

2 Faster Approaches

By making the observation that we are optimizing a convex objective, we can also try using other convex optimization approaches to solve this problem faster. In particular, let's try to use gradient descent.

2.1 Condition Number for Inversion

First, let's define the **condition number for inversion** of a matrix. We will state for the purpose of intuition (deferring a strict analysis of this fact for elsewhere) that having control of the condition number of a matrix allows us to also set a favorable step size for gradient descent, and therefore perform gradient descent faster.

Def: For a matrix A, the condition number for inversion is the ratio between the maximum and minimum singular value of A.

As discussed, we really want to find the projection of b onto the column space of A. We This allows us to find the same solution by considering any other matrix with the same column space - in particular, we will consider a matrix with a smaller condition number.

- Let $\Pi \in \mathbb{R}^{m \cdot n}$ be a $\frac{1}{4}$-subspace embedding for the column space of $A \in \mathbb{R}^{n \cdot d}$.
- Let $\Pi A = U \Sigma V^T$ be the singular value decomposition of the matrix ΠA.
- Let $R = V \Sigma^{-1}$ be our **preconditioner** matrix. This matrix will help us produce a more **well-conditioned** matrix while maintaining the same column space

Note that we can compute an approximation to the SVD in time $O(md^2)$. In general, we cannot obtain an exact form of the svd, even for a matrix with all rational or even all integer entries. We defer discussion of the precision issues associated with this approximation of the SVD, though it is possible to carry this error term through the same analysis.

For convenience later, note several other facts:

- For any matrix M, and any invertible matrix C, the product MC has the same column space as M.
- Multiplying any vector x by any orthonormal matrix Q (satisfying $Q^TQ = I$) preserves the length of x: $\|Qx\|^2 = (Qx)^T(Qx) = x^TQ^TQx = x^Tx = \|x\|^2$.

In particular, these facts imply that when we consider the SVD of a matrix $A = U\Sigma V^T$, where we know that the left and right singular vector matrices are orthonormal ($U^TU = I$, $V^TV = I$), and the singular values matrix is invertible, then we have:

$$\begin{aligned} A &= U\Sigma V^T \\ AV\Sigma^{-1} &= U \\ A\,(\cdot) &= U \end{aligned}$$

The column space of U and A are the same.

We resume with our analysis of the effect of this preconditioner matrix R. Consider how it affects the length of vectors, and apply our definition for the subspace embedding matrix Π:

$$\begin{aligned} \|ARx\| &= (1 \pm \frac{1}{4})\|\Pi ARx\| \\ &= (1 \pm \frac{1}{4}\|(U\Sigma V^T)(V\Sigma^{-1})x\| \\ &= (1 \pm \frac{1}{4})\|x\| \end{aligned}$$

From this, we see that by preconditioning, we have a matrix AR with the same column space as A (because it is constructed by right-multiplying A with two invertible matrices), and with all singular values in the range $1 \pm \frac{1}{4}$ (such that we will be able to perform fast gradient descent). We can also say that this matrix AR is well-conditioned.

2.2 Gradient Descent

We will now consider performing our least squares optimization problem relative to this well conditioned matrix $\tilde{A} = AR$.

We can run the following iterative gradient descent algorithm:

1. Pick $x^{(0)}$ such that $\|\tilde{A}x^{(0)} - b\| \leq 1.1\|\tilde{A}x^* - b\|$. We can compute this $x^{(0)}$ using a $\Theta(1)$-subspace embedding matrix and solving the system analytically.
2. set $x^{(t+1)} \leftarrow x^{(t)} + \tilde{A}^T(b - \tilde{A}x)$. The second term is the negative of the gradient.

Notice that this looks like a typical gradient descent algorithm, though we are using a surprisingly large step size of 1. Recall that we know that $\tilde{A}$ is well-conditioned. When we move our current solution vector $x^{(t)}$ in the direction of the gradient, we want to think about how much the components of this vector along each singular vectors are modified. We will try not to step too far, since then our solution vector will get distorted. Intuitively, we can think of our solution vector in terms of its components along the various singular vectors of $\tilde{A}$:

- For an ill-conditioned matrix, with a large ratio of singular values, the components along one singular vector and another may become relatively very skewed, causing our solution vector to deviate.
- For a well-conditioned matrix like $\tilde{A}$, we know that the relative change in length of components along various singular vectors will not be skewed in this way (because the ratio of stretching is close to 1!). Therefore, we can take a large step along the direction of interest, and know that the components of our solution vector will not become relatively distorted.

Now, we will show that the error $\|\tilde{A}(x^{(t)} - x^*)\|$ goes down exponentially in the number of iterations t.

Recall that $(A^TA)x^* = A^Tb$, and likewise $(\tilde{A}^T\tilde{A})x^* = \tilde{A}^Tb$, which we will substitute below.

Consider the change in our error after a single iteration, applying our iterative update rule:

$$\begin{aligned}
\|\tilde{A}\left(x^{(t+1)} - x^*\right)\| &= \|\tilde{A}\left(x^{(t)} + \tilde{A}^T(b - \tilde{A}x^{(t)}) - x^*\right)\| \\
&= \|\tilde{A}\left(x^{(t)} + (\tilde{A}^T\tilde{A})x^* - (\tilde{A}^T\tilde{A})x^{(t)} - x^{(t)}\right)\| \\
&= \|\tilde{A}\left(I - (\tilde{A}^T\tilde{A})\right)\left(x^{(t)} - x^*\right)\|
\end{aligned}$$

Notice that we are multiplying by a matrix $(I - (\tilde{A}^T\tilde{A}))$; since we know the singular values of $\tilde{A}$ are all close to one, we know that this results in a matrix with all singular values close to 0!

Let $\tilde{A} = U'\Sigma'V'^T$ be the singular value decomposition of $\tilde{A}$. Σ' is diagonal, with all entries in the range $(1 \pm \frac{1}{4})$

Continuing with the same analysis. Notice here that the product of two diagonal matrices is commutative; $A \cdot B = B \cdot A$. Towards the end, we also use the triangle inequality,

$\|A \cdot B\| \leq \|A\| \cdot \|B\|$.

$$\begin{aligned}
\|\tilde{A}\left(x^{(t+1)} - x^*\right)\| &= \|U'\Sigma'V'^T\left(I - V'\Sigma'U'^TU'\Sigma'V'^T\right)\left(x^{(t)} - x^*\right)\| \\
&= \|\Sigma'V'^T\left(I - V'\Sigma'^2V'^T\right)\left(x^{(t)} - x^*\right)\| \\
&= \|\Sigma'\left(I - \Sigma'^2\right)V'^T\left(x^{(t)} - x^*\right)\| \\
&= \|\left(I - \Sigma'^2\right)\Sigma'V'^T\left(x^{(t)} - x^*\right)\| \\
&\leq \|I - \Sigma'^2\| \cdot \|\Sigma'V'^T\left(x^{(t)} - x^*\right)\| \\
&\leq \frac{9}{16} \cdot \|\Sigma'V'^T\left(x^{(t)} - x^*\right)\|
\end{aligned}$$

We see that our error decreases by a constant factor in each iteration, and therefore decreases exponentially in the number of iterations.

The total running time to achieve an error of ε is therefore $O(\log(\frac{1}{\varepsilon}) \cdot (nnz(\tilde{A} + d^3)$, which is the number of iterations required to reach a target error, times the time to compute a single iteration.

2.3 Sketching Matrix with Subspace Embedding and AMPP

Another approach we can take to quickly solving the least-squares regression problem is to find a matrix Π that is simultaneously a 0.01-subspace embedding matrix for the column space of A, and also satisfies the $\sqrt{\frac{\varepsilon}{d}}$-approximate matrix product property.

Recall that the this matrix product property states:

$$Pr_\Pi\left[\|(\Pi A)^T(\Pi B) - A^TB\| < \sqrt{\frac{\varepsilon}{d}}\|A\|_F\|B\|_F\right] > \frac{9}{10}$$

We can build such a matrix using the sparse construction (one non-zero per column), with additional rows to satisfy the approximate matrix product property. The total number of rows will then bounded by the sum of requirements for this combined construction will then be $O(d^2 + d/\varepsilon)$ rows.

With our original matrix A, we have a true optimal solution vector of x^* and an optimal solution vector for the projected problem of $\tilde{x}^*$. For convenience, define the following:

$$\begin{aligned}
A &= U\Sigma V^T \\
Ax^* &= U\alpha, \text{ for some vector } \alpha \\
Ax^* - b &= -w, \text{ the optimal error vector} \\
A\left(\tilde{x}^* - x^*\right) &= U\beta, \text{ for some vector } \beta
\end{aligned}$$

The error for our projected solution vector will be:

$$\|A\tilde{x}^* - b\|_2^2 = \|A\left(\tilde{x}^* - x^*\right) + Ax^* - b\|_2^2$$

Notice that on the left hand side, we are looking for the length of a vector. This vector is most likely NOT perpendicular to the column space of A. Instead, the optimal error vector $Ax^* - b$ is perpendicular to the column space, and forms one leg of our right triangle. The final side of the right triangle is the difference vector between the true optimal solution and our subspace-projected optimal solution (i.e. $A(\tilde{x}^* - x^*)$).

Therefore, we can use Pythagoras' theorem, and apply the convenience substitutions we defined just above.

$$\begin{aligned}\|A\tilde{x}^* - b\|_2^2 &= \|A\left(\tilde{x}^* - x^*\right)\|_2^2 + \|Ax^* - b\|_2^2 \\ &= \|\beta\|_2^2 + OPT^2\end{aligned}$$

Also using our substitutions, we can state the following. Recall that the column space of U (the left singular vector matrix) is the same as the column space of A, as proven above. Finally, note that for a vector in some space, its projection into that same space is unchanged. We can also express this by saying that a projection matrix is idempotent; $P^2 = P$.

$$\begin{aligned}\Pi U(\alpha + \beta) &= \Pi A\tilde{x}^* \\ &= Proj_{span(\Pi A)}(\Pi b) \\ &= Proj_{span(\Pi U)}(\Pi b) \\ &= Proj_{span(\Pi U)}(\Pi\left(U\alpha + w\right) \\ &= \Pi U\alpha + Proj_{span(\Pi U)}(\Pi w) \\ \Pi U\beta &= Proj_{span(\Pi U)}(\Pi w)\end{aligned}$$

Recall that Π is a 0.01-subspace embedding for the span of A (which is also the span of U). This means that ΠU has all singular values in the range $[0.99, 1.01]$, because U is an orthonormal matrix with all singular values equal to 1. To give some intuition, just think about using this matrix on a vector: ΠUy. U first maps the vector into the column space of U, preserving the length. Π then nearly preserves the length, since (Uy) is now a vector in the subspace where we have guarantees. We can say: $\|y\| = \|Uy\| \approx \|\Pi Uy\|$.

Since we know that this matrix ΠU nearly preserves length (up to a constant factor) we can say also say the following. Note that we can left multiply the final rearrangement in the previous equation by $(\Pi U)^T$, and substitute it in here:

$$\begin{aligned}\|\beta\| &\approx_{O(1)} \|(\Pi U)^T(\Pi U)\beta\| \\ &= \|(\Pi U)^T \Pi w\|\end{aligned}$$

Finally, by the approximate matrix product property, we have that

$$Pr_\Pi\left[\|(\Pi U)^T \Pi w - U^T w\|_F^2 < \varepsilon/d \|U\|_F^2 \|w\|_2^2\right] > 9/10$$

Note that $\|U\|_F^2 = d$ because U is orthonormal, and that $U^T w = 0$ because w is our error vector, which is perpendicular to the column space of A, and therefore also perpendicular to the column space of U.

We can substitute both of these facts into the above inequality. We can also use the approximate equality expression that we previously found for β in terms of Π, U, and w, and fold the constant factors into our $O(\cdot)$ notation. Thus, with high probability, we have that:

$$\begin{aligned}\|\beta\|^2 &\leq O(\frac{\varepsilon}{d}) \cdot d \cdot OPT^2 \\ &= O(\varepsilon \cdot OPT^2)\end{aligned}$$

Finally, we can substitute this into our expression for the total error of our projected solution:

$$\begin{aligned}\|A\tilde{x}^* - b\| &= \|\beta\|^2 + OPT^2 \\ &= (1 + O(\varepsilon))OPT\end{aligned}$$