Solving Least Squares with LSRN

CS 6220. Feb 6th, 2020

Instructor: Anil Damle

Scribes: Marissa Gee, Frank Li, Misha Padidar

1 Introduction

In this paper we will present the LSRN iterative algorithm for solving the Linear Least Squares Problem. Linear Least Squares finds a linear regressor for a data set which minimizes the squared residual. Most frequently it appears in statistics, data science, and machine learning though the techniques are put to use in nearly any field using data. Typically, Linear Least Squares is written as the following optimization problem:

$$\min_{x} \|Ax - b\|_2 \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. In general, the number of data points m is larger than the number of features n, and so the resulting system is over-determined. Moreover the solution can be found analytically via the *Normal Equations*

$$x^* = A^\dagger b = (A^T A)^{-1} A^T b \tag{2}$$

where $A^\dagger$ is the Moore-Penrose pseudo inverse.

In practice, there are many methods to solve this problem, most of which depend greatly on the size of m, n as well as structure in the data A. For problems which fit in memory, direct methods are preferred as they are faster and more accurate. However as m or n becomes large, iterative methods become preferable as run-time and solution accuracy can be controlled. In either case, preferred solution schemes seldom *explicitly* form the normal equations as they greatly increase the computational cost of solving the system and are significantly more ill conditioned than the original system.

The method presented here, LSRN, develops a preconditioner for iterative method to solve the strongly over- or under-determined least squares problem. The preconditioned system provably has an eigen-spectrum that is independent of the original system [4]. The practical benefit of this is that the well conditioned system gives LSRN “fully predictable run-time performance” [4]. This is an extreme benefit over other applying other iterative solvers such as CGLS or LSQR out of the box, in which case it is difficult to predict how many iterations the solver will take to converge.

In the remainder of the document we assume that the system is strongly over-determined, $m >> n$, and that A is full column rank, although LSRN can deal with the case when it is not by finding the unique minimum length solution. We refer the reader to the primary text for methods on strongly under-determined systems [4]. Lastly, we would like to note that LSRN can solve Least Squares with Tikhonov Regularization; again we refer the reader to the primary document [4].

2 LSRN

2.1 Motivation

By improving conditioning of the system, preconditioners for iterative methods can greatly reduce the iteration count required for convergence. LSRN leverages this fact by developing a general right-preconditioner to be used by iterative methods for solving least squares problems. Solving the right preconditioned system amounts to first solving

$$\min_y ||ANy - b||_2 \tag{3}$$

and then computing $x^* = Ny^*$, where N is the right-preconditioner and y^* is the solution to (3). We can guarantee that the right-preconditioned system has the same solution as the original system so long as $range(N) = range(A^T)$ [4].

LSRN has two general steps: First it uses randomness to generate a right-preconditioner; Second it employs an iterative method to solve the preconditioned system.

Let us explore the first step. Suppose $A = U\Sigma V^T$ is the SVD of A. Since U has orthonormal columns, it is perfectly conditioned. In an ideal world, we could choose $N = V\Sigma^{-1}$ as our preconditioning matrix, so that the least squares problem would reduce to min $||Ux - b||_2$ which is easy to solve. However, computing the SVD of A is prohibitively expensive as m becomes large. To ameliorate this problem, LSRN first preforms a dimension reduction on A by applying a random matrix, $\tilde{A} = GA$, then approximates $V\Sigma^{-1}$ by computing the SVD of this smaller matrix. As we will see in 3, the use of randomness in this problem is justified, as it allows us to disassociate the conditioning of AN from the the conditioning of A.

The second step of LSRN simply applies an iterative method to the preconditioned system. The authors of LSRN recommend Chebychev Semi-Iterative Methods (CS) or Conjugate Gradient based schemes such as LSQR [4, 3, 5]. Throughout this document we will introduce bounds for LSQR, and not CS iterations as LSQR is a highly favorable Least Squares Solver. LSQR can be thought of as mathematically equivalent to applying conjugate gradient methods to solve the normal equations, but with additional computational efficiency. LSQR is well-studied, and we can get a bound on the relative error of our solution based on the conditioning of the normal equations.

Theorem 1. *Let $x^{(k)}$ be the current iterate after k iterations of the LSQR algorithm. Additionally, let $x^{(0)}$ be the initial guess and x^* be the true solution, then*

$$\frac{\|x^{(k)} - x^*\|_{A^TA}}{\|x^{(0)} - x^*\|_{A^TA}} \leq \left(\frac{\sqrt{\kappa(A^TA)} - 1}{\sqrt{\kappa(A^TA)} + 1} \right)^k \tag{4}$$

where $\| \cdot \|_{A^TA}$ is the norm induced by the symmetric positive definite matrix A^TA.

A few notes about the bound. First, it can be converted into a bound using the 2-norm, but this leads to an additional constant factor in the inequality. Second, we can see that as the condition number of A^TA increases, the base on the right-hand side approaches one, and thus the algorithm can theoretically converge much more slowly. The LSRN preconditioner solves this problem by vastly improving the condition number of the matrix used in the LSQR iterations.

In the following subsection we will introduce the LSRN algorithm and give a more detailed justification of it's impressive performance.

2.2 LSRN Algorithm

The LSRN algorithm for over-determined systems is as follows:

Algorithm 1: LSRN for Over-determined Systems

```
1. Choose oversampling factor $\gamma > 1$ and set $s = \lceil \gamma n \rceil$
2. Draw a matrix $G$ that belongs to $\mathbb{R}^{s \times m}$ such that the entries are i.i.d. and
   sampled from the Standard Normal distribution.
3. Set $\tilde{A} = GA$.
4. Compute $\tilde{\Sigma} \in \mathbb{R}^{n \times n}$ and $\tilde{V}^T \in \mathbb{R}^{n \times n}$ such that $\tilde{A}$ has the reduced SVD $\tilde{U}\tilde{\Sigma}\tilde{V}^T$. $\tilde{U}$
   is not necessary to compute.
5. Set $N = \tilde{V}\tilde{\Sigma}^{-1}$.
6. Apply an Iterative Method to solve $\hat{y} = \text{argmin}_y \|ANy - b\|_2$.
7. Return $\hat{x} = N\hat{y}$.
```

In the above algorithm, $\tilde{A} \in \mathbb{R}^{s \times n}$, so for $n << m$, computing the SVD of $\tilde{A}$ is much more efficient than solving the normal equations for A. Furthermore we note that with probability one range(N) = range(A^T), which guarantees that the preconditioned systems will converge to the true solution.

We can now formally show how LSRN preconditioning improves the condition number of the Least Squares system. Recalling the SVD of A, Algorithm 1 sets $\tilde{A} = GU\Sigma V^T$. As a rotation of a Guassian random matrix is still a Guassian random matrix, the product $G_1 = GU$ will also be i.i.d. and distributed according to the standard normal distribution. Further, we can substitute in the SVD of G_1 to get

$$\tilde{A} = GU\Sigma V^T = U_1\Sigma_1 V_1^T \Sigma V^T \tag{5}$$

As U_1 is orthonormal we can decompose it as $U_1 = \tilde{U}Q_1$ where $\tilde{U}$ is part of the SVD of $\tilde{A}$, and Q_1 is some orthonormal matrix. However as we know the SVD of $\tilde{A}$ is $\tilde{U}\tilde{\Sigma}\tilde{V}^T$, equation (5) implies that $\tilde{\Sigma}\tilde{V}^T = Q_1\Sigma_1 V_1^T \Sigma V^T$, and thus setting $N = \tilde{V}\tilde{\Sigma}^{-1}$ is equivalent to setting

$$N = V\Sigma^{-1}V_1\Sigma_1^{-1}Q_1^T \tag{6}$$

and thus

$$AN = UV_1\Sigma_1^{-1}Q_1^T. \tag{7}$$

Since all of the other matrices are orthonormal, the condition number of the preconditioned matrix, $\kappa(AN)$, depends only on the Σ_1^{-1}, the singular values of the gaussian random matrix G_1. So now we would like to derive formal bounds on the singular values of a Gaussian random matrix, which in turn would bound the condition number of the preconditioned matrix AN.

Theorem 2. *Consider $G_1 \in \mathbb{R}^{s \times n}$ with iid standard normal entries, then for any $t > 0$, we can bound the probabilities P*

$$\max\left\{P(\sigma_1 \geq \sqrt{s} + \sqrt{n} + t), P(\sigma_n \leq \sqrt{s} - \sqrt{n} - t)\right\} \leq \exp\left(\frac{-t^2}{2}\right) \tag{8}$$

This gives us a probabalistic bound on the largest and smallest singular values of G_1, which we can translate into a bound on the condition number of AN:

Theorem 3. *For any* $\alpha \in \left(0, 1 - \sqrt{\frac{n}{s}}\right)$ *then*

$$P\left(\kappa(AN) \leq \frac{1 + \alpha + \sqrt{\frac{n}{s}}}{1 - \alpha - \sqrt{\frac{n}{s}}}\right) \geq 1 - \exp\left(\frac{-\alpha^2 s}{2}\right) \tag{9}$$

Finally, with this new bound on the condition number of AN, we can use Theorem 1 to bound the number of iterations LSQR will take to converge on the preconditioned Least Squares system.

Theorem 4. *Given a tolerance* $\epsilon > 0$*. If* x^* *is the true least squares solution and* $\alpha \in (0, 1 - \sqrt{n/s})$*, then with probability* $1 - \exp(-\alpha^2 s/2)$ *a Conjugate Gradient-like method (LSQR) will have error within*

$$\frac{\|x^{(k)} - x^*\|_{A^TA}}{\|x^*\|_{A^TA}} \leq \epsilon \tag{10}$$

after $k = (\log \epsilon - \log 2)/\log(\alpha + \sqrt{n/s})$ *iterations. where* $||\cdot||_{A^TA}$ *is the norm induced by the positive definite matrix* A^TA*.*

Now with high probability we have a guarantee of our solution accuracy. With this bound we can tune the number of iterations LSRN performs in order to achieve desired accuracy.

In the next section we will analyze the cost of using LSRN.

3 Complexity Analysis

The complexity analysis is largely dependent on the cost of a matrix vector product with A, denoted τ_A, which is assumed to be the largest cost. Assuming the average cost of generating a standard normal random number is c, the cost of Algorithm 1 is

1. $s \times m \times c$ for generating G
2. $s \times \tau_{A^T}$ for computing $\tilde{A}$
3. $2sn^2 + 11n^3$ cost for computing $\tilde{\Sigma}$ and $\tilde{V}^T$. [4, 2]
4. $N_{iter}(\tau_A + \tau_{A^T} + 4n^2)$ for applying an iterative method to solve preconditioned least squares.

where N_{iter} is the number of iterations of the solver. Notice that the computational burden of the SVD in Algorithm 1 is greatly reduced as it is not necessary to compute $\tilde{U}$. Assuming $\tau_{A^T} = \tau_A$, this brings us to a total cost of $\mathcal{O}(11n^3 + (2s + 4N_{iter})n^2 + (2N_{iter} + s)\tau_A + s \times m \times c)$. The number of iterations can be estimated with high probability using our bound 4. For example with an oversampling factor $\gamma = 2$, we can achieve an error tolerance $\epsilon = 10^{-14}$ within $N_{iter} \approx 100$ iterations [4]. This predictability is greatly beneficial, as many Least Squares solvers do not have guarantees for general matrices A [4].

An additional benefit of LSRN is that it is easily parallelizable. The expensive matrix vector product GA can be easily solved across machines, along with the generation of random variables and the SVD of $\tilde{A}$

4 Concluding Remarks

LSRN leverages randomness to develop a preconditioner for Least Squares iterative solvers. When using a Conjugate Gradients based iterative method, such as LSQR, LSRN becomes a powerful, tunable and general least squares solver. We recommend LSRN with LSQR for poorly conditioned systems, where the preconditioning can greatly augment the iteration time and accuracy of an iterative method. The preconditioning, however, comes at a computational expense, so it is not recommended for systems that are extremely well conditioned. In other cases, it may be possible to generate highly effective preconditioners for specialized problems using domain specific knowledge, which may be more effective than LSRN. The advantage of LSRN is that it does not require specific knowledge of the problem to generate a preconditioner. Though LSRN is a powerful method, it does not seem any more favorable than other state-of-the-art methods such as Blendenpik. The authors found that LSRN outperforms Blendenpik when A is sparse [4], but is slightly less efficient for dense A [1, 4]. To that end, we recommend the reader consider application specific information prior to choosing a Least Squares solver.

References

[1] Haim. Avron, Petar. Maymounkov, and Sivan. Toledo. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

[2] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[3] Gene H Golub and Richard S Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods. 1961.

[4] Xiangrui Meng, Michael A Saunders, and Michael W Mahoney. LSRN: A parallel iterative solver for strongly over- or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

[5] Christopher C Paige and Michael A Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.