Householder Sketch for Accurate and Accelerated Least-Mean-Squares Solvers

Jyotikrishna Dass [1] **Rabi Mahapatra** [1]

Abstract

Least-Mean-Squares (LMS) solvers comprise a class of fundamental optimization problems such as linear regression, and regularized regressions such as Ridge, LASSO, and Elastic-Net. Data summarization techniques for big data generate summaries called coresets and sketches to speed up model learning under streaming and distributed settings. For example, (Maalouf et al., 2019) design a fast and accurate Caratheodory set on input data to boost the performance of existing LMS solvers. In retrospect, we explore classical Householder transformation as a candidate for sketching and accurately solving LMS problems. We find it to be a simpler, memory-efficient, and faster alternative that always existed to the above strong baseline. We also present a scalable algorithm based on the construction of distributed Householder sketches to solve LMS problem across multiple worker nodes. We perform thorough empirical analysis with large synthetic and real datasets to evaluate the performance of Householder sketch and compare with (Maalouf et al., 2019). Our results show Householder sketch speeds up existing LMS solvers in the scikit-learn library up to 100x-400x. Also, it is 10x-100x faster than the above baseline with similar numerical stability. The distributed algorithm demonstrates linear scalability with a near-negligible communication overhead.

1. Introduction

Least-Mean-Squares (LMS) solve a fundamental slice of machine learning optimization and statistics problems that comprise ordinary least squares linear regression (Seber & Lee, 2012), regularized models such as ridge regression (Hoerl & Kennard, 1970), LASSO (Tibshirani, 2011), Elastic-net (Zou & Hastie, 2005). Such machine learning models are statistically robust and easily interpretable. Hence, they find applications in cancer research (Kidd et al., 2018), genomics (D'Angelo et al., 2009), cryptocurrency (Panagiotidis et al., 2018), and most recently in understanding factors of COVID-19 outbreak and its impact (Wang et al., 2020; Lauer et al., 2020; Pandey et al., 2020). Training machine learning models on big data requires resources with large memory, and high computational power. However, in practical applications, data is naturally decentralized, which calls for collaboratively training global machine learning model across multiple sources without sharing original data on a central server. A recent body of work aims to create such secure multi-party computation framework for training distributed regression models (Ben-Or et al., 1988; Sanil et al., 2004; Gascón et al., 2017; Zheng et al., 2019). In addition, efforts have to be made towards feasibility of real-time learning at these sources with relatively smaller memory and computing capabilities. Such distributed learning framework will also need to support streaming data batches to update the models.

Addressing the above requirements entails reformulating the fundamental machine learning problems and using smaller and efficient representations of the original full data to accelerate the training time. Data summarization techniques for big data in (Phillips, 2016; Jubran et al., 2019; Drineas et al., 2006; Feldman et al., 2010; Clarkson & Woodruff, 2009) generate such summaries called coresets and sketches to solve the problems approximately. A *coreset* is a (weighted) subset of data points whereas a *sketch* is a linear mapping of few or all data points in the original dataset which aim to preserve or approximate the covariance matrix. It is possible to leverage these data summaries to speedup model learning, with data distributed across or streaming into various locations. More recently, an award-winning work (Maalouf et al., 2019) proposed a coreset and sketch fusion algorithm LMS-BOOST which accurately solves and accelerates common LMS solvers for Linear, Ridge, LASSO, Elastic-Net in scikit-learn library up to 100x.

1.1. Problem Setup

We define the Least-Mean-Squares (LMS) optimization problem for a data set (X, y), $X \in \mathbb{R}^{n \times d}$, and $y \in \mathbb{R}^n$, where, $n \gg d$, as minimizing the sum of squared loss between the observed prediction $x_j w$, and true response y_i

[1]Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA. Correspondence to: Jyotikrishna Dass <dass.jyotikrishna@tamu.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

for any d- dimensional data sample $x_j^T \in \mathbb{R}^d$ (row of X), $j = 1, \ldots, n$, and LMS model coefficient vector $w \in \mathbb{R}^d$.

$$\min_w f(\|Xw - y\|_2) + g(w). \quad (1)$$

In LINEAR REGRESSION, $f(z) = z^2$, and $g(w) = 0$. In RIDGE REGRESSION, $f(z) = z^2$, and $g(w) = \lambda\|w\|_2$, where, $\lambda > 0$, is ridge regularization hyper-parameter. LMS solvers in scikit-learn library solve system of linear equations, i.e. $(X^TX)w = X^Ty$ (LINEAR REGRESSION), or $(X^TX + \lambda I)w = X^Ty$ (RIDGE REGRESSION).

1.2. Related work

The above overdetermined LMS problems can be solved directly by computing the covariance matrix (X^TX), and $(X^TX)^{-1}$ in the Normal Equations. Despite its ease of implementation, this method is not recommended due to its numerical instability associated with squaring of condition number. Moreover, computing and maintaining the inverse, if it exists, for every new incoming data leads to the accumulation of numerical errors, especially with 32-bit floating-point calculations. Hence, a more numerically stable approach is to factorize X using QR decomposition into matrix Q with orthonormal columns, and upper triangular matrix R, when $n \gg d$. QR decomposition handles a much wider range of matrix by avoiding the condition-number-squaring effect (Golub & Van Loan, 2012).

QR decomposition is a pre-processing step to solve LMS problems. A standard method of obtaining a QR Factorization is via Gram-Schmidt Orthogonalization (Björck, 1994) that is unstable compared to QR via Householder reflectors (Golub & Van Loan, 2012). For applications with sparse input matrices, Givens rotation (George & Heath, 1980) allows full exploitation of the underlying sparsity where non-zeros can be successively annihilated. However, it is to be noted that Householder transformation uses fewer arithmetic operations compared to Givens rotations (Golub & Van Loan, 2012). Nevertheless, it is always possible to compute efficient sketch by switching between Householder transformation and Given rotation implementation of QR decomposition based on the sparsity of the input data without much change in the structure of the framework. We also note that Householder transformation is unconditionally backward stable, by applying only orthogonal transformations, hence solving LMS problems is accurate. For rank-deficient systems where rank, $r < d$, Rank-Revealing QR (Chan, 1987) can be used with an additional computation cost of $O(d^2r)$ which is negligible in our problem setup since $r < d \ll n$.

Various iterative methods have been proposed to solve Least squares problems such as LSQR (Paige & Saunders, 1982), LSMR (Fong & Saunders, 2011), Stochastic proximal accelerated gradient descent (Nitanda, 2014), and more recently randomized methods with random mixing and random sampling such as LSRN (Meng et al., 2014) and Blendenpic (Avron et al., 2010). These results are approximate solutions. *However, our focus is using a theoretically accurate sketch of the input data which could be directly plugged to accelerate scikit-learn* LMS *solvers*.

Our work is inspired by the recent work of (Maalouf et al., 2019) which proposed a novel coreset-sketch fusion algorithm called LMS-BOOST to accurately solve and accelerate scikit-learn LMS solvers based on (Carathéodory, 1907) theorem. Specifically, LMS-BOOST generates accurate coresets from the proposed faster implementation of Caratheodory set. The fundamental idea in this fusion algorithm is to partition the n data points each with feature dimension d into multiple clusters optimally. Then, one computes sketch for each cluster, followed by generating coreset for the union of sketches. Finally, one creates a union of clusters corresponding to selected sketches above and recursively runs the original (slower) Caratheodory algorithm on this union to generate sufficiently small coreset. Algorithm 3 in (Maalouf et al., 2019) uses such coreset to summarize the data into a Caratheodory matrix, $S \in \mathbb{R}^{(d^2+1)\times d}$ as per Algorithm 2, Theorem 3.2 in (Maalouf et al., 2019) to preserve the input covariance i.e. $X^TX = S^TS$, Finally, the above matrix is used as a summarized data for LMS solvers with the asymptotic time complexity of $O(nd^2 + logn \times d^8)$. In retrospect, we seek to explore classical Householder transformation as a candidate for sketching and accurately solving LMS problems such that it is computationally faster than LMS-BOOST (Maalouf et al., 2019).

1.3. Contributions

We note that (Maalouf et al., 2019) considers classical QR decomposition approach to be numerically stable alternative for solving LMS problems which we discussed in previous section. However, it makes the following claims without providing any theoretical or empirical analysis.

Claim 1: QR decomposition is relatively time-consuming.

Claim 2: QR decomposition is unsuitable for exact factorization for streaming data.

In this work, we set to rigorously test and check for validity of the above claims made against the QR decomposition. In this regard, we pose the following questions, respectively.

Q1: Whether a classical and simple approach such as QR decomposition could (theoretically) accurately solve and accelerate common LMS solvers compared to the above state of the art recursive and clustering-based fusion algorithm?

Q2: Whether a numerically stable algorithm could generate

accurate distributed sketches via exact factorization on streaming data?

On using Householder-based QR decomposition, we answer the above questions affirmatively and thereby show both of the above claims made against QR decomposition to be *False*. In this work, we strongly advocate for QR decomposition using Householder reflectors as a theoretically accurate sketch for LMS problems by preserving the covariance of the input matrix, and demonstrate it to be relatively much simpler, memory-efficient, and computationally effective alternative to the fast Caratheodory set based fusion algorithm proposed in (Maalouf et al., 2019). Then, we justify distributed Householder-QR as a numerically stable candidate to generate distributed sketches via exact factorization on streaming data. Our main contributions are

1. We provide a critical analysis based on thorough comparison of Householder-based LMS-QR and Caratheodory-based LMS-BOOST (Maalouf et al., 2019), both theoretically and via extensive empirical results which are missing in the literature. In Section 2 we show that a classical Householder-based QR generates $R \in \mathbb{R}^{d\times d}$ in $O(nd^2 - d^3/3)$ time whereas LMS-BOOST constructs $S \in \mathbb{R}^{(d^2+1)\times d}$ in $O(nd^2 + d^8 \times logn)$ time. Note, $X^TX = R^TR$ is similarly preserved but LMS-QR theoretically and empirically outperforms LMS-BOOST by upto 10x to 100x (refer Section 4(ii)). We used same datasets and baselines as (Maalouf et al., 2019) for fair comparison and even experimented with larger dataset sizes upto $24M \times 50$ while theirs was upto $2M \times 7$.

2. We offer a decentralized QR setup to generate distributed Householder sketches for exactly factorizing the input data partitioned across multiple worker nodes without sharing the original data. Analytically, for exact factorization across p separate workers, it takes $O(\frac{n}{p}d^2)$ time with communicated data volume of $d(d+1)/2$ elements per worker where $d \ll n/p$ (refer, Theorem 3.1). In Section 4(iii) we empirically demonstrate Algorithm 3 to have negligible communication overhead resulting in linear scalability (ideal) and capability to handle dominant computations for accurate distributed sketches.

Hence, the goal of this manuscript is to provide a strong case for classical Householder-based QR decomposition and demonstrate its performance capabilities for solving common LMS problems. Eventually, we shed light on benefits of the Householder representations in terms of its memory, computation time to accelerate LMS solvers, its numerical stability, feasibility of being deployed on distributed network to handle large sample size and feature dimensions, and scalability across multiple worker nodes.

Algorithm 1 HOUSEHOLDER-SKETCH(X, y); see Theorem 2.2

Input: A matrix $X \in \mathbb{R}^{n\times d}$, a vector $y \in \mathbb{R}^n$

Output: A matrix $R \in \mathbb{R}^{d\times d}$ is upper triangular such that $X^TX = R^TR$, and a vector $\bar{y} \in \mathbb{R}^d$ is top d elements of the reflected vector Q^Ty

$(\mathcal{V}, R) :=$ HOUSEHOLDER-QR(X) `// see Theorem 2.1, see Algorithm 4 in Supplementary`

$\bar{y} :=$ MULTIPLY-QC$(\mathcal{V}, y,$ `'T'`$)$ `// implicit` Q^Ty, `see (Golub & Van Loan, 2012), see Algorithm 5 in Supplementary`

$R \leftarrow R[0:d,:]$ `//` $d \times d$ `triangular block`

$\bar{y} \leftarrow \bar{y}[0:d]$ `// top` d `elements`

return $(R, \bar{y})$

2. Least Mean Squares - QR

In this section we review LMS-QR; QR decomposition based Least-Mean-Squares (LMS) solver via Householder transformation. The LMS objective in Equation (1) with $X = QR$ reformulates to

$$\min_w f(\|QRw - y\|_2) + g(w). \tag{2}$$

where, Q is an $n \times n$ orthogonal matrix and R is an $n \times d$ upper trapezoidal matrix when $n > d$. We assume the data matrix X is full column rank. With Householder transformation, the benefit is avoiding the explicit computation and storing of large Q which otherwise becomes prohibitive for big data sets. Rather, one may simply represent Q as d Householder matrices $\mathcal{H} = \{H(j) : j = 1, \ldots, d\}$ such that $Q = H(1) \times H(2) \ldots \times H(d)$. Each $H(j)$ has the form $H(j) = I - tau \times v(j) \times v(j)'$, where tau is a real scalar, and $v(j)$ is a real vector called Householder reflector which we store in the reflector set $\mathcal{V} = \{v(j) : j = 1, \ldots, d\}$. In essence, any operations involving Q are now computed using the above memory-efficient reflector set. Theorem 2.1 formally presents Householder-QR.

Theorem 2.1 (Householder-QR (Golub & Van Loan, 2012))**.** *Let matrix $X \in \mathbb{R}^{n\times d}$ with $n > d$. Householder QR decomposition of X generates set of d Householder matrices $\mathcal{H}$ and an $n \times d$ upper trapezoidal matrix R. The Householder matrices are stored as a set of d Householder reflectors $\mathcal{V}$. Total memory footprint of above factors is nd elements with time complexity of $O(nd^2)$ for $n \gg d$.*

We now present Theorem 2.2 which uses the factors from Householder-QR to create Householder sketch as per Algorithm 1 for further applications in LMS problems.

Theorem 2.2 (Householder Sketch)**.** *Let $X \in \mathbb{R}^{n\times d}$ be the original data matrix, $y \in \mathbb{R}^n$ be the corresponding output label or response vector, and $n \gg d$. Let $X = QR$ be Householder QR decomposition. Then, (R, Q^Ty)*

Algorithm 2 LMS-QR$(X, y, \texttt{params})$

Input: A matrix $X \in \mathbb{R}^{n \times d}$, a vector $y \in \mathbb{R}^n$, and a list of LMS parameters, $params$
Output: A vector of model coefficients, $w \in \mathbb{R}^d$

$(R, \bar{y}) :=$ HOUSEHOLDER-SKETCH(X, y) `// see Algorithm 1`
$w :=$ LMS$(R, \bar{y}, params)$ `// LINREG, RIDGECV, LASSOCV, ELASTICCV in sckit-learn`
return w

is a memory-efficient and theoretically accurate sketch of original data (X, y) such that $X^TX = R^TR$, and has memory footprint of $(\frac{d(d+3)}{2})$ elements, computed in time $O(nd^2)$.

Proof. From Equations (1) and (2), and $X = QR$, where $QQ^T = Q^TQ = I$

$$\|Xw - y\|_2 = \|QRw - y\|_2 = \|QRw - QQ^Ty\|_2$$
$$= \|Q\|_2\|Rw - Q^Ty\|_2 = \|Rw - Q^Ty\|_2$$

See remaining proof in supplementary material. □

Accelerating LMS solvers. Householder sketch (R, Q^Ty) is precomputed and applied directly to existing LMS solvers in scikit-learn library instead of original (X, y). Reducing the memory cost from $O(nd)$ in (X, y) to $O(d^2)$ in Householder sketch (R, Q^Ty) speeds up the LMS solver. Specifically, with the Householder sketch, the time for constructing R^TR and $R^T(Q^Ty)$ are $O(d^3)$ and $O(d^2)$, respectively which are significantly faster than the original time $O(nd^2)$ and $O(nd)$ spent by the solver on constructing X^TX and X^Ty, respectively. Then, LMS solves a *primal* problem with system of d equations and d unknowns (w, solver coefficients). We present LMS-QR formally in Algorithm 2. It is to be noted that constructing HOUSEHOLDER-SKETCH takes $O(nd^2)$ for $n \gg d$ (see Theorem 2.2), and is more computationally dominant than running the LMS solver such as LINREG, RIDGECV, LASSOCV, ELASTICCV in sckit-learn. The above trend is also empirically validated in Figure 2 for RIDGE solver as proof of concept.

3. Distributed LMS-QR

In this section, we present a distributed version of LMS-QR that parallelizes the computations, and, scales Algorithm 2 across multiple workers (computing cores or users) to solve the global LMS problem. We recall that HOUSEHOLDER-SKETCH (see Algorithm 1) is the more computational dominant step than the solver in LMS-QR (see Algorithm 2). HOUSEHOLDER-SKETCH calls HOUSEHOLDER-QR

Algorithm 3 DISTRIBUTED LMS-QR$(p, X, y, \texttt{params})$

Input: A scalar $p > 0$ parallel workers (cores or users), a matrix $X = (X_1^T|\dots|X_p^T)^T, X_i \in \mathbb{R}^{\frac{n}{p} \times d}$, a vector $y = (y_1^T|\dots|y_p^T)^T, y_i \in \mathbb{R}^{\frac{n}{p}}$, a list of LMS parameters, $params$
Output: A vector of model coefficients, $w \in \mathbb{R}^d$

`//` $(\mathcal{V}, \texttt{R}) :=$ DISTRIBUTED HOUSEHOLDER-QR(X), `see Theorem 3.1`
for *every worker* $i \in \{1, 2, \dots, p\}$ **do**
  $(\mathcal{V}_i, R_i) :=$ HOUSEHOLDER-QR(X_i) `// see Theorem 2.1`
  $R_i \leftarrow R_i[0:d,:]$ `//` $d \times d$ `triangular block`
  $R_{stack} :=$ GATHER$(R_i, \texttt{root} = 0)$ `//` $R_{stack} =$ `vstack`$(R_1, \dots, R_p)$ `at Master`
end
if $i == 1$ **then** `// check for Master`
  $(\mathcal{V}_M, R_M) :=$ HOUSEHOLDER-QR(R_{stack}) `// see Theorem 2.1`
  $R_M \leftarrow R_M[0:d,:]$ `//` $d \times d$ `triangular block`
end
`//` $\mathcal{V} = [\mathcal{V}_1, \dots, \mathcal{V}_p, \mathcal{V}_M]$ `is never centralized or shared`
`//` $Q = \texttt{diag}(Q_1, \dots, Q_p)Q_M, \texttt{and}, R = R_M,$ `see Theorem 3.1`
`//` $\bar{\texttt{y}} :=$ DISTRIBUTED MULTIPLY-Qc(`V,y,'T'), see Corollary 3.1.1`
for *every worker* $i \in \{1, 2, \dots, p\}$ **do**
  $\bar{y_i} :=$ MULTIPLY-QC$(\mathcal{V}_i, y_i, \texttt{'T'})$ `// implicit` $Q_i^Ty_i$ `,see Algorithm 5 in Supp.`
  $\bar{y_i} \leftarrow \bar{y_i}[0:d]$ `// select top` d `elements`
  $\bar{y}_{stack} :=$ GATHER$(\bar{y_i}, \texttt{root} = 0)$ `//` $\bar{y}_{stack} =$ `vstack`$(\bar{y_1}, \dots, \bar{y_p})$ `at Master`
  if $i == 1$ **then** `// check for Master`
    $y_{\bar{M}} :=$ MULTIPLY-QC$(\mathcal{V}_M, \bar{y}_{stack}, \texttt{'T'})$ `// implicit` $Q_M^T\bar{y}_{stack}$
    $y_{\bar{M}} \leftarrow y_{\bar{M}}[0:d]$ `// select top` d `elements`
  end
end
$\bar{y} := y_{\bar{M}}$ `//` $\bar{y} = Q^Ty = Q_M^T((Q_1^Ty_1)^T|\dots|(Q_p^Ty_p)^T)^T$ `// Solving LMS`
if $i == 1$ **then** `// check for Master`
  $w :=$ LMS$(R, \bar{y}, params)$ `// run LMS solver at Master`
  BROADCAST$(w, \texttt{root} = 0)$ `// every worker receives the global model`
end
return w

to perform $X = QR$ decomposition on the data matrix. Then, MULTIPLY-QC is invoked to eventually compute $\bar{y} = Q^T y \in \mathbb{R}^d$.

Now, we show the feasibility of *exactly* distributing LMS-QR algorithm across multiple worker nodes without any approximations. To design a DISTRIBUTED LMS-QR in Algorithm 3, the more computationally expensive HOUSEHOLDER-SKETCH algorithm is parallelized across multiple *workers*. This is achieved via DISTRIBUTED HOUSEHOLDER-QR (see Theorem 3.1, and Steps 1-9 in Algorithm 3), and DISTRIBUTED MULTIPLY-QC (see Corollary 3.1.1, and Steps 10-19 in Algorithm 3) which create local and master Householder sketches from the local data $(X_i, y_i), i = 1, \ldots, p$, generated *i.i.d.* at each worker node. Finally, we run the LMS solver (Steps 20 - 24 in Algorithm 3) at the *master* with $(R, Q^T y)$, and broadcast the global LMS model coefficients w to all the workers.

Theorem 3.1 (Distributed Householder-QR (Dass et al., 2018))**.** *Let $X = (X_1^T|\ldots|X_p^T)^T$, where, $X_i \in \mathbb{R}^{\hat{n}\times d}$ be local data matrix of parallel worker, $i = 1, \ldots, p$, where $\hat{n} \gg d$, and, $n = p\hat{n}$. Let, $X_i = Q_i R_i$ be constructed via local* HOUSEHOLDER-QR *(see Algorithm 1) for each $i = 1, \ldots, p$, in parallel. Then, $X = QR$ for the complete data matrix can be constructed exactly, such that $Q = \texttt{diag}(Q_1, \ldots, Q_p)Q_M$, and $R = R_M$, where $R_{stack} = Q_M R_M$ via another* HOUSEHOLDER-QR *on $R_{stack} = (R_1^T|\ldots|R_p^T)^T$ gathered from all workers. The above* DISTRIBUTED HOUSEHOLDER-QR *has a computational time complexity of $O(\frac{n}{p}d^2)$, with a communicated data volume of $(\frac{d(d+1)}{2})$ elements by each worker.*

Proof. See full proof in the supplementary material. □

Relation to TSQR and BLOCK-QR. The construction of DISTRIBUTED HOUSEHOLDER-QR (Dass et al., 2018) is inspired from the parallel Tall-Skinny QR (TSQR) algorithm (Demmel et al., 2012). Unlike the binary reduction tree in TSQR, (Dass et al., 2018) use a simple two-level organisation where the first level involves local HOUSEHOLDER-QR on each participating worker in parallel (Step 2 in Algorithm 3), followed by a second level which computes HOUSEHOLDER-QR at the master (Step 7 in Algorithm 3). Since the communication overhead is negligible as demonstrated in Figure 2, the choice of implementation of the DISTRIBUTED HOUSEHOLDER-QR is justified. Moreover, designing such a framework supports decentralized machine learning applications with workers (users) on the edge of distributed network. Here, every worker has same and direct access to the master compared to the neighbors in TSQR's binary reduction scheme. Finally, we would like to highlight that established packages such as ScaLAPACK and Elemental (Poulson et al., 2013) use different matrix blocking and collectives for performing Householder QR on multiple worker nodes via All-reduction scheme. In contrast, DISTRIBUTED HOUSEHOLDER-QR employs a reduction scheme similar to TSQR (Demmel et al., 2012) where the global R resides on only one node (master) rather than being shared across all the workers. Due to relatively lower synchronisation cost, DISTRIBUTED HOUSEHOLDER-QR similar to TSQR (Demmel et al., 2012) obtains a performance benefit over ScaLAPACK (Ballard et al., 2014). BLOCK-QR (Mathias & Stewart, 1993) performs vertical partitions of the columns (features) into blocks, generates Householders and uses WY block representation (Bischof & Van Loan, 1987) and finally performs Level 3 BLAS update before moving to next block. Instead, Algorithm 3 horizontally partitions the rows (samples) for decentralized setup and performs Householder (DGEQRF) operation locally on data partition at each worker in parallel and once globally.

Corollary 3.1.1 (Distributed Multiply-Qc)**.** *Let $c = (c_1^T|\ldots|c_p^T)^T \in \mathbb{R}^n$, where, $c_i \in \mathbb{R}^{\hat{n}}$ be some local vector at parallel worker with local data matrix X_i, $i = 1, \ldots, p$, where $\hat{n} \gg d$, and, $n = p\hat{n}$. Let orthogonal matrices Q_M, and Q_i, $i = 1, \ldots, p$ be constructed via* DISTRIBUTED HOUSEHOLDER-QR *as per Theorem 3.1 such that $Q = \texttt{diag}(Q_1, \ldots, Q_p)Q_M$. Then, the reflected vector, $Q^T c$ (or Qc) can be constructed exactly by making $(p+1)$ calls to* MULTIPLY-QC *(see Step 2 in Algorithm 1) such that*

$$Q^T c = Q_M^T((Q_1^T c_1)^T|\ldots|(Q_p^T c_p)^T)^T$$

or,

$$Qc = \texttt{diag}(Q_1, \ldots, Q_p)Q_M(c_1^T|\ldots|c_p^T)^T$$

The above DISTRIBUTED MULTIPLY-QC *has a computational time complexity of $O(\frac{n}{p}d + pd^2)$, with a communicated data volume of (d) elements by each worker.*

Proof. See full proof in the supplementary material. □

Privacy. From Theorem 3.1, and Corollary 3.1.1, it can be observed that the DISTRIBUTED LMS-QR in Algorithm 3 can locally compute $(R_i, Q_i^T y_i)$ for each worker without the need to share its original data X_i to a centralized node or any of the neighbors. By maintaining Q_i privately, the algorithm avoids any other worker (or master) to reconstruct X_i accurately. In our experiments, *master* is designated via Message Passing Interface protocol. We acknowledge that future work could explore strong theoretically supported privacy-preserving Householder computations.

4. Experiments

In this section we perform extensive empirical analysis for the LMS-QR (Algorithm 2). Recall, HOUSEHOLDER-SKETCH (Algorithm 1) on the input data (X, y) generates memory-efficient $(R, Q^T y)$ to accurately solve, and

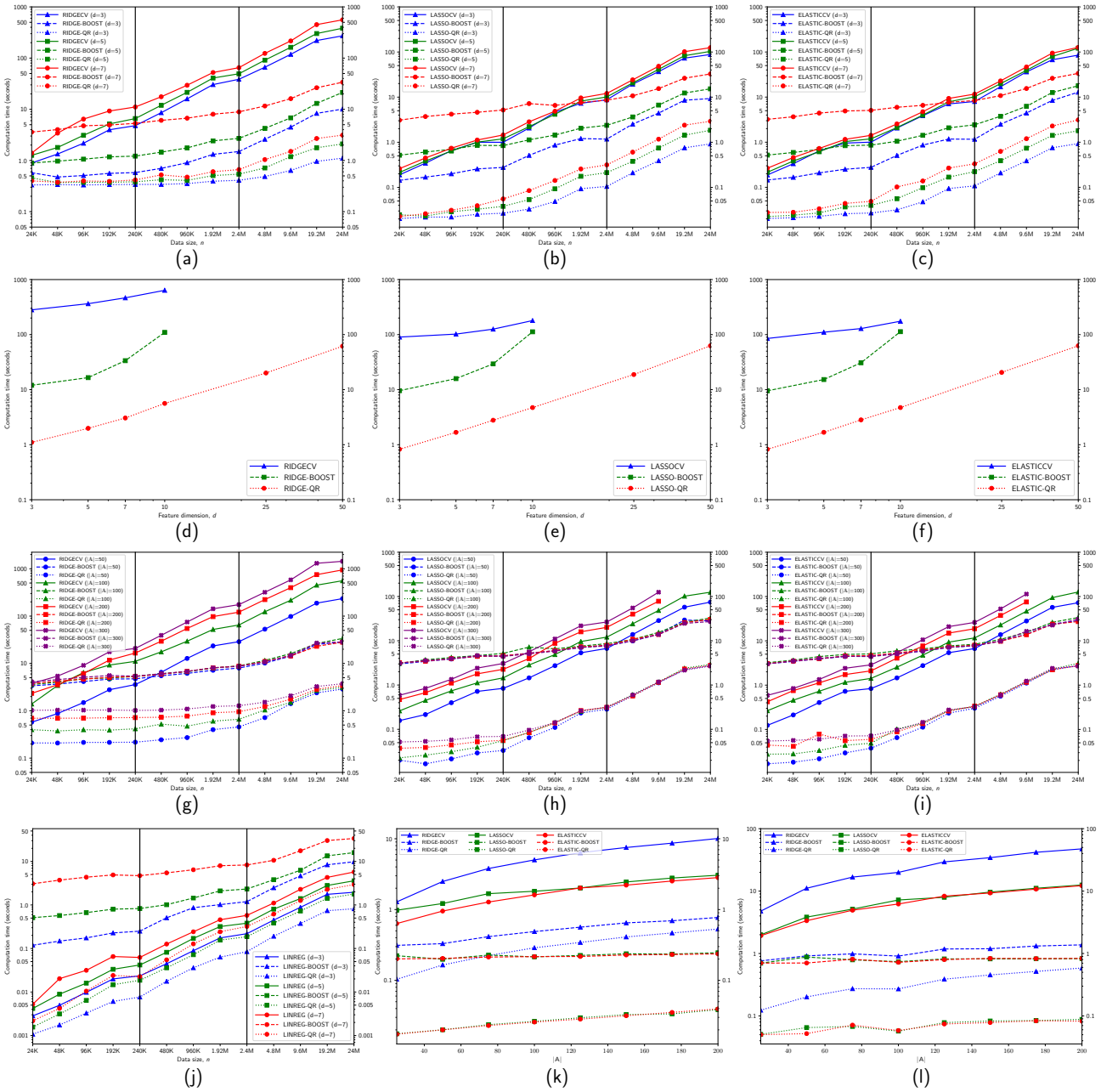

Figure 1. Sequential training time {RIDGE, LASSO, ELASTIC} (a)-(c): vs data size, n with feature dimension, $d = \{3, 5, 7\}$, (d)-(f): vs d with $n = 24M$, (g)-(i): vs n with hyper-parameter size $|\mathbb{A}|$, (j): LINREG vs n, (k): vs $|\mathbb{A}|$ for 3D Road Network, (l): vs $|\mathbb{A}|$ for Household Power Consumption. Cross validation folds, $|m|= 3$ for synthetic datasets (a)-(j) and $|m|= 2$ for real datasets (k)-(l)

.

accelerate common LMS solvers in scikit-learn library. To implement Algorithm 1, we use `LAPACK.dgeqrf()`, and `LAPACK.dormqr()` subroutines for HOUSEHOLDER-QR, and MULTIPLY-QC, respectively. We present detailed experimental setup, and extensive discussion of our results.

`(i) Experimental Setup.` We evaluated the performance of LMS-QR algorithm on regression models such as Linear Regression, Ridge, LASSO, and Elastic-Net in scikit-learn library (Pedregosa et al., 2011). We used Google Colab to run our experiments with the above LMS-QR algorithms via Python3 Google Compute Engine running on Intel Xeon CPU @ 2.20GHz and 25 GB RAM. We used following datasets for evaluation, and fair comparison of LMS-QR performance with the default LMS solvers (with cross validation), and with Fast Caratheodory coreset based

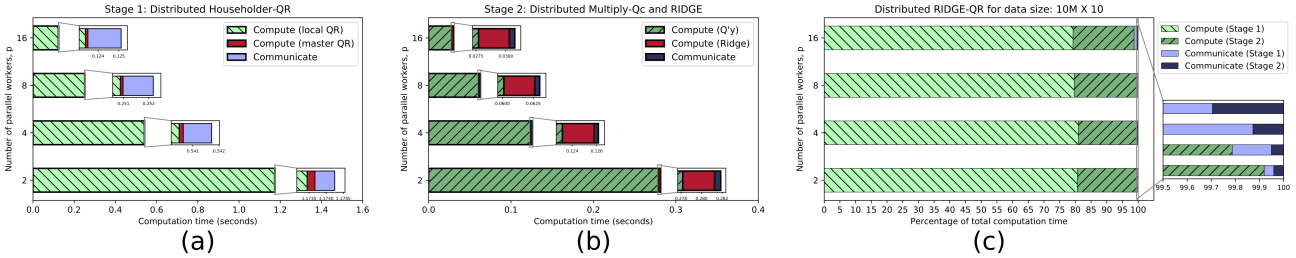

Figure 2. Breakdown of DISTRIBUTED RIDGE-QR training time with zoomed insets depicting communication time (a): Stage 1: DISTRIBUTED HOUSEHOLDER-QR, (b): Stage 2: DISTRIBUTED MULTIPLY-QC and RIDGE, (c): Combined percentage

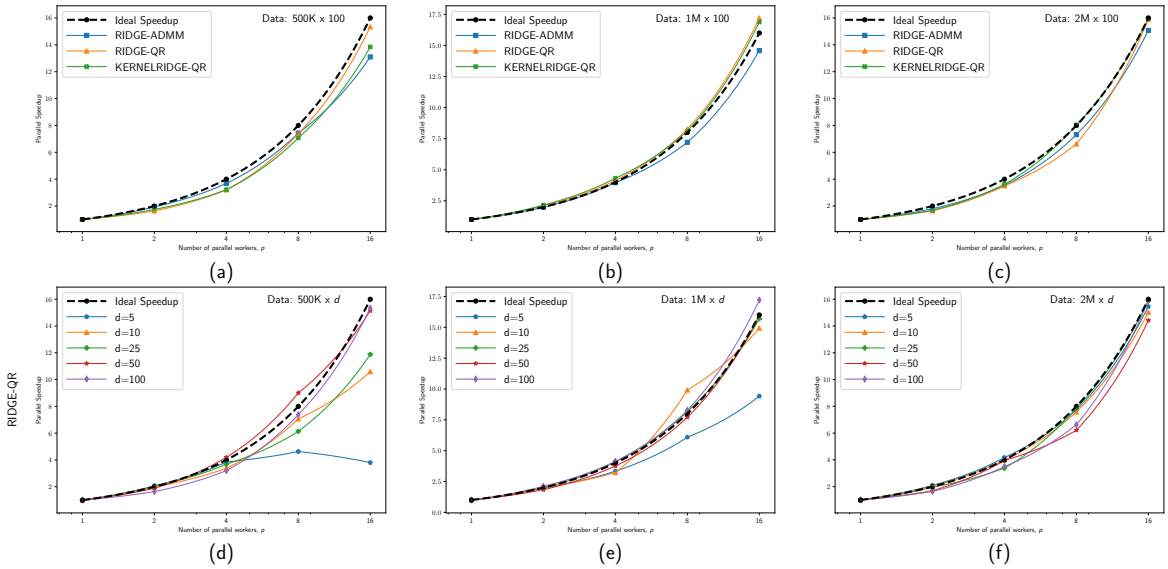

Figure 3. Comparing scalability of various algorithms to solve RIDGE problem on synthetic datasets of size $n \times d$ (a)-(c): Various $n = \{500K, 1M, 2M\}, d = 100$, (d)-(f): DISTRIBUTED RIDGE-QR with $d = \{5, 10, 25, 50, 100\}$

LMS-BOOST (Maalouf et al., 2019).

(i) Synthetic data (X, y) comprising uniform random entries in $[0, 100)$ for sequential experiments.

(ii) 3D Road network(Kaul et al., 2013) dataset with $n = 434,874$ data samples. We selected $d = 2$ feature attributes (*longitude*, *latitude*) as per (Maalouf et al., 2019) to predict *height (in metres)*.

(iii) Individual household electric power consumption (Hebrail & Berard, 2012) dataset with $n = 2,075,259$ data samples. We selected $d = 8$ attributes as per (Maalouf et al., 2019) to predict the *house price*.

For *distributed* experiments, we used the Anaconda Python distribution and MPI for Python (mpi4py) package on the Texas A&M University HPRC *Ada* computing cluster of Intel Xeon CPU @ 2.5GHz. We used synthetic dataset with uniform random entries in $(-100, 100)$ with zero-centering for evaluating the distributed performance of RIDGE-QR as our case study on a cluster of $p = \{2, 4, 8, 16\}$ workers (computing cores). However, it is to be noted that DISTRIBUTED LMS-QR technique is easily applicable to other solvers. Linear algebra was handled by LAPACK/BLAS, through the Intel Math Kernel Library. We ensure that each worker (core) was assigned from a different node in the cluster to ensure distributed memory with MKL threads per core limited to 1. Each test was performed 20 times, and the best result was chosen.

`(ii) Sequential Training Time.` LMS-QR works with memory-efficient $(R, Q^T y)$ with just d rows in R compared to LMS with n rows in X of the original data (X, y), and $(d^2 + 1)$ rows in the reduced matrix from the Fast Caratheodory coresets in LMS-BOOST (Maalouf et al., 2019). Construction of R via Householder transformation in LMS-QR, and the reduced matrix via Fast Caratheodory set for LMS-BOOST take time that is linear in n, and quadratic in d. Figure 1 (a)-(i) depicts the sequen-

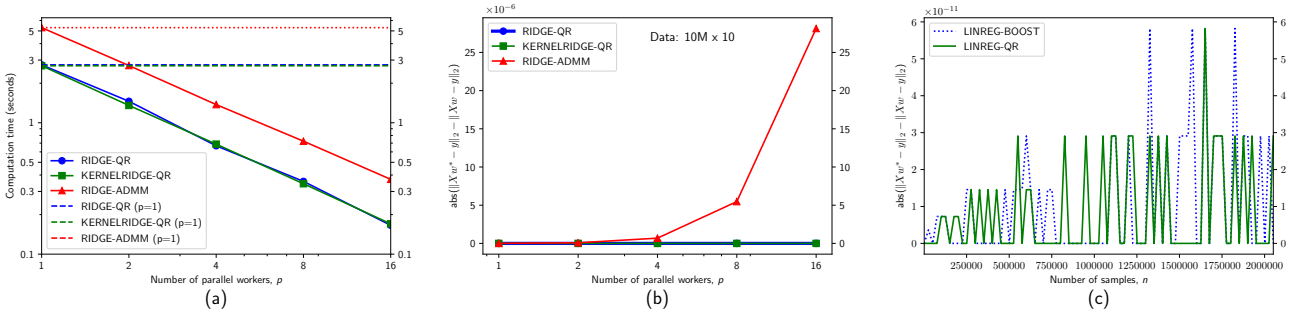

Figure 4. (a)-(b): Comparing DISTRIBUTED RIDGE-QR / KERNELRIDGE-QR (linear kernel), and RIDGE-ADMM for $10M \times 10$ synthetic data based on (a) Computation time (b) Accuracy, w^* is solution from scikit-learn RIDGE, (c) Accuracy of LINREG-QR and LINREG-BOOST on Household Power Consumption dataset ($\sim 2M \times 8$), w^* is solution from scikit-learn LINEARREGRESSION

tial training computation time on synthetic dataset for the LMS-QR, and compares them with respective LMSCV (Pedregosa et al., 2011), and LMS-BOOST (Maalouf et al., 2019), where, LMS={RIDGE, LASSO, ELASTIC}. From Figure 1 (a)-(c), we observe that for various feature dimensions $d = \{3, 5, 7\}$ and data sizes $n = \{240K, \ldots, 24M\}$, LMS-QR accelerates the running time of default LMSCV by 100x for both LASSO, and ELASTIC, and by 400x for RIDGE. In comparison with LMS-BOOST, RIDGE-QR outperforms RIDGE-BOOST by 10x for $d = 7$ and various n in Figure 1(a). Moreover, LASSO/ELASTIC/LINREG-QR runs upto 100x faster when $n < 2.4M$ and upto 10x faster when $n > 2.4M$ than LASSO/ELASTIC/LINREG-BOOST for $d = 7$ in Figure 1 (b)(c)(j). In Figure 1 (d)-(f), we demonstrate the running time of LMS-QR for $n = 24M$ and various $d = \{3, 5, 7, 10, 25, 50\}$. We observe that LMSCV, and LMS-BOOST (Maalouf et al., 2019) with running time $O(nd^2 + logn \times d^8)$, run out of memory for $d = \{25, 50\}$ while LMS-QR could handle growing feature dimension as shown. Hence, we demonstrate that LMS-QR can easily **handle big datasets** with increasing data size n and feature dimension d while enjoying the **fastest running times** compared to LMSCV and LMS-BOOST. For various size of hyper-parameter set for cross validation, $|\mathsf{A}| = \{50, 100, 200, 300\}$, for cross validation, Figure 1 (g)-(i) depict LMS-QR to be consistently faster than LMS-CV and LMS-BOOST. We observe similar trend for the real datasets in Figure 1 (k)-(l).

`(iii) Distributed Training Time.` We evaluate the performance of DISTRIBUTED RIDGE-QR as a case study to demonstrate the effectiveness of the distributed implementation. From Algorithm 3, DISTRIBUTED RIDGE-QR can be split into two main stages: (**Stage 1**) DISTRIBUTED HOUSEHOLDER-QR, and (**Stage 2**) DISTRIBUTED MULTIPLY-QC, followed by solving the Ridge regression. Figure 2 (a)-(b) demonstrate the above two stages on a $10M \times 10$ synthetic dataset. **Stage 1** running time can be further broken into three main components: local HOUSEHOLDER-QR time, master HOUSEHOLDER-QR time, and communication (`gather`) time as illustrated in Figure 2 (a). Since each worker performs its local HOUSEHOLDER-QR on partitioned data of size $\frac{n}{p} \times d$ (Theorem 3.1), we observe it to be empirically more computationally dominant than the master HOUSEHOLDER-QR that works on gathered matrices of size $pd \times d$. We show that on doubling the number of parallel workers, i.e. $p = \{2, 4, 8, 16\}$ workers, the time to calculate the local HOUSEHOLDER-QR is reduced by half, i.e. $\{1.1734, 0.5405, 0.2508, 0.1233\}$ seconds respectively demonstrating it is **fully parallelized**. In comparison, master HOUSEHOLDER-QR computation time, and communication time to `gather` merely $d(d+1)/2$ elements is nearly negligible as depicted in Figure 2 (a).

Next, in Figure 2(b), we observe the timings for **Stage 2** in DISTRIBUTED RIDGE-QR. The running time of this stage can be split into following main components: DISTRIBUTED MULTIPLY-QC time to compute $Q^T y$, and time to solve the ridge regression via popular RIDGE solver (Pedregosa et al., 2011). We observe in Figure 2(b) that DISTRIBUTED MULTIPLY-QC time is computationally dominant as expected from Corollary 3.1.1, and can be **fully parallelized** with reported timings of $\{0.2782, 0.1232, 0.06012, 0.0277\}$ seconds on $p = \{2, 4, 8, 16\}$ workers, respectively. Relatively, we also observe that solving RIDGE regression with $RR^T \in \mathbb{R}^{d \times d}$, is nearly negligible in computation time across various choices of p. Moreover, the communication required in this stage involves a *gather* of $Q_i^T y_i \in \mathbb{R}^d$ at the master, and broadcasting w to all workers, which is negligible as well.

Finally, in Figure 2 (c) we depict the percentage of total running time for DISTRIBUTED RIDGE-QR that is spent on computation and communication components of **Stage 1**, and **Stage 2** for various choices of p. We observe the **communication overhead** to be **nearly negligible**, and local HOUSEHOLDER-QR to be the most computationally

dominant component. The latter can be parallelized significantly across multiple workers. We also observe that for any given dataset size and choice of p, as long as the parallelizable components, namely, local HOUSEHOLDER-QR, and DISTRIBUTED MULTIPLY-QC remain as the most computationally dominant components, the algorithm will continue to scale as per Amdahl's Law (Amdahl, 1967). So for big datasets, we expect good scalability on large number of p workers as discussed next.

`(iv) Scalability.` We discuss parallel speedup under strong scaling scenario wherein the overall problem size stays fixed but the number of parallel workers p is doubled. Parallel speedup is defined as the ratio of the running time for the sequential algorithm to that of the corresponding parallel algorithm on p workers. When speedup of any parallel algorithm is p, it exhibits ideal speedup with linear scalability. Figure 3 (a)-(c) demonstrates the parallel speedup of DISTRIBUTED RIDGE-QR, and compares it with popular distributed technique ADMM (Boyd et al., 2011), here, RIDGE-ADMM that uses original data (X, y) on RIDGE (Pedregosa et al., 2011), for $w-$ update step. In Figure 3 (a)-(c), we observe that with larger data sample sizes $n = \{500K, 1M, 2M\}$, DISTRIBUTED RIDGE-QR along with its dual equivalent DISTRIBUTED KERNELRIDGE-QR (linear kernel) exhibit **almost linear scalability** across p by approaching the **ideal speedup**. This is attributed to its negligible communication overhead, and almost fully parallelizable computational components as discussed earlier. It is also worth noticing from Figure 3 (d)-(f) that as the feature dimension increases from $d = 5$ to $d = 10$ in each plot, DISTRIBUTED RIDGE-QR speedup tends to be more linear for high p, thereby capable of showing high scalability for datasets with large feature size on large number of parallel workers. In Figure 4 (a) we observe that solving RIDGE regression problem with linear kernel in *dual* form using KERNELRIDGE-QR has the same computation time as that of solving the *primal* form using RIDGE-QR for various sequential and distributed settings, $p = \{1, 2, 4, 8, 16\}$. Moreover, Householder-Sketch based sequential ($p = 1$) and distributed ($p = \{2, 4, 8, 16\}$) algorithms run much faster than the corresponding implementation of iterative RIDGE-ADMM algorithms on the original data.

`(v) Numerical Stability.` Figure 4 (b) shows that DISTRIBUTED RIDGE-QR is numerically stable to rounding errors with increasing number of workers p, while ADMM being an iterative learning algorithm is more susceptible. For the sequential implementation on linear regression problem, (Maalouf et al., 2019) had demonstrated much better numerical stability of LINREG-BOOST compared to $(X^T X)^{-1}$. Here, Figure 4 (c) presents the numerical stability of LINREG-QR with scaled factor of 10^{-11} to demonstrate similar trend to that of LINREG-BOOST (Maalouf et al., 2019).

5. Conclusions and Future Work

We demonstrate that Householder transformation generates a theoretically accurate sketch that is relatively more memory-efficient and computationally faster than the LMS-BOOST algorithm in (Maalouf et al., 2019) to accurately solve LMS problems. In principle, Householder sketch accelerates common LMS solvers in scikit-learn library up to 100x-400x, and outperforms the strong baseline LMS-BOOST by 10x-100x with similar numerical stability. The distributed implementation achieves linear scalability with negligible communication overhead for large sample size and dimension across multiple worker nodes. We believe that the above results are valuable for the community to realize not to disregard classical techniques so quickly, rethink how some comparisons are done, identify common misconceptions, and reassess what the most appropriate algorithms for certain problems are. We have open-sourced our codes **here**. Future work includes experimenting with streaming data batches and incorporating fault tolerance and strong theoretical guarantees for privacy to ensure secured multi-party decentralized training.

Acknowledgements

We are grateful to Rengang Yang of Texas A&M University for his initial contributions to the code base to run the distributed experiments in our prior attempt. We thank the anonymous reviewers and the meta-reviewers from NeurIPS 2020 and ICML 2021 for their valuable suggestions, constructive feedback and encouragement which helped us revise and improve the manuscript. Finally, we acknowledge the computing infrastructure and support offered by High Performance Research Computing at Texas A&M University to run our distributed experiments.

References

Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, 1967.

Avron, H., Maymounkov, P., and Toledo, S. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

Ballard, G., Demmel, J., Grigori, L., Jacquelin, M., Nguyen, H. D., and Solomonik, E. Reconstructing householder vectors from tall-skinny qr. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 1159–1170. IEEE, 2014.

Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant

distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pp. 1–10, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912640. doi: 10.1145/62212.62213. URL https://doi.org/10.1145/62212.62213.

Bischof, C. and Van Loan, C. The wy representation for products of householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987.

Björck, Å. Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316, 1994.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Carathéodory, C. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.

Chan, T. F. Rank revealing qr factorizations. *Linear algebra and its applications*, 88:67–82, 1987.

Clarkson, K. L. and Woodruff, D. P. Numerical linear algebra in the streaming model. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pp. 205–214, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062.

D'Angelo, G. M., Rao, D., and Gu, C. C. Combining least absolute shrinkage and selection operator (lasso) and principal-components analysis for detection of gene-gene interactions in genome-wide association studies. In *BMC proceedings*, volume 3, pp. S62. Springer, 2009.

Dass, J., Sarin, V., and Mahapatra, R. N. Fast and communication-efficient algorithm for distributed support vector machine training. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1065–1076, 2018.

Demmel, J., Grigori, L., Hoemmen, M., and Langou, J. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM Journal on Scientific Computing*, 34 (1):A206–A239, 2012.

Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Sampling algorithms for l2 regression and applications. In *SODA '06*, 2006.

Feldman, D., Monemizadeh, M., Sohler, C., and Woodruff, D. P. Coresets and sketches for high dimensional subspace approximation problems. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010.

Fong, D. C.-L. and Saunders, M. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971, 2011.

Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., and Evans, D. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017 (4):345–364, 2017.

George, A. and Heath, M. T. Solution of sparse linear least squares problems using givens rotations. *Linear Algebra and its applications*, 34:69–83, 1980.

Golub, G. H. and Van Loan, C. F. *Matrix computations*, volume 3. JHU press, 2012.

Hebrail, G. and Berard, A. Individual household electric power consumption data set. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption, 2012.

Hoerl, A. E. and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Jubran, I., Maalouf, A., and Feldman, D. Introduction to coresets: Accurate coresets. *arXiv preprint arXiv:1910.08707*, 2019.

Kaul, M., Yang, B., and Jensen, C. S. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pp. 137–146. IEEE, 2013.

Kidd, A. C., McGettrick, M., Tsim, S., Halligan, D. L., Bylesjo, M., and Blyth, K. G. Survival prediction in mesothelioma using a scalable lasso regression model: instructions for use and initial performance using clinical predictors. *BMJ open respiratory research*, 5(1), 2018.

Lauer, S. A., Grantz, K. H., Bi, Q., Jones, F. K., Zheng, Q., Meredith, H. R., Azman, A. S., Reich, N. G., and Lessler, J. The incubation period of coronavirus disease 2019 (covid-19) from publicly reported confirmed cases: estimation and application. *Annals of internal medicine*, 172(9):577–582, 2020.

Maalouf, A., Jubran, I., and Feldman, D. Fast and accurate least-mean-squares solvers. In *Advances in Neural Information Processing Systems*, pp. 8305–8316, 2019.

Mathias, R. and Stewart, G. A block qr algorithm and the singular value decomposition. *Linear Algebra and Its Applications*, 182:91–100, 1993.

Meng, X., Saunders, M. A., and Mahoney, M. W. Lsrn: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

Nitanda, A. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, pp. 1574–1582, 2014.

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

Panagiotidis, T., Stengos, T., and Vravosinos, O. On the determinants of bitcoin returns: A lasso approach. *Finance Research Letters*, 27:235–240, 2018.

Pandey, G., Chaudhary, P., Gupta, R., and Pal, S. Seir and regression model based covid-19 outbreak predictions in india. *arXiv preprint arXiv:2004.00958*, 2020.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Phillips, J. M. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

Poulson, J., Marker, B., Van de Geijn, R. A., Hammond, J. R., and Romero, N. A. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software (TOMS)*, 39(2): 1–24, 2013.

Sanil, A. P., Karr, A. F., Lin, X., and Reiter, J. P. Privacy preserving regression modelling via distributed computation. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 677–682, 2004.

Seber, G. A. and Lee, A. J. *Linear regression analysis*, volume 329. John Wiley & Sons, 2012.

Tibshirani, R. Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282, 2011.

Wang, C., Pan, R., Wan, X., Tan, Y., Xu, L., Ho, C. S., and Ho, R. C. Immediate psychological responses and associated factors during the initial stage of the 2019 coronavirus disease (covid-19) epidemic among the general population in china. *International journal of environmental research and public health*, 17(5):1729, 2020.

Zheng, W., Popa, R. A., Gonzalez, J. E., and Stoica, I. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 724–738. IEEE, 2019.

Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.