Robust Regression on MapReduce

Xiangrui Meng XIMENG@LINKEDIN.COM
LinkedIn Corporation, 2029 Stierlin Ct, Mountain View, CA 94043

Michael W. Mahoney MMAHONEY@CS.STANFORD.EDU
Department of Mathematics, Stanford University, Stanford, CA 94305

Abstract

Although the MapReduce framework is now the *de facto* standard for analyzing massive data sets, many algorithms (in particular, many iterative algorithms popular in machine learning, optimization, and linear algebra) are hard to fit into MapReduce. Consider, *e.g.*, the ℓ_p regression problem: given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, find a vector $x^* \in \mathbb{R}^n$ that minimizes $f(x) = \|Ax - b\|_p$. The widely-used ℓ_2 regression, *i.e.*, linear least-squares, is known to be highly sensitive to outliers; and choosing $p \in [1, 2)$ can help improve robustness. In this work, we propose an efficient algorithm for solving strongly over-determined ($m \gg n$) robust ℓ_p regression problems to moderate precision on MapReduce. Our empirical results on data up to the terabyte scale demonstrate that our algorithm is a significant improvement over traditional iterative algorithms on MapReduce for ℓ_1 regression, even for a fairly small number of iterations. In addition, our proposed interior-point cutting-plane method can also be extended to solving more general convex problems on MapReduce.

1 Introduction

Statistical analysis of massive data sets presents very substantial challenges both to data infrastructure and to algorithm development. In particular, many popular data analysis and machine learning algorithms that perform well when applied to small-scale and medium-scale data that can be stored in RAM are infeasible when applied to the terabyte-scale and petabyte-scale data sets that are stored in distributed environments and that are increasingly common. In this paper, we develop algorithms for variants of the robust regression problem, and we evaluate implementations of them on data of up to the terabyte scale. In addition to being of interest since ours are the first algorithms for these problems that are appropriate for data of that scale, our results are also of interest since they highlight "algorithm engineering" challenges that will become more common as researchers try to scale up small-scale and medium-scale data analysis and machine learning methods. For example, at several points we had to work with variants of more primitive algorithms that were *worse* by traditional complexity measures but that had better communication properties.

1.1 MapReduce and Large-scale Data

The MapReduce framework, introduced by (Dean & Ghemawat, 2004) in 2004, has emerged as the *de facto* standard parallel environment for analyzing massive data sets. Apache Hadoop [1], an open source software framework inspired by Google's MapReduce, is now extensively used by companies such as Facebook, LinkedIn, Yahoo!, etc. In a typical application, one builds clusters of thousands of nodes containing petabytes of storage in order to process terabytes or even petabytes of daily data. As a parallel computing framework, MapReduce is well-known for its scalability to massive data. However, the scalability comes at the price of a very restrictive interface: sequential access to data, and functions limited to map and reduce. Working within this framework demands that traditional algorithms be redesigned to respect this interface. For example, the Apache Mahout [2] project is building a collection of scalable machine learning algorithms that includes algorithms for collaborative filtering, clustering, matrix decomposition, etc.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1] Apache Hadoop, http://hadoop.apache.org/
[2] Apache Mahout, http://mahout.apache.org/

Although some algorithms are easily adapted to the MapReduce framework, many algorithms (and in particular many iterative algorithms popular in machine learning, optimization, and linear algebra) are not. When the data are stored on RAM, each iteration is usually very cheap in terms of floating-point operations (FLOPs). However, when the data are stored on secondary storage, or in a distributed environment, each iteration requires at least one pass over the data. Since the cost of communication to and from secondary storage often dominates FLOP count costs, each pass can become very expensive for very large-scale problems. Moreover, there is generally no parallelism between iterations: an iterative algorithm must wait until the previous step gets completed before the next step can begin.

1.2 Our Main Results

In this work, we are interested in developing algorithms for robust regression problems on MapReduce. Of greatest interest will be algorithms for the strongly over-determined ℓ_1 regression problem,[3] although our method will extend to more general ℓ_p regression. For simplicity of presentation, we will formulate most of our discussion in terms of ℓ_p regression; and toward the end we will describe the results of our implementation of our algorithm for ℓ_1 regression.

Recall the *strongly over-determined ℓ_p regression problem*: given a matrix $A \in \mathbb{R}^{m \times n}$, with $m \gg n$, a vector $b \in \mathbb{R}^m$, a number $p \in [1, \infty)$, and an error parameter $\epsilon > 0$, find a $(1+\epsilon)$-approximate solution $\hat{x} \in \mathbb{R}^n$ to:

$$f^* = \min_{x \in \mathbb{R}^n} \|Ax - b\|_p, \tag{1}$$

i.e., find a vector $\hat{x}$ such that

$$\|A\hat{x} - b\|_p \leq (1+\epsilon) f^*, \tag{2}$$

where the ℓ_p norm is given by $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$. A more robust alternative to the widely-used ℓ_2 regression is obtained by working with ℓ_p regression, with $p \in [1, 2)$, where $p = 1$ is by far the most popular alternative. This, however, comes at the cost of increased complexity. While ℓ_2 regression can be solved with, *e.g.*, a QR decomposition, ℓ_1 regression problems can be formulated as linear programs, and other ℓ_p regression problems can be formulated as convex programs. In those cases, iterated weighted least-squares methods or simplex methods or interior-point methods are typically used in practice. These algorithms tend to require dot products, orthogonalization, and thus a great deal of communication, rendering them challenging to implement in the MapReduce framework.

In this paper, we describe an algorithm with better communication properties that is efficient for solving strongly over-determined ℓ_p regression problems to moderate precision on MapReduce.[4] Several aspects of our main algorithm are of particular interest:

- Single-pass conditioning. We use a recently-developed fast rounding algorithm (which takes $\mathcal{O}(mn^3 \log m)$ time to construct a $2n$-rounding of a centrally symmetric convex set in $\mathbb{R}^n$ (Clarkson et al., 2013)) to construct a single-pass deterministic conditioning algorithm for ℓ_p regression.
- Single-pass random sampling. By using a constrained form of ℓ_p regression (that was also used recently (Clarkson et al., 2013)), we show that the method of subspace-preserving random sampling (Dasgupta et al., 2009) can be (easily) implemented in the MapReduce framework, *i.e.*, with map and reduce functions, in a single pass.
- Effective initialization. By using multiple subsampled solutions from the single-pass random sampling , we can construct a small initial search region for interior-point cutting-plane methods.
- Effective iterative solving. By performing in parallel multiple queries at each iteration, we develop a randomized IPCPM (interior-point cutting plane method) for solving the convex ℓ_p regression program.

In addition to describing the basic algorithm, we also present empirical results from a numerical implementation of our algorithm applied to the ℓ_1 regression problems on data sets of size up to the terabyte scale.

1.3 Prior Related Work

There is a large literature on robust regression, distributed computation, MapReduce, and randomized matrix algorithms that is beyond our scope to review. See, *e.g.*, (Rousseeuw & Leroy, 1987), (Bertsekas & Tsitsiklis, 1991), (Dean & Ghemawat, 2004), and (Mahoney, 2011), respectively, for details. Here, we will review only the most recent related work.

Strongly over-determined ℓ_1 regression problems were considered by (Portnoy & Koenker, 1997), who used a uniformly-subsampled solution for ℓ_1 regression to estimate the signs of the optimal residuals in order

[3] The ℓ_1 regression problem is also known as the Least Absolute Deviations or Least Absolute Errors problem.

[4] Interestingly, both our single-pass conditioning algorithm as well as our iterative procedure are *worse* in terms of FLOP counts than state-of-the-art algorithms (developed for RAM) for these problems—see Tables 1 and 2—but we prefer them since they perform better in very large-scale distributed settings that are of interest to us here.

to reduce the problem size; their sample size is proportional to $(mn)^{2/3}$. (Clarkson, 2005) showed that, with proper conditioning, relative-error approximate solutions can be obtained from row norm-based sampling; and (Dasgupta et al., 2009) extended these subspace-preserving sampling schemes to ℓ_p regression, for $p \in [1, \infty)$, thereby obtaining relative-error approximations. (Sohler & Woodruff, 2011) proved that a Cauchy Transform can be used for ℓ_1 conditioning and thus ℓ_1 regression in $\mathcal{O}(mn^2 \log n)$ time; this was improved to $\mathcal{O}(mn \log n)$ time with the Fast Cauchy Transform by (Clarkson et al., 2013), who also developed an ellipsoidal rounding algorithm (see Lemma 1 below) and used it and a fast random projection to construct a fast single-pass conditioning algorithm (upon which our Algorithm 1 below is based). (Clarkson & Woodruff, 2013) and (Meng & Mahoney, 2013) show that both ℓ_2 regression and ℓ_p regression can be solved in input-sparsity time via subspace-preserving sampling. The large body of work on fast randomized algorithms for ℓ_2 regression (and related) problems has been reviewed recently (Mahoney, 2011).

To obtain a $(1+\epsilon)$-approximate solution in relative scale, the sample sizes required by all these algorithms are all proportional to $1/\epsilon^2$, which limits sampling algorithms to "low precision", *e.g.*, $\epsilon \approx 10^{-2}$, solutions. By using the output of the sampling/projection step as a preconditioner for a traditional iterative method, thereby leading to an $\mathcal{O}(\log(1/\epsilon))$ dependence, this problem has been overcome for ℓ_2 regression (Mahoney, 2011). For ℓ_1 regression, the $\mathcal{O}(1/\epsilon^2)$ convergence rate of the subgradient method of (Clarkson, 2005) was improved by (Nesterov, 2009), who showed that, with a smoothing technique, the number of iterations can be reduced to $\mathcal{O}(1/\epsilon)$. Interestingly (and as we will return to in Section 3.3), the rarely-used ellipsoid method (see (Grötschel et al., 1981)) as well as IPCPMs (see (Mitchell, 2003)) can solve general convex problems and converge in $\mathcal{O}(\log(1/\epsilon))$ iterations—with extra poly(n) work per iteration.

More generally, there has been a lot of interest recently in distributed machine learning computations. For example, (Daumé et al., 2012) describes efficient protocols for distributed classification and optimization; (Balcan et al., 2012) analyzes communication complexity and privacy aspects of distributed learning; (Mackey et al., 2011) adopts a divide-and-conquer approach to matrix factorization such as CUR decompositions; and (Zhang et al., 2012) develop communication-efficient algorithms for statistical optimization. Algorithms for these and other problems can be analyzed in models for MapReduce (Karloff et al., 2010; Goodrich, 2010; Feldman et al., 2010); and work on parallel and distributed approaches to scaling up machine learning has been reviewed recently (Bekkerman et al., 2011).

2 Background and Overview

In the remainder of this paper, we use the following formulation of the ℓ_p regression problem:

$$\begin{array}{ll} \text{minimize}_{x \in \mathbb{R}^n} & \|Ax\|_p \\ \text{subject to} & c^T x = 1. \end{array} \tag{3}$$

This formulation of ℓ_p regression, which consists of a homogeneous objective and an affine constraint, can be shown to be equivalent to the formulation of (1).[5] Denote the feasible region by $\Omega = \{x \in \mathbb{R}^n \,|\, c^T x = 1\}$, where recall that we are interested in the case when $m \gg n$. Let $\mathcal{X}^*$ be the set of all optimal solutions to (3) and x^* be an arbitrary optimal solution. Then, let $f(x) = \|Ax\|_p$, $f^* = \|Ax^*\|_p$, and let

$$g(x) = A^T [Ax]^{p-1} / \|Ax\|_p^{p-1} \in \partial f(x),$$

where $([Ax]^{p-1})_i = \text{sign}(a_i^T x)|a_i^T x|^{p-1}$ and a_i is the i-th row of A, $i = 1, \ldots, m$. Note that $g(x)^T x = f(x)$. For simplicity, we assume that A has full column rank and $c \neq \mathbf{0}$. Our assumptions imply that $\mathcal{X}^*$ is a nonempty and bounded convex set and $f^* > 0$. Thus, given an $\epsilon > 0$, our goal is to find an $\hat{x} \in \Omega$ that is a $(1+\epsilon)$-approximate solution to (3) in relative scale, *i.e.*, such that $f(\hat{x}) < (1+\epsilon) f^*$.

As with ℓ_2 regression, ℓ_p regression problems are easier to solve when they are well-conditioned. The ℓ_p-norm condition number of A, denoted $\kappa_p(A)$, is defined as:

$$\kappa_p(A) = \sigma_p^{\max}(A) / \sigma_p^{\min}(A),$$

where

$$\sigma_p^{\max}(A) = \max_{\|x\|_2 \leq 1} \|Ax\|_p \text{ and } \sigma_p^{\min}(A) = \min_{\|x\|_2 \geq 1} \|Ax\|_p.$$

This implies

$$\sigma_p^{\min}(A) \|x\|_2 \leq \|Ax\|_p \leq \sigma_p^{\max}(A) \|x\|_2, \quad \forall x \in \mathbb{R}^n.$$

We use κ_p, $\sigma_p^{\min}$, and $\sigma_p^{\max}$ for simplicity when the underlying matrix is clear. The element-wise ℓ_p-norm of A is denoted by $\|A\|_p$. We use $\mathcal{E}(d, E) = \{x \in \mathbb{R}^n \,|\, x = d + Ez, \ \|z\|_2 = 1\}$ to describe an ellipsoid where $E \in \mathbb{R}^{n \times n}$ is a non-singular matrix. The volume of a full-dimensional ellipsoid $\mathcal{E}$ is denoted by $|\mathcal{E}|$. We

[5] In particular, the "new" A is A concatenated with $-b$, etc. Note that the same formulation is also used by (Nesterov, 2009) for solving unconstrained convex problems in relative scale as well as by (Clarkson et al., 2013).

use $\mathcal{S}(S,t)=\{x\in\mathbb{R}^n \mid Sx\le t\}$ to describe a polytope, where $S\in\mathbb{R}^{s\times n}$ and $t\in\mathbb{R}^s$ for some $s\ge n+1$.

Given an ℓ_p regression problem, its condition number is generally unknown and can be arbitrarily large; and thus one needs to run a conditioning algorithm before randomly sampling and iteratively solving. Given any non-singular matrix $E\in\mathbb{R}^n$, let y^* be an optimal solution to the following problem:

$$\begin{array}{ll}\text{minimize}_{y\in\mathbb{R}^n} & \|AEy\|_p \\ \text{subject to} & c^TEy=1.\end{array} \tag{4}$$

This problem is equivalent to (3), in that we have $x^*=Ey^*\in\mathcal{X}^*$, but the condition number associated with (4) is $\kappa_p(AE)$, instead of $\kappa_p(A)$. So, the conditioning algorithm amounts to finding a non-singular matrix $E\in\mathbb{R}^n$ such that $\kappa_p(AE)$ is small. One approach to conditioning is via ellipsoidal rounding. In this paper, we will modify the following result from (Clarkson et al., 2013) to compute a fast ellipsoidal rounding.

Lemma 1 ((Clarkson et al., 2013)). *Given $A\in\mathbb{R}^{m\times n}$ with full column rank and $p\in[1,2)$, it takes at most $\mathcal{O}(mn^3\log m)$ time to find a non-singular matrix $E\in\mathbb{R}^{n\times n}$ such that*

$$\|y\|_2\le\|AEy\|_p\le 2n\|y\|_2,\quad\forall y\in\mathbb{R}^n.$$

Finally, we call a work *online* if it is executed on MapReduce, and *offline* otherwise. An online work deals with large-scale data stored on secondary storage but the work can be well distributed on MapReduce; an offline work deals with data stored on RAM.

3 ℓ_p Regression on MapReduce

In this section, we will describe our main algorithm for ℓ_p regression on MapReduce.

3.1 Single-pass Conditioning Algorithm

The algorithm of Lemma 1 for computing a $2n$-rounding is *not* immediately-applicable to large-scale ℓ_p regression problems, since each call to the oracle requires a pass to the data.[6] We can group n calls together within a single pass, but we would still need $\mathcal{O}(n\log m)$ passes. Here, we present a *deterministic* single-pass conditioning algorithm that balances the cost-performance trade-off to provide a $2n^{2/p}$-conditioning of A. See Algorithm 1. Our main result for Algorithm 1 is given in the following lemma.

Lemma 2. *Algorithm 1 is a $2n^{2/p}$-conditioning algorithm and it runs in $\mathcal{O}((mn^2+n^4)\log m)$ time.*

[6]The algorithm takes a centrally-symmetric convex set described by a separation oracle that is a subgradient of $\|Ax\|_p$; see (Clarkson et al., 2013) for details.

Algorithm 1 A single-pass conditioning algorithm.

Input: $A\in\mathbb{R}^{m\times n}$ with full column rank & $p\in[1,2)$.
Output: A non-singular matrix $E\in\mathbb{R}^{n\times n}$ such that

$$\|y\|_2\le\|AEy\|_p\le 2n^{2/p}\|y\|_2,\ \forall y\in\mathbb{R}^n.$$

1: Partition A along its rows into sub-matrices of size $n^2\times n$, denoted by $A_1,\dots,A_M$.
2: For each A_i, compute its economy-sized singular value decomposition (SVD): $A_i=U_i\Sigma_iV_i^T$.
3: Let $\tilde{A}_i=\Sigma_iV_i^T$ for $i=1,\dots,M$,
$$\tilde{\mathcal{C}}=\{x\mid(\sum_{i=1}^M\|\tilde{A}_ix\|_2^p)^{1/p}\le 1\},\text{ and }\tilde{A}=\begin{pmatrix}\tilde{A}_1\\ \vdots\\ \tilde{A}_M\end{pmatrix}.$$
4: Compute $\tilde{A}$'s SVD: $\tilde{A}=\tilde{U}\tilde{\Sigma}\tilde{V}^T$.
5: Let $\mathcal{E}_0=\mathcal{E}(0,E_0)$ where $E_0=n^{1/p-1/2}\tilde{V}\tilde{\Sigma}^{-1}$. $\mathcal{E}_0$ gives an $(Mn^2)^{1/p-1/2}$-rounding of $\tilde{\mathcal{C}}$.
6: With the algorithm of Lemma 1, compute an ellipsoid $\mathcal{E}=\mathcal{E}(0,E)$ that gives a $2n$-rounding of $\tilde{\mathcal{C}}$.
7: Return E.

Proof. The idea is to use block-wise reduction in ℓ_2-norm and apply fast rounding to a small problem. The tool we need is simply the equivalence of vector norms. Let $\mathcal{C}=\{x\in\mathbb{R}^n\mid\|Ax\|_p\le 1\}$, which is convex, full-dimensional, bounded, and centrally symmetric. Adopting notation from Algorithm 1, we first have

$$n^{1-2/p}\tilde{\mathcal{C}}\subseteq\mathcal{C}\subseteq\tilde{\mathcal{C}}$$

because for all $x\in\mathbb{R}^n$,

$$\|Ax\|_p^p=\sum_{i=1}^M\|A_ix\|_p^p\le n^{2-p}\sum_{i=1}^M\|A_ix\|_2^p=n^{2-p}\sum_{i=1}^M\|\tilde{A}_ix\|_2^p$$

and

$$\|Ax\|_p^p=\sum_{i=1}^M\|A_ix\|_p^p\ge\sum_{i=1}^M\|A_ix\|_2^p=\sum_{i=1}^M\|\tilde{A}_ix\|_2^p.$$

Next we prove that $\mathcal{E}_0$ gives an $(Mn^2)^{1/p-1/2}$-rounding of $\tilde{C}$. For all $x\in\mathbb{R}^n$, we have

$$\begin{aligned}\sum_{i=1}^M\|\tilde{A}_ix\|_2^p&\le\sum_{i=1}^M\|\tilde{A}_ix\|_p^p=\|\tilde{A}x\|_p^p\le(Mn)^{1-p/2}\|\tilde{A}x\|_2^p\\&=(Mn)^{1-p/2}\|\tilde{\Sigma}\tilde{V}^Tx\|_2^p,\end{aligned}$$

and

$$\begin{aligned}\sum_{i=1}^M\|\tilde{A}_ix\|_2^p&\ge n^{p/2-1}\sum_{i=1}^M\|\tilde{A}_ix\|_p^p=n^{p/2-1}\|\tilde{A}x\|_p^p\\&\ge n^{p/2-1}\|\tilde{A}x\|_2^p=n^{p/2-1}\|\tilde{\Sigma}\tilde{V}^Tx\|_2^p.\end{aligned}$$

Then by choosing $E_0=n^{1/p-1/2}\tilde{V}\tilde{\Sigma}^{-1}$, we get

$$\|E_0^{-1}x\|_2\le(\sum_{i=1}^M\|\tilde{A}_ix\|_2^p)^{1/p}\le(Mn^2)^{1/p-1/2}\|E_0^{-1}x\|_2$$

	time	κ_1
(Clarkson, 2005)	$\mathcal{O}(mn^5 \log m)$	$(n(n+1))^{1/2}$
Lemma 1	$\mathcal{O}(mn^3 \log m)$	$2n$
Lemma 2 & Algorithm 1	$\mathcal{O}(mn^2 \log m)$	$2n^2$
(Sohler & Woodruff, 2011)	$\mathcal{O}(mn^2 \log n)$	$\mathcal{O}(n^{3/2} \log^{3/2} n)$
(Clarkson et al., 2013)	$\mathcal{O}(mn \log m)$	$\mathcal{O}(n^{5/2} \log^{1/2} n)$
(Clarkson et al., 2013)	$\mathcal{O}(mn \log n)$	$\mathcal{O}(n^{5/2} \log^{5/2} n)$
(Meng & Mahoney, 2013)	$\mathcal{O}(\text{nnz}(A))$	$\mathcal{O}(n^3 \log^3 n)$

Table 1. Comparison of ℓ_1-norm conditioning algorithms on the running time and conditioning quality.

for all $x \in \mathbb{R}^n$ and hence $\mathcal{E}_0$ gives an $(Mn^2)^{1/p-1/2}$-rounding of $\tilde{\mathcal{C}}$. Since $n^{1-2/p}\tilde{\mathcal{C}} \subseteq \mathcal{C} \subseteq \tilde{\mathcal{C}}$, we know that any $2n$-rounding of $\tilde{\mathcal{C}}$ is a $2n \cdot n^{2/p-1} = 2n^{2/p}$-rounding of $\mathcal{C}$. Therefore, Algorithm 1 computes a $2n^{2/p}$-conditioning of A. Note that the rounding procedure is applied to a problem of size $Mn \times n \approx m/n \times n$. Therefore, Algorithm 1 only needs a single pass through the data, with $\mathcal{O}(mn^2)$ FLOPs and an offline work of $\mathcal{O}((mn^2 + n^4) \log m)$ FLOPs. The offline work requires m RAM, which might be too much for large-scale problems. In such cases, we can increase the block size from n^2 to, for example, n^3. This gives us a $2n^{3/p-1/2}$-conditioning algorithm that only needs m/n offline RAM and $\mathcal{O}((mn + n^4) \log m)$ offline FLOPs. The proof follows similar arguments. □

See Table 1 for a comparison of the results of Algorithm 1 and Lemma 2 with prior work on ℓ_1 norm conditioning (and note that some of these results, *e.g.*, those of (Clarkson et al., 2013) and (Meng & Mahoney, 2013), have extensions that apply to ℓ_p-norm conditioning). Although the Cauchy Transform (Sohler & Woodruff, 2011) and the Fast Cauchy Transform (Clarkson et al., 2013) are independent of A and require little offline work, there are several concerns with using them in our application. First, the constants hidden in κ_1 are not explicitly given, and they may be too large for practical use, especially when n is small. Second, although random sampling algorithms do not require $\sigma_p^{\min}$ and $\sigma_p^{\max}$ as inputs, some algorithms, *e.g.*, IPCPMs, need accurate bounds of them. Third, these transforms are randomized algorithms that fail with certain probability. Although we can repeat trials to make the failure rate arbitrarily small, we don't have a simple way to check whether or not any given trial succeeds. Finally, although the online work in Algorithm 1 remains $\mathcal{O}(mn^2)$, it is embarrassingly parallel and can be well distributed on MapReduce. For large-scale strongly over-determined problems, Algorithm 1 with block size n^3 seems to be a good compromise in practice. This guarantees $2n^{3/p-1/2}$-conditioning, and the $\mathcal{O}(mn^2)$ online work can be easily distributed on MapReduce.

3.2 Single-pass Random Sampling

Here, we describe our method for implementing the subspace-sampling procedure with map and reduce functions. Suppose that after conditioning we have $\sigma_p^{\min}(A) = 1$ and $\kappa_p(A) = \text{poly}(n)$. (Here, we use A instead of AE for simplicity.) Then the following method of (Dasgupta et al., 2009) can be used to perform subspace-preserving sampling.

Lemma 3 ((Dasgupta et al., 2009)). *Given $A \in \mathbb{R}^{m \times n}$ that is (α, β, p)-conditioned[7] and an error parameter $\epsilon < \frac{1}{7}$, let $r \geq 16(2^p+2)(\alpha\beta)^p(n \log \frac{12}{\epsilon} + \log \frac{2}{\delta})/(p^2\epsilon^2)$, and let $S \in \mathbb{R}^{m \times m}$ be a diagonal "sampling matrix," with random entries:*

$$S_{ii} = \begin{cases} \frac{1}{p_i} & \text{with probability } p_i, \\ 0 & \text{otherwise,} \end{cases}$$

where the importance sampling probabilities

$$p_i \geq \min\left\{1, \frac{\|a_i\|_p^p}{\|A\|_p^p} \cdot r\right\}, \quad i = 1, \ldots, m.$$

Then, with probability at least $1-\delta$, the following holds for all $x \in \mathbb{R}^n$,

$$(1-\epsilon)\|Ax\|_p \leq \|SAx\|_p \leq (1+\epsilon)\|Ax\|_p. \qquad (5)$$

This subspace-preserving sampling lemma can be used, with the formulation (3), to obtain a relative-error approximation to the ℓ_p regression problem, the proof of which is immediate.

Lemma 4 ((Clarkson et al., 2013)). *Let S be constructed as in Lemma 3, and let $\hat{x}$ be the optimal solution to the subsampled problem:*

$$\begin{array}{ll} \textit{minimize}_{x \in \mathbb{R}^n} & \|SAx\|_p \\ \textit{subject to} & c^T x = 1. \end{array}$$

Then with probability at least $1-\delta$, $\hat{x}$ is a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to (3).

It is straightforward to implement this algorithm in MapReduce in a single pass. This is presented in Algorithm 2. Importantly, note that more than one subsampled solution can be obtained in a single pass. This translates to a higher precision or a lower failure rate; and, as described in Section 3.3, it can also be used to construct a better initialization.

Several practical points are worth noting. First, $n\kappa_p^p$ is an upper bound of $\|A\|_p^p$, which makes the actual sample size likely to be smaller than r. For better control on the sample size, we can compute $\|A\|_p^p$ directly via one pass over A prior to sampling, or we can set a

[7] See (Clarkson et al., 2013) for the relationship between $\kappa_p(A)$ and the notion of (α, β, p)-conditioning.

Algorithm 2 A single-pass sampling algorithm.

Input: $A \in \mathbb{R}^{m\times n}$ with $\sigma_p^{\max}(A) = \kappa_p$, $c \in \mathbb{R}^n$, a desired sample size r, and an integer N.
Output: N approximate solutions: $\hat{x}_k,\ k = 1, \ldots, N$.

function MAPPER(a: a row of A)
  Let $p = \min\{r\|a\|_p^p/(n\kappa_p^p), 1\}$.
  Emit $(k, a/p)$ with probability p, $k = 1, \ldots, N$.
end function
function REDUCER(k, $\{a_i\}$)
  Assemble A_k from $\{a_i\}$.
  Compute $\hat{x}_k = \arg\min_{c^T x=1} \|A_k x\|_p$.
  Emit $(k, \hat{x}_k)$.
end function

big r in mappers and discard rows at random in reducers if the actual sample size is too big. Second, in practice, it is hard to accept ϵ as an input and determine the sample size r based on Lemma 3. Instead, we choose r directly based on our hardware capacity and running time requirements. For example, suppose we use a standard primal-dual path-following algorithm (see (Nesterov & Nemirovsky, 1994)) to solve subsampled problems. Then, since each problem needs $\mathcal{O}(rn)$ RAM and $\mathcal{O}(r^{3/2}n^2 \log \frac{r}{\epsilon})$ running time for a $(1+\epsilon)$-approximate solution, where r is the sample size, this should dictate the choice of r. Similar considerations apply to the use of the ellipsoid method or IPCPMs.

3.3 A Randomized IPCPM Algorithm

A problem with a vanilla application of the subspace-preserving random sampling algorithm is accuracy: it is very efficient if we only need one or two accurate digits (see (Clarkson et al., 2013) for details), but if we are looking for "moderate-precision" solutions, *e.g.*, those with $\epsilon \approx 10^{-5}$, then we very quickly be limited by the $\mathcal{O}(1/\epsilon^2)$ sample size required by Lemma 3. For example, setting $p = 1$, $n = 10$, $\alpha\beta = 1000$, and $\epsilon = 10^{-3}$ into Lemma 3, we get a sample size of approximately 10^{12}, which as a practical matter is certainly intractable for a "subsampled" problem. In this section, we will describe an algorithm with a $\mathcal{O}(\log(1/\epsilon))$ dependence, which is thus appropriate for computing moderate-precision solutions. This algorithm will be a randomized IPCPM with several features specially-designed for MapReduce. In particular, the algorithm will take advantage of the multiple subsampled solutions and the parallelizability of the MapReduce framework.

As background, recall that IPCPMs are similar to the bisection method but work in a high dimensional space. An IPCPM requires a polytope $\mathcal{S}_0$ that is known to contain a full-dimensional ball $\mathcal{B}$ of desired

	num. iter.	addl work
subgradient (Clarkson, 2005)	$\mathcal{O}(n^4/\epsilon^2)$	
gradient (Nesterov, 2009)	$\mathcal{O}(m^{1/2}\log m/\epsilon)$	
ellipsoid (Grötschel et al., 1981)	$\mathcal{O}(n^2\log(\kappa/\epsilon))$	$\mathcal{O}(n^2)$
IPCPMs (see text for refs.)	$\mathcal{O}(n\log(\kappa/\epsilon)$	poly(n)

Table 2. Iterative algorithms for ℓ_p regression: number of iterations and extra work per iteration.

solutions described by a separation oracle. At step k, a query point $x_k \in \operatorname{int}\mathcal{S}_k$ is sent to the oracle. If the query point is not a desired solution, the oracle returns a half space $\mathcal{K}_k$ which contains $\mathcal{B}$ but not x_k, and then we set $\mathcal{S}_{k+1} = \mathcal{S}_k \cap \mathcal{K}_k$ and continue. If x_k is chosen such that $|\mathcal{S}_{k+1}|/|\mathcal{S}_k| \le \alpha,\ \forall k$ for some $\alpha < 1$, then the IPCPM converges geometrically. Such a choice of x_k was first given by (Levin, 1965), who used (but did not provide a way to compute) the center of gravity of $\mathcal{S}_k$. (Tarasov et al., 1988) proved that the center of the maximal-volume inscribed ellipsoid also works; (Vaidya, 1996) showed the volumetric center works, but he didn't give an explicit bound; and (Bertsimas & Vempala, 2004) suggest approximating the center of gravity by random walks, *e.g.*, the hit-and-run algorithm (Lovász, 1999). Table 2 compares IPCPMs with other iterative methods on ℓ_p regression problems. Although they require extra work at each iteration, IPCPMs converge in the fewest number of iterations.[8]

For completeness, we will first describe a standard IPCPM approach to ℓ_p regression; and then we will describe the modifications we made to make it work in MapReduce. Assume that $\sigma_p^{\min}(A) = 1$ and $\kappa_p(A) = \text{poly}(n)$. Let $\hat{f}$ always denote the best objective value we have obtained. Then for any $x \in \mathbb{R}^n$, by convexity,

$$g(x)^T x^* = f(x) + g(x)^T(x^* - x) \le f^* \le \hat{f}. \qquad (6)$$

This subgradient gives us the separation oracle. Let x_0 be the minimal ℓ_2-norm point in Ω, in which case

$$\|Ax_0\|_p \le \kappa_p\|x_0\|_2 \le \kappa_p\|x^*\|_2 \le \kappa_p\|Ax^*\|_p,$$

and hence x_0 is a κ_p-approximate solution. Moreover,

$$\|x^* - x_0\|_\infty \le \|x^*\|_2 \le \|Ax^*\|_p \le \|Ax_0\|_p, \qquad (7)$$

[8] It is for this reason that ICPCMs seem to be good candidates for improving subsampled solutions. Previous work assumes that data are in RAM, which means that the extra work per iteration is expensive. Since we consider large-scale distributed environments where data have to be accessed via passes, the number of iterations is the most precious resource, and thus the extra computation at each iteration is relatively inexpensive. Indeed, by using a randomized IPCPM, we will demonstrate that subsampled solutions can be improved in very few passes.

Algorithm 3 A randomized IPCPM

Input: $A \in \mathbb{R}^{m\times n}$ with $\sigma_p^{\min}(A) \geq 1$, $c \in \mathbb{R}^n$, a set of initial points, number of iterations M, and $N \geq 1$.
Output: An approximate solution $\hat{x}$.

```
Choose K = O(n).
Compute (f(x), g(x)) for each initial point x.
Let f̂ = f(x̂) always denote the best we have.
for i=0,...,M-1 do
    Construct S_i from known (f, g) pairs and f̂.
    Generate random walks in S_i : z_1^(i), z_2^(i), ...
    Let x_k^(i) = (1/K) Σ_{j=(k-1)K+1}^{kK} z_j^(i),  k = 1, ..., N.
    Compute (f(x_k^(i)), g^(i)(x_k^(i))) for each k.
end for
Return x̂.
```

which defines the initial polytope $\mathcal{S}_0$. Given $\epsilon > 0$, for any $x \in \mathcal{B} = \{x \in \Omega \mid \|x - x^*\|_2 \leq \epsilon\|Ax_0\|_p/\kappa_p^2\}$,

$$\begin{aligned}\|Ax\|_p - \|Ax^*\|_p \quad &\leq \|A(x - x^*)\|_p \leq \kappa_p\|x - x^*\|_2 \\ &\leq \epsilon\|Ax_0\|_p/\kappa_p \leq \epsilon\|Ax^*\|_p.\end{aligned}$$

So all points in $\mathcal{B}$ are $(1 + \epsilon)$-approximate solutions. The number of iterations to reach a $(1 + \epsilon)$-approximation is

$$\mathcal{O}(\log(|\mathcal{S}_0|/|\mathcal{B}|)) = \mathcal{O}(\log((\kappa_p^2/\epsilon)^n)) = \mathcal{O}(n\log(n/\epsilon)).$$

This leads to an $\mathcal{O}((mn^2 + \text{poly}(n))\log(n/\epsilon))$-time algorithm, which is better than sampling when ϵ is very small. Note that we will actually apply the IPCPM in a coordinate system defined on Ω, where the mappings from and to the coordinate system of $\mathbb{R}^n$ are given by Householder transforms; we omit the details.

Our randomized IPCPM for use on MapReduce, which is given in Algorithm 3, differs from the standard approach just described in two aspects: sampling initialization; and multiple queries per iteration. In both cases, we take important advantage of the peculiar properties of the MapReduce framework.

For the initialization, note that constructing $\mathcal{S}_0$ from x_0 may not be a good choice since we can only guarantee $\kappa_p = \text{poly}(n)$. Recall, however, that we actually have N subsampled solutions from Algorithm 2, and all of these solutions can be used to construct a better $\mathcal{S}_0$. Thus, we first compute $\hat{f}_k = f(\hat{x}_k)$ and $\hat{g}_k = g(\hat{x}_k)$ for $k = 1, \ldots, N$ in a single pass. For each $\hat{x}_k$, we define a polytope containing x^* using (6) and

$$\|x^* - \hat{x}_k\|_\infty \leq \|A(x^* - \hat{x}_k)\|_p \leq f^* + \hat{f}_k \leq \hat{f} + \hat{f}_k.$$

We then merge all these polytopes to construct $\mathcal{S}_0$, which is described by $2n + N$ constraints. Note also that it would be hard to use all the available approximate solutions if we chose to iterate with a subgradient or gradient method.

For the iteration, the question is which query point we send at each step. Here, instead of one query, we send multiple queries. Recall that, for a data intensive job, the dominant cost is the cost of input/output, and hence we want to extract as much information as possible for each pass. Take an example of one of our runs on a 10-node Hadoop cluster: with a matrix A of size $10^8 \times 50$, then a pass with a single query took 282 seconds, while a pass with 100 queries only took 328 seconds—so the extra 99 queries come almost "for free." To generate these multiple queries, we follow the random walk approach proposed by (Bertsimas & Vempala, 2004). The purpose of the random walk is to generate uniformly distributed points in $\mathcal{S}_k$ such that we can estimate the center of gravity. Instead of computing one estimate, we compute multiple estimates.

We conclude our discussion of our randomized IPCPM algorithm with a few comments.

- The online work of computing (f, g) pairs and the offline work of generating random walks can be done partially in parallel. Because $\mathcal{S}_{i+1} \subset \mathcal{S}_i$, we can continue generating random walks in $\mathcal{S}_i$ while computing (f, g) pairs. When we have $\mathcal{S}_{i+1}$, simply discard points outside $\mathcal{S}_{i+1}$. Even if we don't have enough points left, it is very likely that we have a warm-start distribution that allows fast mixing.
- The way we choose query points works well in practice but doesn't guarantee faster convergence. How to choose query points for guaranteed faster convergence is worth further investigation. However, we are not expecting that by sending $\mathcal{O}(n)$ queries per step we can reduce the number of iterations to $\mathcal{O}(\log(1/\epsilon))$, which may require exponentially many queries.
- Sending multiple queries makes the number of linear inequalities describing $\mathcal{S}_k$ increase rapidly, which is a problem if we have too many iterations. But here we are just looking for, say, fewer than 30 iterations. Otherwise, we can purge redundant or unimportant linear constraints on the fly.

4 Empirical Evaluation

The computations are performed on a Hadoop cluster with 40 CPU cores. We used the ℓ_1 regression test problem from (Clarkson et al., 2013). The problem is of size $5.24e9 \times 15$, generated in the following way:

- The true signal x^* is a standard Gaussian vector.
- Each row of the design matrix A is a canonical

	$\frac{\|x-x^*\|_1}{\|x^*\|_1}$	$\frac{\|x-x^*\|_2}{\|x^*\|_2}$	$\frac{\|x-x^*\|_\infty}{\|x^*\|_\infty}$
ALG1	[0.0057, 0.0076]	[0.0059, 0.0079]	[0.0059, 0.0091]
CT	[0.008, 0.0115]	[0.0090, 0.0146]	[0.0113, 0.0211]
UNIF	[0.0572, 0.0951]	[0.089, 0.166]	[0.129, 0.254]
NOCD	[0.0823, 22.1]	[0.126, 70.8]	[0.193, 134]

Table 3. The 1st and the 3rd quartiles of the relative errors in 1-, 2-, and ∞-norms from 100 independent subsampled solutions of sample size 100000.

vector, which means that we only estimate a single entry of x^* in each measurement. The number of measurements on the i-th entry of x^* is twice as large as that on the $(i+1)$-th entry, $i = 1, \ldots, 14$. We have 2.62 billion measurements on the first entry while only 0.16 million measurements on the last. Imbalanced measurements apparently create difficulties for sampling-based algorithms.

- The response vector b is given by

$$b_i = \begin{cases} 1000\epsilon_i & \text{with prob. } 0.001 \\ a_i^T x^* + \epsilon_i & \text{otherwise} \end{cases}, \ i = 1, \ldots, m,$$

where $\{\epsilon_i\}$ are i.i.d. samples drawn from the standard Laplace distribution. 0.1% measurements are corrupted to simulate noisy real-world data.

Since the problem is separable, we know that an optimal solution is simply given by the median of responses corresponding to each entry. If we use ℓ_2 regression, the optimal solution is given by the mean values, which is inaccurate due to corrupted measurements.

We first check the accuracy of subsampled solutions. We implement Algorithm 1 with block size n^3 (ALG1), which gives $2n^{5/2}$-conditioning; and the Cauchy transform (CT) by (Sohler & Woodruff, 2011), which gives asymptotic $\mathcal{O}(n^{3/2}\log^{3/2} n)$-conditioning; and then we use Algorithm 2 to compute 100 subsampled solutions in a single pass. We compute $\|AE\|_1$ explicitly prior to sampling for a better control on the sample size. We choose $r = 100000$ in Algorithm 2. We also implement Algorithm 2 without conditioning (NOCD) and uniform sampling (UNIF) for comparison. The 1st and the 3rd quartiles of the relative errors in 1-, 2-, and ∞-norms are shown in Table 3. ALG1 clearly performs the best, achieving 0.01 relative error in all the metrics we use. CT has better asymptotic conditioning quality than ALG1 in theory, but it doesn't generate better solutions in this test. This confirms our concerns on the hidden constant in κ_1 and the failure probability. UNIF works but it is about a magnitude worse than ALG1. NOCD generates large errors. So both UNIF and NOCD are not reliable approaches.

Next we try to iteratively improve the subsampled solutions using Algorithm 3. We implement and compare

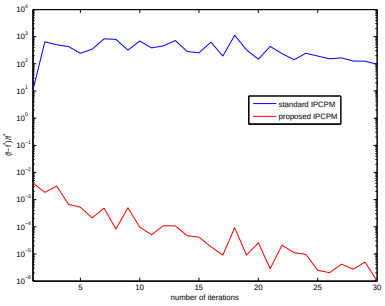

Figure 1. A standard IPCPM approach (single point initialization and single query per iteration) vs. the proposed approach (sampling initialization and multiple queries per iteration) on relative errors in function value.

the proposed IPCPM with a standard IPCPM based on random walks with single point initialization and single query per iteration. We set the number of iterations to 30. The running times for each of them are approximately the same. Figure 1 shows the convergence behavior in terms of relative error in objective value. IPCPMs are not monotonically decreasing algorithms. Hence we see even we begin with a 10-approximate solution with the standard IPCPM, the error goes to 10^3 after a few iterations and the initial guess is not improved in 30 iterations. The sampling initialization helps create a small initial search region; this makes the proposed IPCPM begin at a 10^{-2}-approximate solution, stay below that level, and reach 10^{-6} in only 30 iterations. Moreover, it is easy to see that the multiple-query strategy improves the rate of convergence, though still at a linear rate.

5 Conclusion

We have proposed an algorithm for solving strongly over-determined ℓ_p regression problems, for $p \in [1, 2)$, with an emphasis on its theoretical and empirical properties for $p = 1$. Although some of the building blocks of our algorithm are not better than state-of-the-art algorithms in terms of FLOP counts, we have shown that our algorithm has superior communication properties that permit it to be implemented in MapReduce and applied to terabyte-scale data to obtain a "moderate-precision" solution in only a few passes. The proposed method can also be extended to solving more general convex problems on MapReduce.

Acknowledgments

Most of the work was done while the first author was at ICME, Stanford University supported by NSF DMS-1009005. The authors would like to thank Suresh Venkatasubramanian for helpful discussion and for bringing to our attention several helpful references.

References

Balcan, M.-F., Blum, A., Fine, S., and Mansour, Y. Distributed learning, communication complexity and privacy. *Arxiv preprint arXiv:1204.3514*, 2012.

Bekkerman, R., Bilenko, M., and Langford, J. (eds.). *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011.

Bertsekas, D. P. and Tsitsiklis, J. N. Some aspects of parallel and distributed iterative algorithms—a survey. *Automatica*, 27(1):3–21, 1991.

Bertsimas, D. and Vempala, S. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004.

Clarkson, K. L. Subgradient and sampling algorithms for ℓ_1 regression. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 257–266. SIAM, 2005.

Clarkson, K. L. and Woodruff, D. P. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM symposium on Theory of Computing (STOC)*, 2013.

Clarkson, K. L., Drineas, P., Magdon-Ismail, M., Mahoney, M. W., Meng, X., and Woodruff, D. P. The Fast Cauchy Transform and faster robust linear regression. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

Dasgupta, A., Drineas, P., Harb, B., Kumar, R., and Mahoney, M. W. Sampling algorithms and coresets for ℓ_p regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009.

Daumé, III, H., Phillips, J. M., Saha, A., and Venkatasubramanian, S. Efficient protocols for distributed classification and optimization. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory*, pp. 154–168, 2012.

Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI)*, pp. 137–149, 2004.

Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., and Svitkina, Z. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6 (4):Article 66, 2010.

Goodrich, M. T. Simulating parallel algorithms in the MapReduce framework with applications to parallel computational geometry. *Arxiv preprint arXiv:1004.4708*, 2010.

Grötschel, M., Lovász, L., and Schrijver, A. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

Karloff, H., Suri, S., and Vassilvitskii, S. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 938–948, 2010.

Levin, A. Y. On an algorithm for the minimization of convex functions. In *Soviet Mathematics Doklady*, volume 160, pp. 1244–1247, 1965.

Lovász, L. Hit-and-run mixes fast. *Math. Prog.*, 86(3): 443–461, 1999.

Mackey, L., Talwalkar, A., and Jordan, M. I. Divide-and-conquer matrix factorization. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, 2011.

Mahoney, M. W. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011. Also available at: arXiv:1104.5557.

Meng, X. and Mahoney, M. W. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the 45th Annual ACM symposium on Theory of Computing (STOC)*, 2013.

Mitchell, J. E. Polynomial interior point cutting plane methods. *Optimization Methods and Software*, 18(5): 507–534, 2003.

Nesterov, Y. Unconstrained convex minimization in relative scale. *Mathematics of Operations Research*, 34(1): 180–193, 2009.

Nesterov, Y. and Nemirovsky, A. *Interior Point Polynomial Methods in Convex Programming*. SIAM, 1994.

Portnoy, S. and Koenker, R. The Gaussian hare and the Laplacian tortoise: computability of squared-error versus absolute-error estimators. *Statistical Science*, 12(4): 279–300, 1997.

Rousseeuw, P. J. and Leroy, A. M. *Robust Regression and Outlier Detection*. Wiley, 1987.

Sohler, C. and Woodruff, D. P. Subspace embeddings for the ℓ_1-norm with applications. In *Proceedings of the 43rd annual ACM symposium on Theory of computing (STOC)*, pp. 755–764. ACM, 2011.

Tarasov, S., Khachiyan, L. G., and Erlikh, I. The method of inscribed ellipsoids. In *Soviet Mathematics Doklady*, volume 37, pp. 226–230, 1988.

Vaidya, P. M. A new algorithm for minimizing convex functions over convex sets. *Math. Prog.*, 73:291–341, 1996.

Zhang, Y., Duchi, J., and Wainwright, M. J. Communication-efficient algorithms for statistical optimization. In *Annual Advances in Neural Information Processing Systems 26: Proceedings of the 2012 Conference*, 2012.