# An Empirical Evaluation of Sketching for Numerical Linear Algebra

Yogesh Dahiya
IIT Kanpur
Kanpur, India
yogeshdh@iitk.ac.in

Dimitris Konomis
CMU
Pittsburgh, USA
dkonomis@cs.cmu.edu

David P. Woodruff
CMU
Pittsburgh, USA
dwoodruf@cs.cmu.edu

## ABSTRACT

Over the last ten years, tremendous speedups for problems in randomized numerical linear algebra such as low rank approximation and regression have been made possible via the technique of randomized data dimensionality reduction, also known as sketching. In theory, such algorithms have led to optimal input sparsity time algorithms for a wide array of problems. While several scattered implementations of such methods exist, the goal of this work is to provide a comprehensive comparison of such methods to alternative approaches. We investigate least squares regression, iteratively reweighted least squares, logistic regression, robust regression with Huber and Bisquare loss functions, leverage score computation, Frobenius norm low rank approximation, and entrywise $\ell_1$-low rank approximation. We give various implementation techniques to speed up several of these algorithms, and the resulting implementations demonstrate the tradeoffs of such techniques in practice.

## CCS CONCEPTS

• **Theory of computation → Numeric approximation algorithms**;

## KEYWORDS

Logistic Regression, Low Rank Approximation, Regression, Robust Methods, Sketching

**ACM Reference Format:**
Yogesh Dahiya, Dimitris Konomis, and David P. Woodruff. 2018. An Empirical Evaluation of Sketching for Numerical Linear Algebra. In *KDD 2018: 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3219819.3220098

## 1 INTRODUCTION

A number of recent advances in numerical linear algebra have been made possible by techniques in randomized algorithms. Using the technique of linear sketching, one first compresses a given matrix to a much smaller matrix by multiplying it by a random matrix with certain properties. Much of the expensive computation can then be performed on the smaller matrix, thereby accelerating the solution for the original problem. Several well-studied problems in this area include least squares regression, low rank approximation, and robust variants of these.

Many of these problems can be solved deterministically and exactly. For example, for the least squares regression problem one can solve the normal equations, while for low rank approximation one can compute the singular value decomposition (SVD). However, on large scale datasets, these algorithms are often too slow, and randomization has led to much faster, often optimal algorithms for these problems.

A typical approach is to *sketch and solve*. For example, in overconstrained least squares regression we solve $\min_{x\in\mathbb{R}^d} \|Ax-b\|_2$, where $A$ is a given "tall and thin" $n \times d$ matrix with $n \gg d$, and $b$ is a given $n \times 1$ vector. One chooses a random $s \times n$ matrix $S$ with $s \ll n$, and $S \cdot A$ *compresses* $A$ to an $s \times d$ matrix, which is called a *sketch*. We also compute $S \cdot b$. We then solve a much smaller regression problem $\min_{x\in\mathbb{R}^d} \|SAx - Sb\|_2$, - henceforth referred to as the sketched problem, which we can afford to solve exactly. For appropriate $S$, if one takes the $x'$ minimizing the small regression problem, then $\|Ax' - b\|_2 \leq (1+\epsilon)\min_x \|Ax-b\|_2$ with high probability, where $\epsilon > 0$ is a given accuracy parameter. Since the small regression problem is efficiently solvable, the bottleneck is just in computing $S \cdot A$. Although naïvely this multiplication could take *snd* time, one can choose $S$ to be a Fast Johnson Lindenstrauss transform (FJLT) [19], or a CountSketch matrix [4, 5, 7, 14, 16]. Using a CountSketch matrix, $S \cdot A$ can be computed in $\text{nnz}(A)$ time, where $\text{nnz}(A)$ denotes the number of non-zero entries of $A$. Some preliminary CountSketch algorithms have been implemented in the Skylark package [1], though as far as we are aware, only for the basic least squares regression setting described above, and not for the many regression variants we describe below or for the many variants of low rank approximation we consider.

*Our Contributions.* We give the first comprehensive implementation of sketching-based methods in a wide range of variants of regression and low rank approximation, empirically validating their performance.

First, we consider sketching-based preconditioning. An issue with the solution above for regression is that the dependence on $\epsilon$ is quadratic. Alternatively, in [5], a constant-factor initial solution together with a constant-factor preconditioner are obtained using CountSketch. These are then used in a gradient-descent algorithm for regression with only $\log(1/\epsilon)$ iterations. We refer to this technique as sketching-based preconditioning. Although sketching-based preconditioning has been implemented in [3, 6, 15], the use of CountSketch has not been explored in this context. We show

[1] https://github.com/xdata-skylark/libskylark


*KDD 2018, August 19–23, 2018, London, United Kingdom*

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220098

in practice, with the same sketching dimension, the constant factor preconditioner provided by CountSketch is as good as the one provided by other slower sketching matrices such as the FJLT and random Gaussian projections. We then use sketching-based preconditioning to speed up the iteratively reweighted least squares (IRLS) routine in logistic and robust regression and identify other optimization problems where this can help improve efficiency and numerical stability of the existing iterative methods. We also give results on leverage score approximation. To the best of our knowledge, sketching-based preconditioning with CountSketch has not been implemented for any of these tasks.

We next consider the low rank approximation problem, namely, that of finding a rank-$k$ matrix $B$ for which $\|A - B\|$ is small for a given matrix $A$. Here we consider both the standard Frobenius norm, as well as recent more robust variants based on the entrywise $\ell_1$-norm. As far as we are aware, only very preliminary implementations with synthetic data sets have been performed on such error measures, in [5] and [20], respectively.

In terms of results, for sketching-based preconditioning we observe that for least squares regression (LS) we obtain speedups of a factor of 2-3 over normal equation-based methods, and improved iterative methods by a factor of 20. For Huber regression, we consistently obtain speedups of a factor 4 on synthetic data, while on large real datasets we obtain speedups as large as 20. For logistic regression we improve the Newton method by a factor of 3-4.

We next describe our results for approximate low rank approximation algorithms with respect to both the Frobenius (approximate $\ell_F$-algorithm) and entrywise $\ell_1$ norm (approximate $\ell_1$-algorithm). For each algorithm, we measure the approximation factor and its running time on 3 synthetic datasets (three random $n = 10^5 \times m = 10^3$ matrices $M$ with $\|M\|_0 = 0.25nm$, $0.50nm$ and $0.75nm$ nonzero entries, respectively) as well as on the NIPS dataset [17] (an $n = 11463 \times m = 5812$ matrix $M$ containing the number of times each of 114634 words appear in every one of the 5811 NIPS conference papers published from 1987-2015) and the movielens dataset [12], which contains 1M (million) ratings (0-5) from 6000 users on 4000 movies. The approximate-$\ell_F$ algorithm finds a solution with cost at most 10 times the optimal one (as found by the truncated SVD) but does so 10 times faster.

We view our empirical results as strong evidence of the practical value of the recent theoretical breakthroughs in sketching-based methods for regression, low rank approximation, and their many variants. Our code is available at https://goo.gl/932E9q.

## 2 PRELIMINARIES

### 2.1 Subspace Embeddings

Definition 2.1. *A $(1 \pm \epsilon)$ $\ell_2$-subspace embedding for the column space of an $n \times d$ matrix $A$ is a matrix $S$ for which for all $x \in R^d$, $\|SAx\|_2^2 = (1 \pm \epsilon)\|Ax\|_2^2$.*

Definition 2.2. *A CountSketch with sketching dimension $s$ is a random linear map $\Phi D : R^n \to R^s$ defined as follows. Let $h : [n] \to [s]$ be a random map, so that for each $i \in [n]$, $h(i) = s'$ for $s' \in [s]$ with probability $1/s$. Then $\Phi \in \{0,1\}^{s \times n}$ is an $s \times n$ binary matrix with $\Phi_{h(i),i} = 1$, and all remaining entries 0. Define $D$ to be an $n \times n$ random diagonal matrix, with each diagonal entry independently chosen to be $+1$ or $-1$ with equal probability.*

The following theorem from [5, 14, 16] shows that CountSketch is an $\ell_2$-subspace embedding:

Theorem 2.1. *The Count-Sketch matrix $S \in R^{s \times n}$ with $s = O(r^2/(\epsilon^2\delta))$ rows is a $1 \pm \epsilon$ subspace embedding for any fixed $A \in R^{n \times d}$ of rank $r$ with probability at least $1 - \delta$. Further, this holds if the hash function $h$ defining $S$ is only pairwise independent, and the diagonal entries of $D$ defining $S$ are only 4-wise independent.*

Corollary 2.1.1. *A subspace embedding preserves the spectrum, i.e., the set of singular values, of the input matrix $A$, namely $\sigma_k(SA) = (1 \pm O(\epsilon))\sigma_k(A)$ where $S$ is a $(1 \pm \epsilon)$ $\ell_2$ subspace embedding.*

Note that Corollary 2.1.1 follows from a direct application of the Courant-Fischer-Weyl min-max principle and the definition of a subspace embedding (see, e.g., Appendix C of [13]).

### 2.2 Leverage Scores

Leverage scores measure the "importance" of a row of $A$ in the row space of a matrix $A$. They have been used in various randomized numerical linear algebra problems and estimating them is an important problem in its own right. For a matrix $A \in R^{n \times d}$, the $i$-th leverage score of row $i \in [n]$ is $h_i = [A(A^TA)^-A^T]_{ii}$, where for a matrix $C$, $C^-$ denotes its Moore-Penrose pseudoinverse. Algorithm 1 provides a sketching-based method for estimating leverage scores for an over-determined linear system [21].

**Algorithm 1** Estimating Leverage scores

**Input:** $A \in R^{n \times d}$, with $n \gg d$
**Output:** $h$ leverage scores
1: Find a right preconditioner $P$ to $A$ using Algorithm 2
2: Generate $G \in R^{r \times t}$ of i.i.d. Gaussian$(0, 1/t)$ random variables, where $r = \text{rank}(A)$ and $t = (\log n)/0.1$
3: Calculate $H = A \cdot (P \cdot G)$
4: $h = \text{diag}(HH^T)$

## 3 REGRESSION

### 3.1 Least Squares Regression

Given a matrix $A \in R^{n \times d}$ and a vector $b \in \mathbb{R}^n$, the LS problem seeks to find the vector $x^* \in \mathbb{R}^d$ for which:

$$x^* = \arg\min \|x\|_2, \text{ subject to } x \in \arg\min_{z \in R^d} \|Az - b\|_2. \quad (1)$$

In this work we consider over-determined ($n > d$) and under-determined ($n < d$) instances of the LS problem (1). Traditional ways of solving LS include orthogonal factorization methods such as SVD and QR, normal equation-based methods like Cholesky factorization, and LU and iterative methods like conjugate gradient and LSMR [9].

In more detail, it is well-known that the optimal solution of the problem (1) is $x^* = A^-b$. Recall that if $U\Sigma V^T$ is the compact SVD of $A$, then the pseudoinverse is given by $A^- = V\Sigma^{-1}U^T$. This approach is accurate and robust to rank-deficiency. Another approach to solve this problem is via the normal equations $A^TAx = A^Tb$, which characterizes all solutions to (1). To obtain the minimum norm solution we have $x^* = (A^TA)^-A^Tb = A^T(AA^T)^-b$. If $A$ has full column rank

then $A^TA$ is invertible and positive definite and (1) can be solved using a Cholesky factorization. A similar argument applies when $A$ has full row rank using $AA^T$. If $\text{rank}(A) < \min(n, d)$, then we would need to find the eigensystem of $A^TA$ or $AA^T$. The normal equations approach is the least expensive, but is also the least accurate, especially on ill-conditioned problems as it squares the condition number while forming the normal equations. For details, we refer the reader to chapter 5 of [11]. A third approach to solving (1) is by applying Krylov subspace methods on the normal equations. For example, conjugate gradient or LSMR could be applied. Although the cost per iteration of these procedures is $\text{nnz}(A)$, the convergence rate is proportional to $(\frac{\sqrt{\kappa(A^TA)}-1}{\sqrt{\kappa(A^TA)}+1})^k$, where $\kappa(A^TA)$ is the condition number of $A^TA$ and $k$ is the iteration count, making its time complexity $O(\text{nnz}(A)\sqrt{\kappa(A^TA)}\log(\frac{1}{\epsilon}))$ for $\epsilon$ relative accuracy. Because of its dependence on the condition number, this method is not preferred for ill-conditioned problems. On the other hand, it performs competitively for large well-conditioned (having small constant upper bound on the condition number) problems. In light of this, a lot of work has been done to transform problem (1) to an equivalent problem with a reduced condition number. This step is known as preconditioning, which we now describe.

Note for any non-singular matrix $P$, solving the following problem $y^* = \arg\min_y \|Py\|_2$ s.t. $y \in \arg\min_z \|APz - b\|_2$ gives a solution to the problem (1) as $x^* = Py^*$, by a simple change of variables. Also if we consider $P$ such that $\mathcal{R}(P) = \mathcal{R}(A^T)$, where for a matrix $C$, $\mathcal{R}(C)$ denotes its column space, then the solution to $y^* = \arg\min_y \|y\|_2$ s.t. $y \in \arg\min_z \|APz - b\|_2$, gives us a solution (1) as $x^* = Py^*$. This is true as when $\mathcal{R}(P) = \mathcal{R}(A^T)$, $A^- = P(AP)^-$. To see this, let $A = U\Sigma V^T$ be the compact SVD of $A$ and $P = VZ$, where $Z$ has full row rank. Then note that $P(AP)^- = P(AP)^T(AP(AP)^T)^- = VZZ^T\Sigma U^T(U\Sigma ZZ^T\Sigma U^T)^- = VZZ^T\Sigma U^TU\Sigma^{-1}(ZZ^T)^{-1}\Sigma^{-1}U^T = V\Sigma^{-1}U^T = A^-$. This is known as right preconditioning. Also if we consider $P$ for which $\mathcal{R}(A) = \mathcal{R}(P)$, then the solution to $y^* = \text{argmin}_y||y||$ s.t. $y \in \text{argmin}_z||P^TAz - P^Tb||_2$ is the same as $x^*$. This is similar to the result above. When $\mathcal{R}(P) = \mathcal{R}(A)$, then $(P^TA)^-P^T = A^-$. This is known as left preconditioning. Algorithm 2 gives an algorithm for right preconditioning, and the algorithm for left preconditioning is obtained just by applying Algorithm 2 to $A^T$. To see that $AP$ is well-conditioned when $P$ comes from the right preconditioner algorithm, we use the fact that $SA$ has the same spectrum as $A$ up to a small constant factor (recall Corollary 2.1.1), and so a good preconditioner for $SA$ will be a good preconditioner for $A$. Formally, we have $\|APx\|^2 \leq (\frac{1}{1-\epsilon})\|SAPx\|^2 = (\frac{1}{1-\epsilon})\|Ux\|^2 = (\frac{1}{1-\epsilon})\|x\|^2$. Similarly, $\|APx\|^2 \geq (\frac{1}{1+\epsilon})\|x\|^2$, giving us an $O(\frac{1+\epsilon}{1-\epsilon})$ bound on the condition number of $AP$. We also note that $\mathcal{R}(P) = \mathcal{R}(A^T)$. To see this, let $A = U\Sigma V^T$ and $SA = U_S\Sigma_S V_S^T$ be the compact SVD of $A$ and $SA$. Then $\mathcal{R}(P) = \mathcal{R}(V_S\Sigma_S^{-1}) = \mathcal{R}((SA)^T) = \mathcal{R}(V\Sigma(SU)^T) = \mathcal{R}(V) = \mathcal{R}(A^T)$, where we have used the fact that $SU$ has full column rank. For more details, we refer the reader to [21] and [15].

We note that when right preconditioning (Algorithm 2) a full rank matrix $A$, the expensive SVD factorization of $SA$ can be replaced by a cheaper QR factorization, as $R^{-1}$ turns out to be a right preconditioner with high probability. We implement Algorithm 2 and combine it with LSMR to illustrate the effectiveness of sketching-based preconditioning. We evaluate our approach, which we call PLSMR, on both synthetic and real datasets.

**Algorithm 2** Right Preconditioner

**Input:** $A \in R^{n\times d}$, with $n \gg d$; $s$ sketching dimension
**Output:** $P$ preconditioner; $r = \text{rank}(A)$
1: Generate $S$ =Count-Sketch$(s, n)$
2: $U, \Sigma, V$ =SVD$(S \cdot A)$
3: $tol = \Sigma[0, 0] \times n \times \epsilon_{mac}$ {$\epsilon_{mac}$ is machine precision}
4: Compute rank of $SA$, $r = sum([\sigma > tol \text{ for } \sigma \in \Sigma])$
5: $P = V[:, 0 : r] \cdot \Sigma[0 : r, 0 : r]^{-1}$

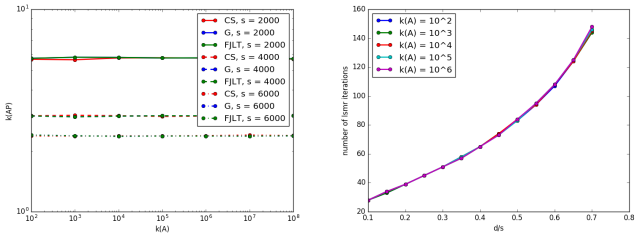



**Figure 1: Left: $\kappa(A)$ vs $\kappa(AP)$ for different sketching dimensions $s$, for Gaussian ( denoted by G), Count-Sketch (denoted by CS) and the FJLT sketching matrix. Right: Number of LSMR iterations vs. $\frac{d}{s}$ for CountSketch preconditioning.**

*3.1.1 Condition number vs. Sketching Dimension.* We compare the (right) preconditioning quality provided by CountSketch, FJLT, and Gaussian sketching matrices $S$. In theory, CountSketch is not optimal with respect to the sketching dimension (number of rows of $S$), but in practice it provides similar conditioning to the other two sketching matrices. We generate matrices of size $n \times d = 100K \times 1K$, with non-zero density 0.1 (i.e., $0.1nd$ non-zero elements) and condition number ranging from $10^2$ to $10^8$. We use MATLAB's "sprandn" function for synthetic data generation. The left plot in Figure 1 compares $\kappa(AP)$ with $\kappa(A)$ for sketches of varying sketching dimension $s$. We observe that CountSketch performs as well as the other sketching matrices. Moreover from the plots we see that $\kappa(AP)$ is independent of $\kappa(A)$ and we get good conditioning, $\kappa(AP) \approx 5$, with $s$ being just $2d$. The right plot in Figure 1 measures the number of iterations taken by LSMR against the oversampling factor, $\gamma = \frac{d}{s}$, and allows for the following observations. First, as expected, the number of iterations taken by LSMR is independent of the condition number of the input matrix (the number of iterations depends on $\kappa(AP)$, which is independent of $\kappa(A)$). Second, the number of LSMR iterations decreases as the sketching dimension increases.

*3.1.2 LS Experiments on Synthetic Datasets.* The left plot in Figure 2 gives the time taken by various components of PLSMR w.r.t. the sketching dimension. We observe that as the sketching dimension increases, the time taken to generate and apply sketch $S$ to $A$ does not change much, whereas the time taken to QR-factorize $SA$ increases, and the time taken by LSMR decreases, giving us the optimal choice for $s$. We found that in practice setting $\gamma$ to a value

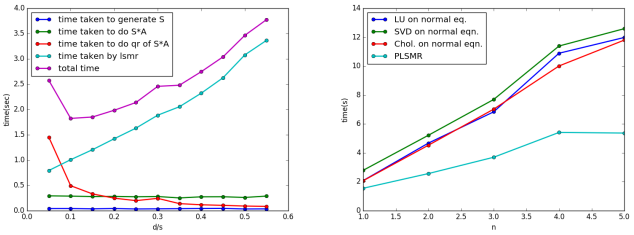



**Figure 2: Left: time vs $\frac{d}{s}$ of various parts of PLSMR on an input matrix of size $1M \times 1K$, 1% non-zero density and condition number $1e6$. Right: Running times of various methods for solving the LS problem on $n \cdot M \times 1K$ sized matrices, with 1% density of non-zero entries, and $1e6$ condition number.**

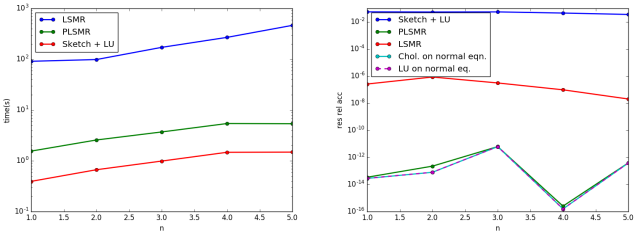



**Figure 3: Left: Running times of PLSMR, LSMR and Sketch + LU on $n \cdot M \times 1K$ sized matrices, with 1% density of non-zero entries, and $1e6$ condition number. Right: Relative accuracy of the residual, $|\|A\hat{x} - b\|_2 - \|Ax^* - b\|_2| / \|Ax^* - b\|_2$, where $x^* = (A^T A)^- A^T b$ is the pseudoinverse solution to the normal equations and $\hat{x}$ is the solution for a particular method.**

between 0.2 and 0.5 works well. Next, we compare various traditional ways of solving the LS problem. We created matrices $A$ with varying dimensions $n \times d = nM \times 1K$ ($n \in \{1, 2, 3, 4, 5\}$), 1% non-zero density and condition number $\kappa(A) = 1e6$. The response variable $b$ was generated as $Ax^* + scale \cdot \epsilon$, where $x^*$ and $\epsilon$ are standard i.i.d. normal random variables and $scale = 0.25 \frac{\|Ax^*\|_2}{\|\epsilon\|_2}$. The right plot in 2 compares normal equation-based methods with PLSMR. The left plot in 3 compares LSMR, the PLSMR method on (1), and LU factorization on the sketched problem (Sketch + LU). The right plot in 3 compares the solution accuracy of the various methods. As the plots indicate, PLSMR gives a speedup of 2 over normal equation-based methods and improves LSMR by a factor of 20 with its accuracy on par with traditional methods. Note that although LU on the sketched problem runs faster than PLSMR, it does not have the same accuracy guarantees because it solves the sketched problem which only provides a $(1+O(\epsilon))$-approximate solution. We do not compare our approach with orthogonal factorization-based methods like SVD and QR, since they do not perform competitively on large over/under-determined matrices.

**Table 1: Real datasets**

| Datasets | n | d | nnz | Rank | $\kappa$ |
|---|---|---|---|---|---|
| twitter | 583,250 | 76 | 41.2e6 | 76 | 1.6e8 |
| landmark | 71,952 | 2704 | 1.1e6 | 2671 | 1e8 |
| connectus | 394,792 | 512 | 1.1e6 | 458 | 1e15 |
| rail2586 | 923,269 | 2,586 | 8.0e6 | 2586 | 496 |

**Table 2: LS on real datasets**

| Datasets | CNEQ | SNEQ | SVD | LSMR | PLSMR |
|---|---|---|---|---|---|
| twitter | 0.059s | 0.066s | 1.54s | 16.0s | 0.72s |
| landmark | - | 30.8s | 54.6s | 31.25s | 10.0s |
| connectus | - | 0.16s | 17.0s | 4.95s | 0.33s |
| rail2586 | 0.66s | 21.5s | - | 37.6s | 3.63s |

CNEQ : Cholesky on normal equations; SNEQ : SVD on normal equations.

*3.1.3 LS Experiments on Real datasets.* Table 1 sheds light on the parameters of datasets used for our linear and robust regression experiments. The landmark, connectus and rail2586 datasets are from the University of Florida Sparse Matrix Collection [8], and twitter is from the UCI ML repository [2]. Table 2 compares the performance of PLSMR with other methods. We observe that PLSMR significantly outperforms SVD and LSMR, and beats normal equation-based methods for large and dense matrices where setting up the normal equations requires $\Theta(nd^2)$ time. This is not true in the case of the rail2586 dataset, as it is extremely sparse and setting up the normal equations is fast for this particular dataset.

## 3.2 Robust Regression

It is well-known that ordinary least squares (OLS) is not robust to outliers. Robust regression methods are designed to dampen the influence of outlying cases in order to provide a better fit to the majority of the data. The most common method for robust regression is M-estimation. We define the problem of M-estimation (we call this RR, for robust regression). For a design matrix $A \in R^{n \times d}$ and response variable $b$, RR is the problem of solving $\min_{x \in R^d} \|Ax - b\|_G$ with $\|y\|_G = \sum_i G(y_i)$ where $G$ is a loss function having the following properties: 1) $G(x) \geq 0$, 2) $G(0) = 0$, 3) $G(x) = G(-x)$ and 4) $G(x_1) \geq G(x_2)$ for $|x_1| \geq |x_2|$. One can see that OLS falls into the class of M-estimators with loss function $G(x) = x^2$. Other loss functions include Huber, which is $G(e) = \frac{e^2}{2}$ if $|e| \leq k$, and $G(e) = k|e| - \frac{k^2}{2}$ if $|e| > k$, and Bisquare, which is $G(e) = \frac{k^2}{6}(1-(1-(e/k)^2)^3)$ if $|e| \leq k$, and $G(e) = \frac{k^2}{6}$ if $|e| > k$. The $k$ in the function is the scale parameter which controls the level of resistance to outliers.

We now describe an iteratively reweighted least squares (IRLS) based approach to solve this problem. Note that by setting the gradient of the loss function to zero, we get $\sum_{i=1}^n G'(a_i^T x - b_i) a_i^T = 0$, or equivalently $A^T W A x = A^T W b$, which is a weighted least squares problem, where $W$ is a diagonal matrix with $W_{i,i} = w(a_i^T x - b_i) = G'(a_i^T x - b_i)/(a_i^T x - b_i)$. Since the weights depend on the estimated

coefficients and the estimated coefficients on the weights, this is solved by an iterative procedure where we update the weights and the coefficients alternatively. Algorithm 3 describes this method and is also found in the implementation of Matlab's *robustfit* procedure for robust regression. In steps 2-4 we estimate the stan-

**Algorithm 3** IRLS for Robust Regression (RobustFit)

**Input:** $A \in R^{n\times d}$, $b \in R^n$, loss function $G$
**Output:** $x$ solution to RR

1: Find the initial estimates $x_0$ by running OLS
2: Find the leverage scores of each data point, $h_1, \ldots, h_n$
3: Calculate the residual using the current estimate
4: Estimate the variance $\hat{\sigma}^2$ of the error distribution using the median absolute deviation (MAD) of the residual
5: Scale the residuals to get the studentized residual, where the scale factor for the $i$-th data point is $\frac{1}{\sqrt{\hat{\sigma}^2(1-h_i)}}$
6: Find the weights from the studentized residual
7: Solve the weighted least squares problem $A^TWAx = A^TWb$ using the weights from the previous step to get the new estimate
8: Repeat step 3-7 until the estimates converge

dard deviation of the residuals, which is to be used as a scale parameter for the loss functions for the purposes of renormalization, together with $1 - h_i$. Let $\hat{b}$ be the estimated response variable. Note that the variance of the residual for ordinary least squares is $\text{Var}(\hat{b} - b) = \text{Var}(AA^-b - b) = \text{Var}((I - AA^-)b) = (I - AA^-)\text{Var}(b)(I - AA^-)^T = \sigma^2(I - AA^-)$, where we have assumed that the error distribution has the covariance matrix $\sigma^2 I$. To find $\sigma$, we use a robust estimate, namely the median absolute deviation of the residuals. In step 5 we scale the residuals so that we can use the loss function with scale parameter set to 1. In step 6 we find the weights from the scaled residuals, also known as the studentized residuals. In step 7 we solve the weighted least squares problem and repeat the process in step 8. For details, see [1]. The computationally intensive steps in the algorithm are steps 2 and 7. For over- and under-determined problems we can solve step 7 efficiently using methods from section 3.1 and for step 2 we can use Algorithm 1, giving us the faster version of Algorithm 3 using sketching.

*3.2.1 Experiments for Leverage Score Estimation.* Algorithm 1 is a sketching-based solution for estimating leverage scores. To verify the integrity of the estimates, Figure 4 contains plots of actual and estimated leverage scores, on each of two real datasets.

*3.2.2 Robust Regression Experiments on Synthetic Datasets.* We generate matrices $A$ with varying dimensions $n \times d = n \cdot 100K \times 500$ ($n \in \{1, 2, 3, 4, 5\}$), 10% non-zero density and condition number $\kappa(A) = 1e6$. We then generate $b$ as follows: first, we generate $b^* = Ax^*$ and $b_\epsilon = Ax^* + scale \cdot \epsilon$ where $x^*$ and $\epsilon$ are standard normal random variables and $scale = 0.25\frac{\|Ax^*\|}{\|\epsilon\|}$. Second, we corrupt 10% of the entries of the response variable $b_\epsilon$ by adding noise $1000\epsilon_i$ instead of $\epsilon_i$. This is a simple way of introducing outlier behavior in the data. We run the various algorithms on the pair $(A, b_\epsilon)$. We compare Algorithm 3 with leverage scores and weighted least square

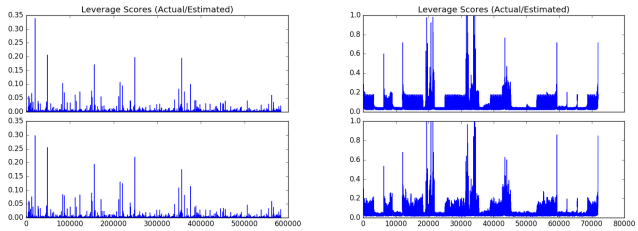



**Figure 4: Leverage Scores (actual/estimated) for twitter and landmark datasets**

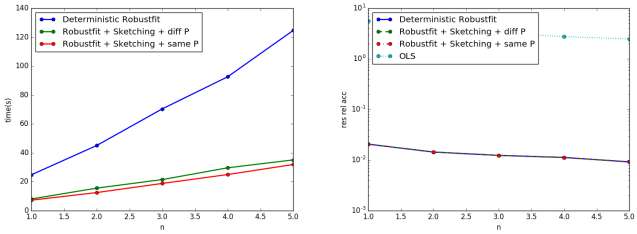



**Figure 5: Left: Running times of Robustfit with and without sketching on $(n \cdot 100K) \times 500$ sized matrices, with 10% density of non-zero entries, and $1e6$ condition number. Robustfit + sketching + diff P (same P) denotes Robustfit with sketching techniques using a different (the same) preconditioner for each WLS problem. Right: Relative accuracy of the residual, $|b^* - \hat{b}|/|b^*|$, where $b^*$ is the response variable without noise and $\hat{b}$ is the predicted response variable. OLS indicates the accuracy of ordinary least squares regression.**

(WLS) problems solved in various ways: (1) using deterministic methods, and (2) using sketching techniques introduced above. The left plot in Figure 5 gives the timing results and the right figure gives accuracy results. As we can see, *RobustFit* with sketching techniques is faster than the deterministic version by a factor of $\sim 5$ and has similar accuracy guarantees. Additionally, we observe that using the same preconditioner performs slightly better than when using a different preconditioner for each WLS problem, as during experiments we observed that the weights did not change by much. Thus we could circumvent the need for a new preconditioner for each WLS problem. We also compared *RobustFit* with sketching with *scipy.optimize.least_squares*'s Trust Region Reflective (TRF) algorithm, and found our technique outperforms it by a factor of 10. The left figure in Figure 6 compares the running time of TRF with the sketching-based Robustfit procedure on synthetic datasets. All experiments were run using the Huber loss function.

*3.2.3 RR Experiments on Real Datasets.* Table 3 shows running times of experiments performed on real datasets. It is evident that sketching-based methods significantly outperform deterministic methods. Note that we were not able to run Algorithm 3 on the rail2586 ($n \times d = 9.2e5 \times 2.5e3$) dataset, as calculating leverage

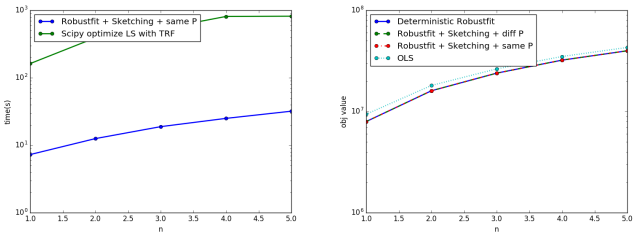



**Figure 6: Left: Running time vs input size of Robustfit with sketching and TRF on synthetic datasets. Right: Objective value vs. input size for robust regression on synthetic datasets. OLS refers to ordinary least squares.**

scores requires a $QR$ or $SVD$ factorization, which creates run-time and memory issues. We also note that on real datasets, the weights do change and the benefits one gets by using the same preconditioner are not enough as the conditioning of the WLS problems will deteriorate over iterations and the performance of the LSMR step will drop.

**Table 3: Robust Regression on Real Datasets**

| Datasets | RF + D | RF + SSP | RF + SDP |
|---|---|---|---|
| connectus | 43.9s | 7.1s | 8.6s |
| landmark | 396.6s | 90.3s | 19.5s |
| twitter | 12.0s | 5.7s | 6.4s |
| rail2586 | - | 119.8s | 104.8s |

RF + D : Deterministic RobustFit; RF + SSP : Sketching-based RobustFit with the same preconditioner; RF + SDP: Sketching-based RobustFit with a different preconditioner

## 3.3 Logistic Regression

Logistic regression is a regression model where the dependent variables are categorical. Given a matrix $A \in R^{n\times d}$ and a response variable $b \in \{0,1\}^n$, the data likelihood function is $P(b|A,x) = \prod_{i=1}^n P(b_i = 1|a_i,x) = \prod_{i=1}^n (\mu_i)^{b_i}(1-\mu_i)^{1-b_i}$, where $\mu_i = \frac{\exp(x^T a_i)}{1+\exp(x^T a_i)}$. Maximizing the likelihood function is equivalent to minimizing the cross entropy loss function so we define the logistic regression (LR) problem as: $\min_{x\in R^d} L(x) = \sum_{i=1}^n -b_i \log(\mu_i) - (1-b_i)\log(1-\mu_i)$.

We now give a Newton-based method to solve this, which is also known as the IRLS method. The gradient and Hessian of $L$ are $g = \nabla_x L(x) = A^T(\mu - b)$ and $H = A^T SA$, respectively, where $\mu \in \mathbb{R}^n$ is the vector with entries as defined above and $S \in \mathbb{R}^{n\times n}$ is a diagonal matrix with $S_{ii} = \mu_i(1-\mu_i)$. The iterative updates in Newton's method are: $x_{t+1} = x_t - H_t^{-1}(g) = x_t + (A^T S_t A)^{-1}A^T(b-\mu_t)$, and so $x_{t+1} = (A^T S_t A)^{-1}A^T S_t(Ax_t + S_t^{-1}(b-\mu_t))$. Note that this equation corresponds to a weighted least squares problem instance and thus can be sped up using the sketching method (PLSMR) from section 3.1.

For logistic regression we compared solvers in *scikit-learn*, Python's machine learning library (namely, liblinear, lbfgs, and sag), with

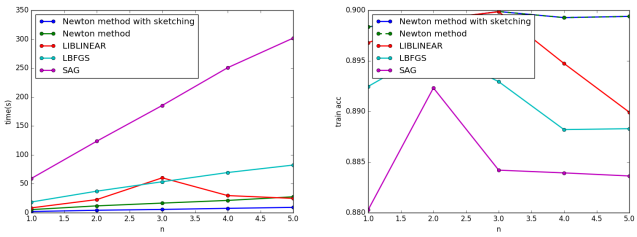



**Figure 8: Left: Runtime vs. input size (training + test) for logistic regression on synthetic datasets. Right: training accuracy for logistic regression on synthetic datasets.**

Newton-based IRLS methods with and without sketching. Synthetic datasets were generated in the following manner: we first generated a data matrix ($A$) with varying input size ($n \times d = n \cdot 1M \times 1K$), 1% non-zero density, and condition number $\kappa(A) = 1e6$. Then we generated $b^* = \text{sigmoid}(Ax) > 0.5$ ($x$ is a standard normal random variable), and randomly flipped 10% of the entries of $b^*$ to generate the response variable $b$. We used a random 80% of the data for training and 20% for testing. Regularization parameters were set to 0 in *scikit-learn* solvers. The left plot in Figure 8 gives the timing results and the right plot compares the training accuracy of the various methods. Figure 7 plots the test accuracy and the objective value. We observed that the sketching-based Newton method yields the best performance overall with respect to both time and accuracy.

The real datasets used for experiments are Susy, of size $5,000,000 \times 19$, and Covertype, of size $58192 \times 55$ from the UCI ML repository[2]. Tables 4 and 5 give the results. NM + LU, NM + LSMR and NM + PLSMR refer to using the Newton method with the weighted least squares problem solved using LU, LSMR and PLSMR, respectively.

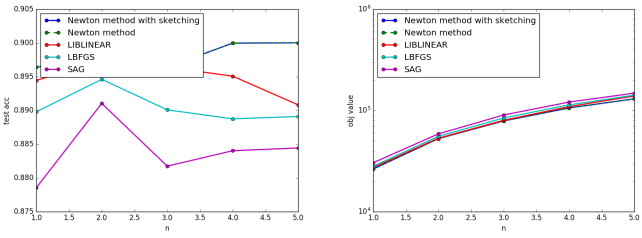



**Figure 7: Left: Test accuracy vs input size (training + test) for logistic regression on synthetic datasets. Right: Objective value vs. input size on synthetic datasets.**

We note that although the Newton method with PLSMR is preferred over a vanilla LSMR, using LU seems to be the best algorithm for WLS problems. We suspect this happens due to the small second dimension of the matrices: when $d$ is small, the overhead of computing the right preconditioner outweighs the time taken to form the normal equations.

**Table 4: Logistic Regression on SUSY**

| Method | Time | Train error | Test error | Objective value |
|---|---|---|---|---|
| Liblinear | 48.01s | 0.7883 | 0.7882 | 2048772.4 |
| LBFGS | 45.66s | 0.7883 | 0.7882 | 2048922.3 |
| SAG | 35.66s | 0.7883 | 0.7882 | 2048799.4 |
| Newton-cg | 89.96s | 0.7883 | 0.7882 | 2048861.3 |
| NM + LU | 20.07s | 0.7883 | 0.7882 | 1832415.9 |
| NM + LSMR | 30.64s | 0.7883 | 0.7882 | 1832415.9 |
| NM + PLSMR | 27.5s | 0.7883 | 0.7882 | 1832415.9 |

**Table 5: Logistic Regression on CoverType**

| Method | Time | Train error | Test error | Objective value |
|---|---|---|---|---|
| Liblinear | 7.95s | 0.7632 | 0.7649 | 230233.8 |
| LBFGS | 75.3s | 0.7608 | 0.7598 | 234650.1 |
| SAG | 417.5s | 0.7590 | 0.7563 | 237099.2 |
| Newton-cg | 383.7s | 0.7704 | 0.7723 | 4369998.5 |
| NM + LU | 3.8s | 0.7703 | 0.7722 | 224329.8 |
| NM + LSMR | 49.6s | 0.7703 | 0.7722 | 224329.6 |
| NM + PLSMR | 5.6s | 0.7705 | 0.7723 | 224328.5 |

### 3.4 Speeding Up Newton Updates

In this section we show that the IRLS approach is the same as Newton's method for certain types of optimization problems, and one can further apply a sketching-based solution to speed them up. We define the following optimization problem $\min_{x\in R^d} L(x) = \sum_{i=1}^{n} f(a_i^T x - b_i)$, where one can think of $f : R \to R$ as the loss function, where $f$ on a vector is defined element-wise, that is, $f_i(x) = f(x_i)$. Note that the gradient and Hessian of the loss function are $g = \nabla_x L(x) = A^T f'(Ax - b)$, and $H = A^T \operatorname{diag}(f''(Ax - b))A$, respectively. Letting $e = f'(Ax_t - b)$ and $S_t = \operatorname{diag}(f''(Ax_t - b))$, the iterative updates for Newton's method are $x_{t+1} = x_t - H_t^{-1}(g) = x_t + (A^T S_t A)^{-1} A^T e$, and so $x_{t+1} = (A^T S_t A)^{-1} A^T S_t (Ax_t + S_t^{-1} e)$, and so $x_{t+1} = \arg\min_x \|S_t^{1/2} Ax - (S_t^{1/2} Ax_t + S_t^{-1/2} e)\|_2$.

The update is nothing but a weighted least squares problem and one can use sketching-based solutions to speed it up when $A$ is over- or under-determined. The standard way of doing a Newton update is to find the Hessian and compute the descent direction as $H^{-1}g$, which naïvely takes $O(\text{nnz}(A)d + d^3)$ time. Using sketching-based preconditioning, one can reduce the complexity of the Hessian computation to $O(\text{nnz}(A) + d^3 + \sqrt{\kappa}\log\frac{1}{\epsilon}\,\text{nnz}(A))$. As preconditioning reduces the condition number to a small constant, one observes that $\sqrt{\kappa}\log\frac{1}{\epsilon}$ will be smaller compared to $d$ and the time taken by the sketching approach will be strictly lower than the standard way. We also note that if $A$ is ill-conditioned, then forming $A^T SA$ and inverting it introduces numerical instability. Therefore, sketching-based preconditioning not only speeds up the Newton updates but also provides numerical stability. We note that $\ell_p$ regression and M-estimation both fit into this framework.

## 4 LOW-RANK APPROXIMATION

In low rank approximation, given a rank-$r$ matrix $A \in \mathbb{R}^{m\times n}$, one seeks to find a rank-$k$ matrix $A''$ that minimizes $\|A - A'\|_p$. When $p = 2$ (spectral norm) or $p = F$ (Frobenius norm), the problem has a closed form solution, which can be obtained through the singular value decomposition of $A$. As in the case of regression, using the entrywise $\ell_1$ norm ($p = 1$) is more robust and less sensitive to outliers. Unfortunately for low rank approximation this norm not only does not have a closed form solution, but is NP-hard [10].

### 4.1 Frobenius Norm

Clarkson and Woodruff [5] developed a randomized approximation algorithm for $\ell_F$ low-rank approximation that achieves an $O(\text{nnz}(A) + (n + d)\text{poly}(k/\epsilon))$ running time, and achieves a $(1 + \epsilon)$-approximation factor. That is, given the optimal solution $A^*$, their algorithm outputs $A'$ such that: $\|A - A'\|_F \le (1 + \epsilon)\|A - A^*\|_F$.

**Algorithm 4** $\ell_F$ Low-Rank Approximation Algorithm.

1: Generate CountSketch $S$
2: Generate CountSketch $R$
3: Compute $SA$
4: Compute $AR$ and $SAR$
5: Solve $\min_{\text{rank}-k\,Y} \|Y - AR(SAR)^+ SAR\|_F^2$ exactly, using the SVD
6: Output $Y(SAR)^+ SA$

### 4.2 Entrywise $\ell_1$ Norm

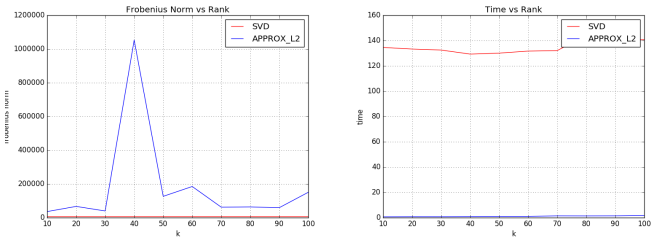
Song, Woodruff, and Zhong [20] recently proposed a randomized algorithm for entrywise $\ell_1$ norm low rank approximation, with an approximation factor of $\alpha = \log n \cdot \text{poly}(k)$ and running time $O(\text{nnz}(A) + m \cdot \text{poly}(k))$. This algorithm also heavily relies on sketching, where the underlying sketching matrices are combinations of CountSketch and Cauchy random matrices. Formally, if $A^*$ is a real minimizer, then the algorithm outputs $A'$ such that: $\|A - A'\|_1 \le \log m \cdot \text{poly}(k)\|A - A^*\|_1$.

**Algorithm 5** $\ell_1$ Low-Rank Approximation Algorithm

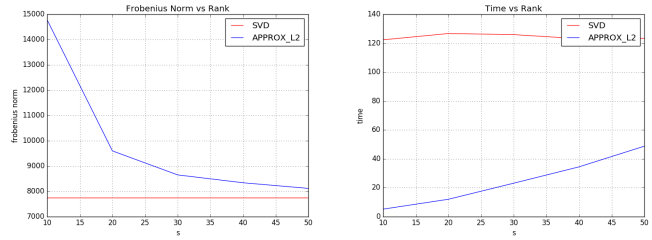
1: Generate Cauchy matrix $S$
2: Compute $SA$
3: Compute $C_{i,*} = \text{argmin}_x \|x^T SA - A_{i,*}\|_1$
4: Compute $B = C \cdot SA$
5: Generate Cauchy sketches $T_L, R, D, T_R$
6: Compute $SA$
7: Compute $T_L BR$, $DBT_R$ and $T_L BT_R$
8: Solve $\min_{X,Y} \|T_L BRXYDBT_R - T_L BT_R\|_F$ exactly
9: Output $BRX, YDB$

### 4.3 Experiments

We implemented both algorithms and evaluated them on synthetic and real-world datasets. We performed 2 different experiments on 3 synthetic datasets and 2 real-world datasets. In the first experiment, we vary the rank $k$ of the desired approximation, whereas in the

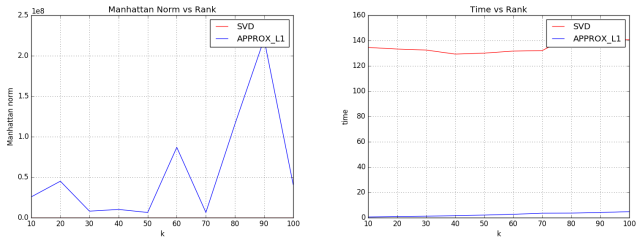



(a) $\|\|_F$ vs $k$ (b) Time vs $k$

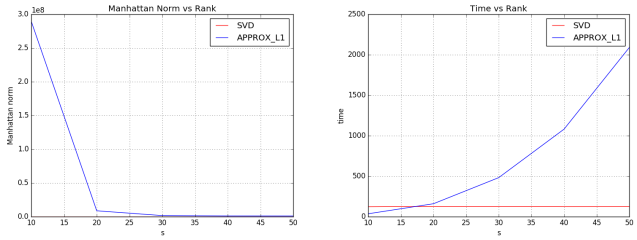



(c) $\|\|_F$ vs $c$ (d) Time vs $c$

**Figure 9: $\ell_F$-approximate algorithm on NIPS. Plots illustrate $\ell_F$ Norm & Time vs (i) rank $k$ and (ii) number of rows/columns $s = c \cdot \frac{k}{\epsilon^2}$ of sketching matrices $S$ and $R$. First experiment ((a), (b)): $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ and $c = 1$. Second experiment ((c),(d)): rank $k = 50$ and $c \in \{10, 20, 30, 40, 50\}$. Both experiments use $\epsilon = 0.1$.**




(a) $\|\|_F$ vs $k$ (b) Time vs $k$




(c) $\|\|_F$ vs $c$ (d) Time vs $c$

**Figure 10: $\ell_1$-approximate algorithm on NIPS. Plots illustrate $\ell_1$ Norm & Time vs (i) rank $k$ and (ii) number of rows/columns $s = c \cdot k \log k$ of sketching matrices $S$, $R$, $T_L$ and $T_R$. First experiment ((a),(b)): $k \in \{10, 20, 30, 40, 50\}$ and $c = 1$. Second experiment ((c),(d)): rank $k = 50$ and $c \in \{10, 20, 30, 40, 50\}$. Both experiments use $\epsilon = 0.1$.**

second experiment we fix a rank $k = 50$ and vary the number of rows/columns of the sketching matrices. For evaluating the approximation algorithm with respect to the $\ell_F$ norm we use $s = c \cdot \frac{k}{\epsilon}$ rows/columns of sketching matrices $S$ and $R$ with $\epsilon = 0.1$, whereas for evaluating the approximation algorithm with respect to the $\ell_1$ norm, $s = c \cdot k \log k$ is the small dimension for the sketching matrices $S$, $R$, $T_L$ and $T_R$.

*4.3.1 Synthetic Datasets.* We generated 3 sparse synthetic datasets: a matrix with $n = 10^5$ and $m = 10^3$ random i.i.d. normal variables, with $\|M\|_0 = \text{nnz}(M) = 0.50nm$ (sparse), $\|M\|_0 = \text{nnz}(M) = 0.50nm$ (semi-dense) and $|M\|_0 = \text{nnz}(M) = 0.75nm$ (dense) nonzero elements, respectively.

*4.3.2 Real Datasets.* As our first real-world dataset, we use the MovieLens 1M dataset [12], which contains 1 million ratings (0-5) from 6000 users on 4000 movies. As our second real-world dataset, we use the NIPS dataset [17], which contains the distribution of words in the full text of the NIPS conference papers published from 1987 to 2015. The dataset is in the form of a 11463 x 5812 matrix of word counts, containing 11463 words and 5811 NIPS conference papers. Each column contains the number of times each word appears in the corresponding document.

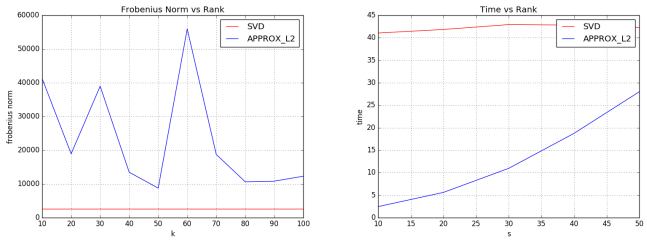
## 4.4 Discussion

The approximate-$\ell_F$ algorithm succeeds as we vary $k$. The norm $\|A - A_F\|_F = c\|A - A_k\|_F$, for $c$ a constant in the range [10,20]. The quality of the $\ell_F$-algorithm approximation factor improves and its running time increases as we increase the number $s$ of rows of the sketching matrices.

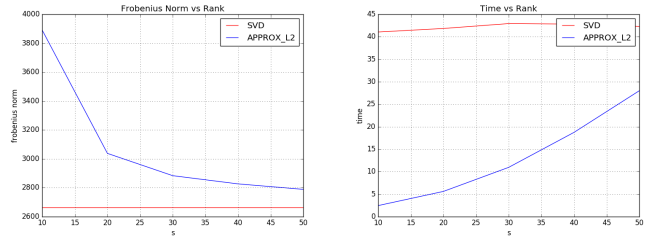
The approximate-$\ell_1$ algorithm succeeds as we vary $k$. The norm $\|A - A_1\|_1 = c\|A - A_k\|_F$, for $c$ a constant in the range $[10^5, 10^8]$. Observe that since computing the optimum $\|A - U^*V^*\|_1$ is NP-hard and requires the use of a polynomial system solver, we compare against the Frobenius norm $\|A - A_k\|_F$. We know that $\|A - U^*V^*\|_1 \leq \sqrt{n \cdot m}\|A - U^*V^*\|_F$. Also the $\ell_1$-approximate algorithm does not have a constant factor approximation but rather a $\log m \cdot k \cdot \text{poly}(\log k)$ factor approximation. As expected, the running time of the approximate-$\ell_1$ algorithm increases as we increase the number $s$ of rows of the sketching matrices.

## 5 CONCLUSION

In our work we provided a comprehensive empirical study of the performance of sketching for various regression and low rank approximation problems. We showed that sketching-based preconditioning leads to extremely well-conditioned systems which then lead to state-of-the-art performance on the various problems we considered. We also showed theoretical algorithms for Frobenius and robust low rank approximation perform well in practice. We hope this study will push the use of recent theoretical sketching advances more to practice. Several avenues for future study include the use of CountSketch to speed up least squares regression inside of interior point and convex optimization methods. Although several sketching-based implementations have been done for convex

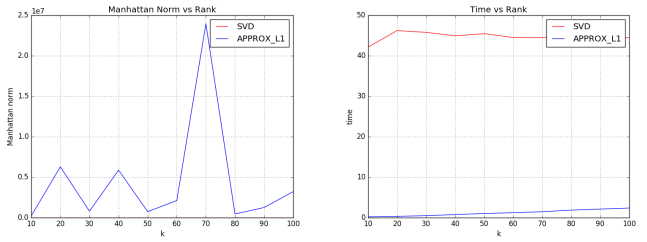



(a) $\|\|_F$ vs $k$ (b) Time vs $k$

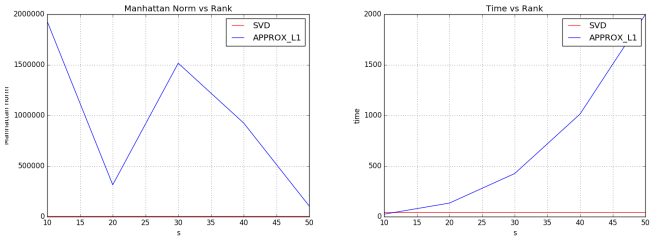



(c) $\|\|_F$ vs $c$ (d) Time vs $c$

**Figure 11: $\ell_F$-approximate algorithm on MovieLens. Plots illustrate $\ell_F$ Norm & Time vs (i) rank $k$ and (ii) number of rows/columns $s = c \cdot \frac{k}{\epsilon^2}$ of sketching matrices $S$ and $R$. First experiment ((a), (b)): $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ and $c = 1$. Second experiment ((c),(d)): rank $k = 50$ and $c \in \{10, 20, 30, 40, 50\}$. Both experiments use $\epsilon = 0.1$.**




(a) $\|\|_F$ vs $k$ (b) Time vs $k$




(c) $\|\|_F$ vs $c$ (d) Time vs $c$

**Figure 12: $\ell_1$ algorithm on MovieLens. Plots illustrate $\ell_1$ Norm & Time vs (i) rank $k$ and (ii) small dimension $s = c \cdot k \log k$ of Sketching matrices $S$, $R$, $T_L$ and $T_R$. First experiment ((a),(b)): $k \in \{10, 20, 30, 40, 50\}$ and $c = 1$. Second experiment ((c),(d)): rank $k = 50$ and $c \in \{10, 20, 30, 40, 50\}$. Both experiments use $\epsilon = 0.1$.**

optimization [18], sparse sketching matrices seem to not have been studied in that context.

**Acknowledgments:** Dimitris Konomis and David P. Woodruff are supported in part by an Office of Naval Research (ONR) grant N00014-18-1-2562. Dimitris Konomis is in part supported from the “Alexander S. Onassis" Foundation.

## REFERENCES

[1] Robert Andersen. 2008. *Modern methods for robust regression.* Number 152. Sage.
[2] Arthur Asuncion and David Newman. 2007. UCI machine learning repository. (2007).
[3] Haim Avron, Petar Maymounkov, and Sivan Toledo. 2010. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM Journal on Scientific Computing* 32, 3 (2010), 1217–1236.
[4] Jean Bourgain, Sjoerd Dirksen, and Jelani Nelson. 2015. Toward a Unified Theory of Sparse Dimensionality Reduction in Euclidean Space. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015.* 499–508.
[5] Kenneth L Clarkson and David P Woodruff. 2013. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing.* ACM, 81–90.
[6] ES Coakley, Vladimir Rokhlin, and Mark Tygert. 2011. A fast randomized algorithm for orthogonal projection. *SIAM Journal on Scientific Computing* 33, 2 (2011), 849–868.
[7] Michael B. Cohen. 2016. Nearly Tight Oblivious Subspace Embeddings by Trace Inequalities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016.* 278–287.
[8] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1.
[9] David Chin-Lung Fong and Michael Saunders. 2011. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing* 33, 5 (2011), 2950–2971.
[10] Nicholas Gillis and Stephen G. Vavasis. 2015. On the complexity of robust pca and $\ell_1$-norm low-rank matrix approximation. *arXiv preprint arXiv:1711.09883* (2015).
[11] Gene H Golub and Charles F Van Loan. 2012. *Matrix computations.* Vol. 3. JHU Press.
[12] F. Maxwell Harper and Joseph.A Konstan. 2015. The MovieLens Datasets: History and Context.. In *ACM Transactions on Interactive Intelligent Systems (TiiS).* ACM.
[13] Yi Li, Huy L. Nguyen, and David P. Woodruff. 2014. On Sketching Matrix Norms and the Top Singular Vector. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014.* 1562–1581.
[14] Xiangrui Meng and Michael W. Mahoney. 2013. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013.* 91–100.
[15] Xiangrui Meng, Michael A Saunders, and Michael W Mahoney. 2014. LSRN: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing* 36, 2 (2014), C95–C118.
[16] Jelani Nelson and Huy L. Nguyen. 2013. OSNAP: Faster Numerical Linear Algebra Algorithms via Sparser Subspace Embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA.* 117–126.
[17] V. Perrone, P.A. Jenkins, D Spano, and Y.W. Teh. 2013. Poisson Random Fields for Dynamic Feature Models. (2013).
[18] M. Pilanci and M. J. Wainwright. 2014. Randomized Sketches of Convex Programs with Sharp Guarantees. *ArXiv e-prints* (April 2014). arXiv:cs.IT/1404.7203
[19] Tamas Sarlos. 2006. Improved approximation algorithms for large matrices via random projections. In *FOCS.* 143–152.
[20] Zhao Song, David Woodruff, and Peilin Zhong. 2017. Low rank approximation with entrywise $\ell_1$-norm error. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing.* ACM, 688–701.
[21] David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.