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

Jyotikrishna Dass and Rabi Mahapatra
Texas A&M University

June 11, 2021

APPENDIX

A. Theorems and Proofs

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

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

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

(Accurate sketch) So, it is possible to replace the original data (X, y) used in existing LMS solvers with (R, Q^Ty) which preserves the covariance $X^TX = R^TQ^TQR = R^TR$ and solves the optimization problem accurately. For example, Ridge regression with ridge parameter λ solves $(X^TX + \lambda I)w = X^Ty$ in primal form which can be reformulated to $(R^TR + \lambda I)w = R^T(Q^Ty)$.

(Memory savings) R is a $d \times d$ upper triangular matrix with $\left(\frac{d(d-1)}{2}\right)$ elements above the diagonal and d on the diagonal resulting in $\left(\frac{d(d+1)}{2}\right)$ elements compared to original data matrix X that has nd elements. Q^Ty is a reflected response vector. It is to be noted that only top d rows of Q^T will be sufficient to compute Q^Ty since $n \gg d$. Hence, reflected response vector (Q^Ty) is of size d compared to the original LMS formulation with response vector y of size n. Hence, the total memory footprint of (R, Q^Ty) is $O\left(\frac{d(d+3)}{2}\right)$ elements which makes it memory-efficient than the original (X, y) occupying $n(d + 1)$ space.

(Time complexity) The above sketch (R, Q^Ty) is computed via Householder QR decomposition (Householder-QR in Step 1 of Algorithm 1) of X which generates upper triangular matrix, R, and orthonormal matrix Q that is internally stored as Householder reflectors. The time complexity of the above decomposition is $O(nd^2 - d^3/3)$ (Golub & Van Loan, 2012). Calculation of Q^Ty is done implicitly by applying Householder reflectors to the response vector y (Multiply-Qc in Step 2 of Algorithm 1) in time $O(nd)$ (Golub & Van Loan, 2012). Hence, it can be seen that the total computation time for the sketch (R, Q^Ty) is $O(nd^2 + nd - d^3/3)$ which results in $O(nd^2)$ for $n \gg d$. □

Theorem 2.3. *Let $X \in \mathbb{R}^{n\times d}$, $y \in \mathbb{R}^n$, $(R, Q^T y) \coloneqq$ HOUSEHOLDER-SKETCH(X, y) that accelerates primal ridge solver via* RIDGE-QR. *Then, $(R, Q^T y)$ is also the Householder sketch for the corresponding Kernel Ridge Regression problem that accelerates the dual problem via* KERNELRIDGE-QR, *and solves it with the same memory and time complexity, independent of data size $\big(n\big)$, as that of primal* RIDGE-QR.

Proof. Kernel Ridge Regression with original data (X, y) solves $(K + \lambda I)\beta = y$, where, $K \in \mathbb{R}^{n\times n}$ is the Kernel matrix, and $\beta \in \mathbb{R}^n$ is the vector of dual variables. For any pair of row vectors in input data, $x_i, x_j \in \mathbb{R}^{1\times d}$, each element of Kernel matrix $K(i, j) = \kappa(x_i, x_j)$, where $\kappa()$ is a Reproducing Kernel Hilbert Space (RKHS) kernel function such that $\kappa(x_i, x_j) = \langle\phi(x_i), \phi(x_j)\rangle$ and $\phi()$ is transformation from input space to RKHS feature space (Burges, 1998).

For a **linear kernel function**, $\kappa(x_i, x_j) = x_i x_j^T$, $K = XX^T$, and the objective is to solve the equation $(XX^T + \lambda I)\beta = y$ such that the model coefficient in input space, $w = X^T\beta$. By applying $X = QR$ via HOUSEHOLDER-QR(X), the above dual problem reformulates to

$$\begin{aligned}
&& (XX^T + \lambda I)\beta = y \\
\Rightarrow && (QRR^TQ^T + \lambda I)\beta = y \\
\Rightarrow && (QRR^TQ^T + \lambda QQ^T)\beta = y \\
\Rightarrow && Q(RR^TQ^T + \lambda Q^T)\beta = y \\
\Rightarrow && (RR^TQ^T + \lambda Q^T)\beta = Q^T y \\
\Rightarrow && (RR^T + \lambda I)\bar{\beta} = \bar{y}
\end{aligned}$$

where, $\bar{y} = Q^T y$ and $\bar{\beta} = Q^T\beta$. The model coefficients in the input space,

$$w = X^T\beta = R^TQ^T\beta = R^T\bar{\beta}$$

Hence, solving $(XX^T + \lambda I)\beta = y$, system of n equations in n unknowns in KERNELRIDGE with original (X, y) is equivalent to solving a much smaller system of d equations in d unknowns $(n \gg d)$, **accurately** and **faster**, with $(RR^T + \lambda I)\bar{\beta} = \bar{y}$ in KERNELRIDGE-QR with memory-efficient $(R, Q^T y)$ sketch. It is worth noting here that once $(R, Q^T y)$ sketch is available, the memory and time complexity for solving *dual* in KERNELRIDGE-QR is **independent** of data size n, and is **same** to that of solving the same problem in *primal* form via RIDGE-QR. Figure 4(a) demonstrates the the above similarity in solving RIDGE-QR, and KERNELRIDGE-QR (with linear kernel) based on computation time. Moreover, KERNELRIDGE-QR calculates the model coefficient w using a triangular matrix in $w = R^T\bar{\beta}$ in d^2 flops compared to $(2n-1)d$ flops for $w = X^T\beta$ in the original KERNELRIDGE with (X, y).

For any **non-linear kernel** function such as Radial Basis Function, it is possible to represent $K \approx AA^T$ with some low-rank matrix, $A \in \mathbb{R}^{n\times k}$ via any kernel approximation techniques (Williams & Seeger, 2001; Si et al., 2017). This can be followed by constructing memory-efficient $(R, Q^T y) \coloneqq$ HOUSEHOLDER-SKETCH(A, y) from Algorithm 1. Now, solving the approximated *dual* problem formulation for non-linear kernels via KERNELRIDGE-QR is equivalent in space and time complexity to solving the approximated problem in *primal* form via RIDGE-QR on $(R, Q^T y)$. Moreover, any of the above RIDGE-QR or KERNELRIDGE-QR is faster than solving the *primal* form via RIDGE with (A, y).

Hence, $(R, Q^T y)$ is also the Householder sketch for Kernel Ridge Regression , where, R is defined based on linear or non-linear kernel, for accelerating the *dual* problem via KERNELRIDGE-QR. ☐

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

Proof.

$$X = \begin{pmatrix} X_1 \\ X_2 \\ . \\ . \\ X_p \end{pmatrix} = \begin{pmatrix} Q_1R_1 \\ Q_2R_2 \\ . \\ . \\ Q_pR_p \end{pmatrix} = \mathtt{diag}(Q_1,\dots,Q_p) \begin{pmatrix} R_1 \\ R_2 \\ . \\ . \\ R_p \end{pmatrix}$$

Let us define, $R_{stack} = \begin{pmatrix} R_1 \\ R_2 \\ . \\ . \\ R_p \end{pmatrix} = Q_M R_M$, via Householder-QR in Algorithm 1 (or Theorem 2.1). Then,

$$X = \mathtt{diag}(Q_1,\dots,Q_p)R_{stack} = \mathtt{diag}(Q_1,\dots,Q_p)Q_M R_M$$

Also, it is given that $X = QR$ via Householder-QR on complete matrix X. Hence, $Q = \mathtt{diag}(Q_1,\dots,Q_p)Q_M$, is the orthogonal matrix, and, $R = R_M$ is the upper triangular matrix.

Time complexity and Communication volume. For a given local data $X_i \in \mathbb{R}^{\hat{n}\times d}$, where, $n = \hat{n}p$, each $X_i = Q_i R_i$ at $i-$th parallel worker is computed via local Householder-QR (as per Algorithm 1, and Theorem 2.1). From Theorem 2.2, each local Householder-QR takes $O(\hat{n}d^2 - d^3/3)$, in parallel for all the workers. Subsequently, $R_{stack} = Q_M R_M$ is performed via master Householder-QR in time $O(\times pd \times d^2 - d^3/3)$, where, $R_{stack} \in \mathbb{R}^{pd\times d}$ is obtained by *gathering* (communicating) local upper-triangular matrices $R_i \in \mathbb{R}^{d\times d}$, i.e., $\big(\frac{d(d+1)}{2}\big)$ elements from each parallel worker $i = 1,\dots,p$ to the master ($i = 1$). Hence, total computation time for Distributed Householder-QR is $O(\hat{n}d^2 + pd^3 - 2d^3/3)$ or $O(\frac{n}{p}d^2)$ for $\hat{n} \gg d$ (i.e. $n \gg pd$). It is worth noticing that the above computational time is dominant by local Householder-QR as observed in Figure 2(a). □

Corollary 3.1.1 (Distributed Multiply-Qc). *Let $c = (c_1^T | \dots | c_p^T)^T \in \mathbb{R}^n$, where, $c_i \in \mathbb{R}^{\hat{n}}$ be some local vector at parallel worker with local data matrix X_i, $i = 1, \dots, p$, where $\hat{n} \gg d$, and, $n = p\hat{n}$. Let, orthogonal matrices Q_M, and Q_i, $i = 1, \dots, p$ be constructed via* Distributed Householder-QR *as per Theorem 3.1 such that $Q = \mathit{diag}(Q_1, \dots, Q_p)Q_M$. 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 $Q^T c = Q_M^T\big((Q_1^T c_1)^T | \dots | (Q_p^T c_p)^T\big)^T$ or, $Qc = \mathit{diag}(Q_1, \dots, Q_p)Q_M(c_1^T | \dots | c_p^T)^T$. The above* Distributed Multiply-Qc *has a computational time complexity of $O\big(\frac{n}{p}d + pd^2\big)$, with a communicated data volume of $\big(d\big)$ elements by each worker.*

Proof. From Theorem 3.1, for $X = (X_1^T | \dots | X_p^T)^T$, its corresponding orthogonal matrix $Q = \mathtt{diag}(Q_1, \dots, Q_p)Q_M$. Hence,

$$Q^T = Q_M^T \mathtt{diag}(Q_1^T, \dots, Q_p^T)$$

For a given vector $c \in \mathbb{R}^n$, $Q^T c$ via Multiply-Qc (Step 2 in Algorithm 1) can be equivalently computed from $c = (c_1^T | \dots | c_p^T)^T$ comprising local vector $c_i \in \mathbb{R}^{\hat{n}}$, where, $i = 1, \dots, p$, as follows

$$Q^T c = Q_M^T \mathtt{diag}(Q_1^T, \dots, Q_p^T) c = Q_M^T \begin{pmatrix} Q_1^T c_1 \\ \\ Q_2^T c_2 \\ \cdot \\ \cdot \\ \\ Q_p^T c_p \end{pmatrix}$$

In Distributed Multiply-Qc algorithm, the above is implemented as follows. Each worker, $i = 1, \dots, p$, computes its local reflected vectors $Q_i^T c_i \in \mathbb{R}^d$ via Multiply-Qc (refer Step 2 in Algorithm 1) in parallel with time $O(2\hat{n}d)$ as shown in Theorem 2.2. Once these local reflected vectors, each of size d elements are *gathered* (communicated) from each worker to the master, a stacked vector $\Big((Q_1^T c_1)^T | \dots | (Q_p^T c_p)^T\Big)^T \in \mathbb{R}^{pd \times d}$ is constructed. Then, a master Multiply-Qc is applied on this stacked vector using Q_M^T in time $O(2 \times pd \times d)$, i.e., $O(2pd^2)$. Hence, total computation time of Distributed Multiply-Qc is $O(2\hat{n}d + 2pd^2)$, i.e., $O(\frac{n}{p}d + pd^2)$, since $n = \hat{n}p$. ∎

B. Figures

For more clarity, we provide enlarged figures from Section 4 (Experiments and Results) of the main paper. Following is the organization of figures in the supplementary document.

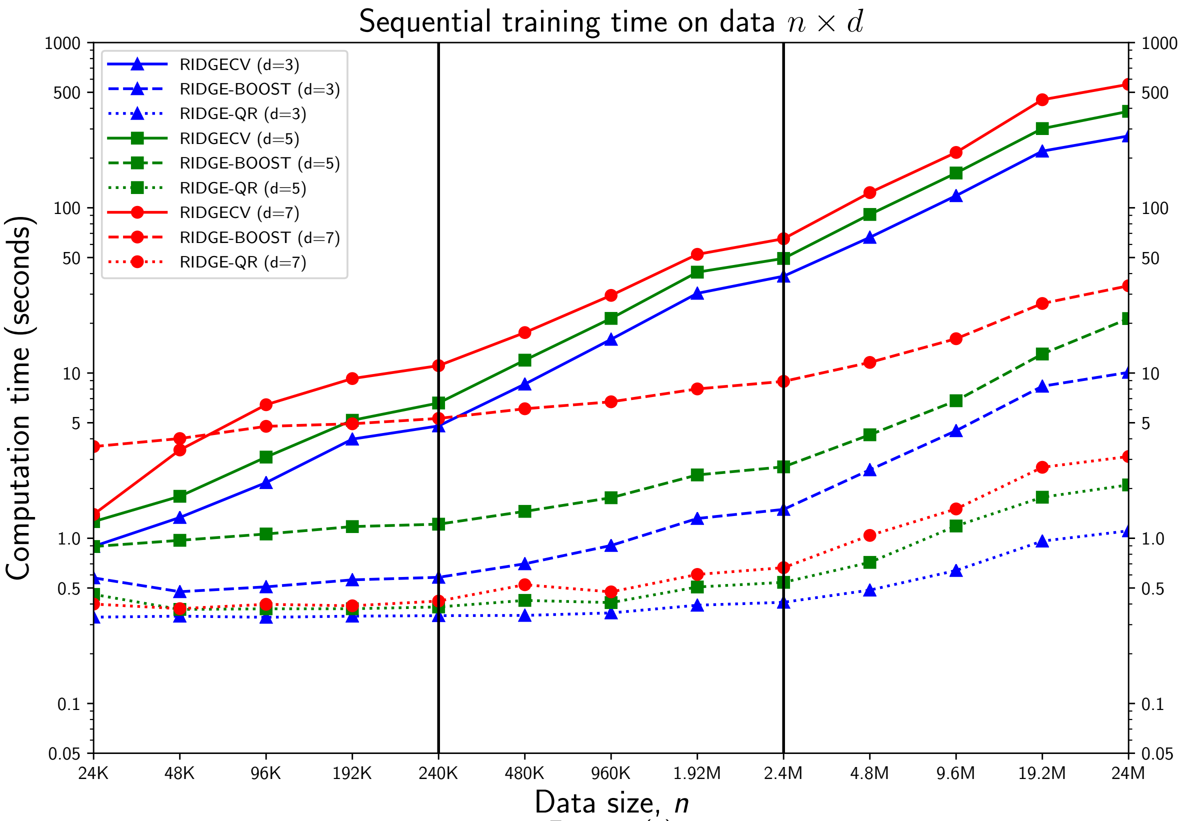

Figure 1 (a)

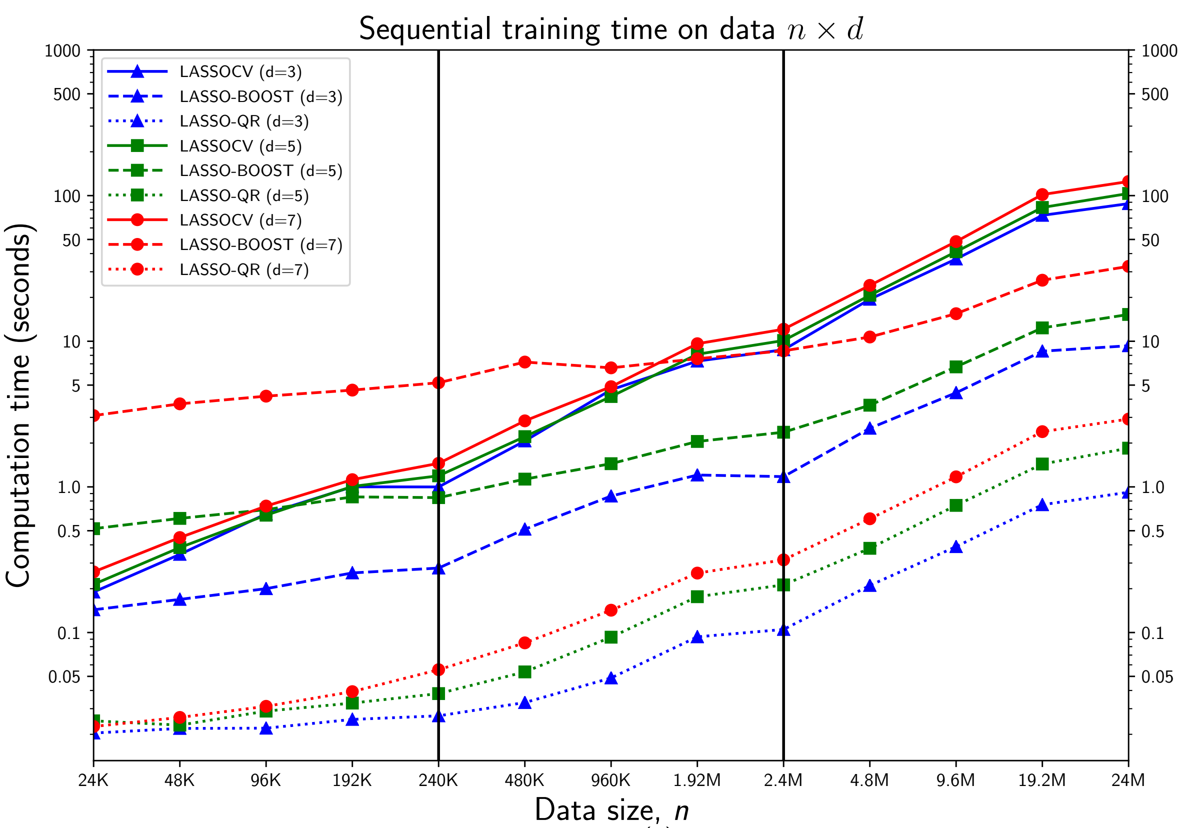

Figure 1 (b)

Figure 1: Sequential training time (a)Ridge (b)LASSO

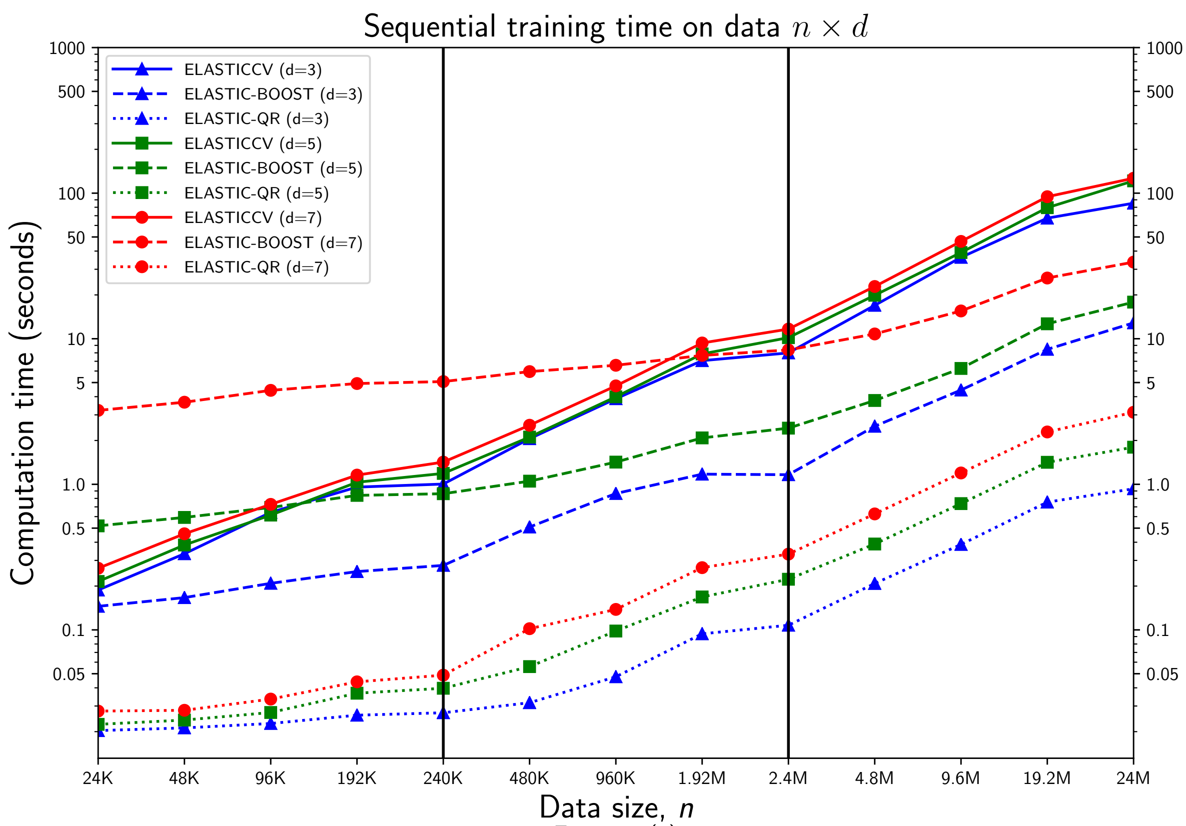

Figure 1 (c)

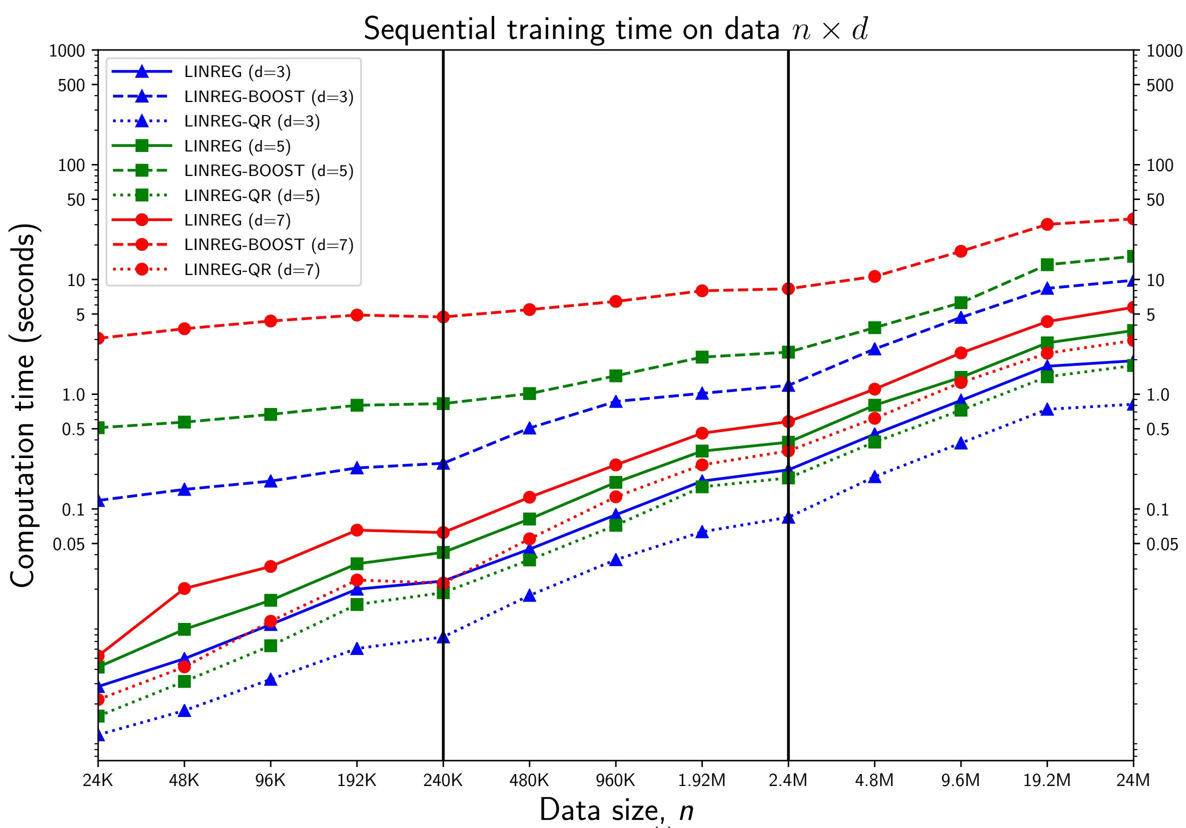

Figure 1 (j)

Figure 1: Sequential training time (c)Elastic-net (j)Linear Regression

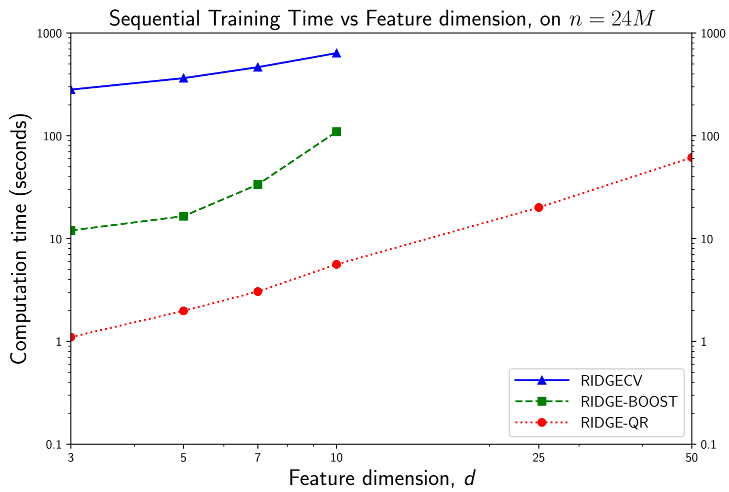

Figure 1 (d)

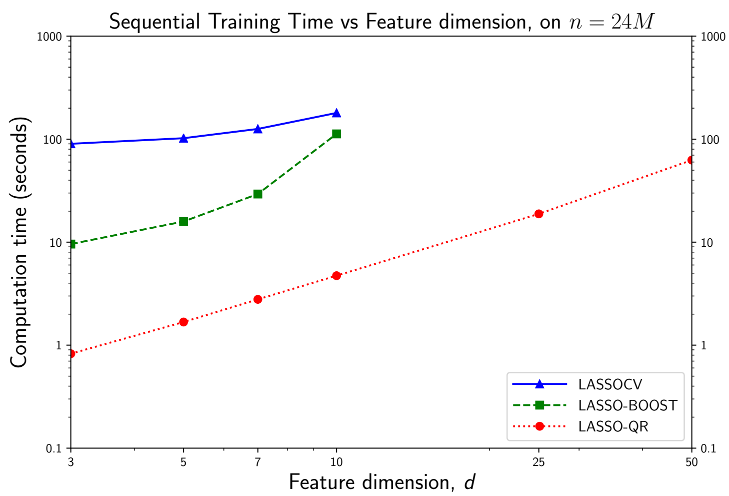

Figure 1 (e)

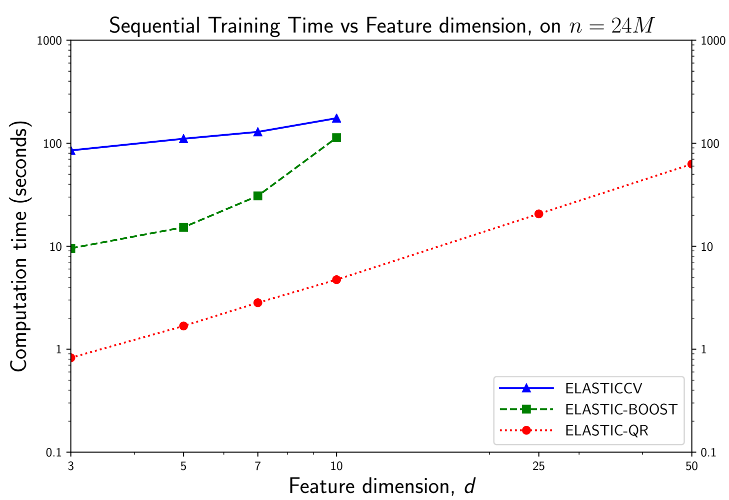

Figure 1 (f)

Figure 1: Sequential training time on $n = 24M$, and various feature dimension $d = \{3, 5, 7, 10, 25, 50\}$ (d) Ridge, (e) LASSO, (f) Elastic-net

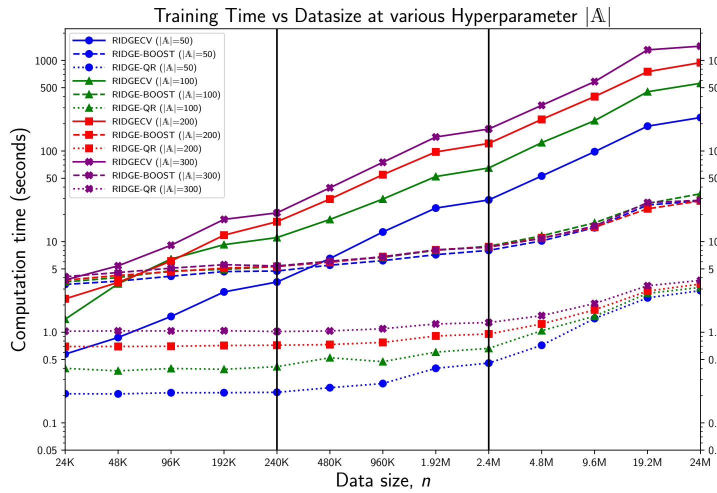

Figure 1 (g)

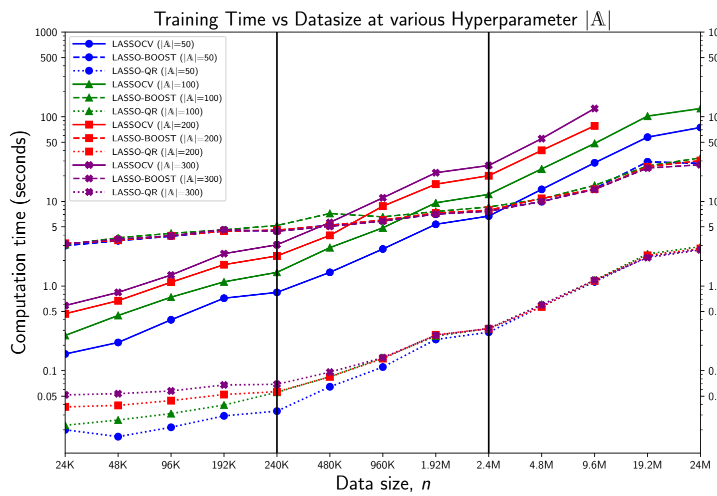

Figure 1 (h)

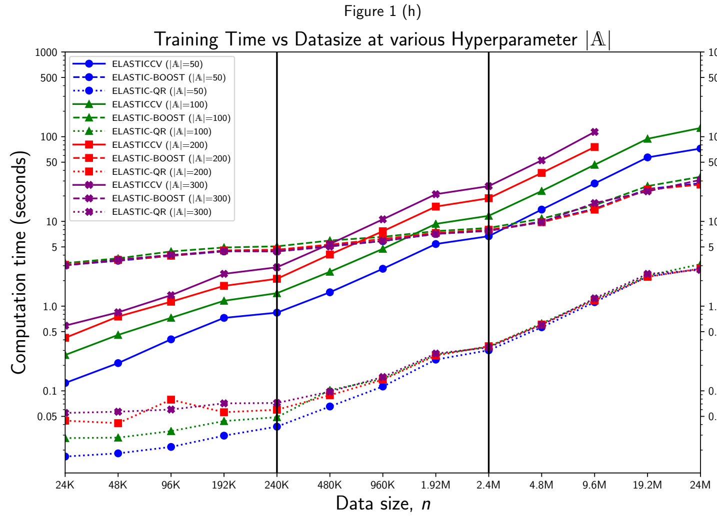

Figure 1 (i)

Figure 1: Sequential training time for various hyper-parameter set size $|\mathbb{A}|$ (g) Ridge, (h) LASSO, (i) Elastic-net

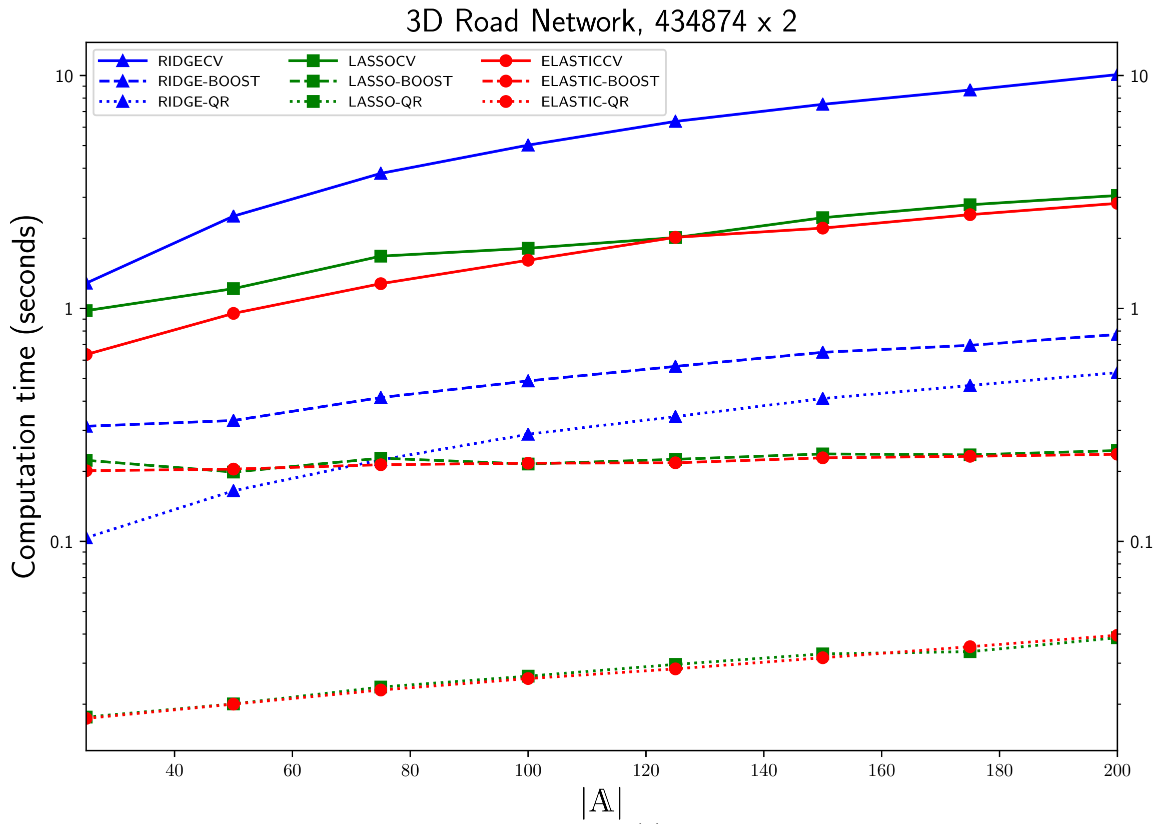

Figure 1 (k)

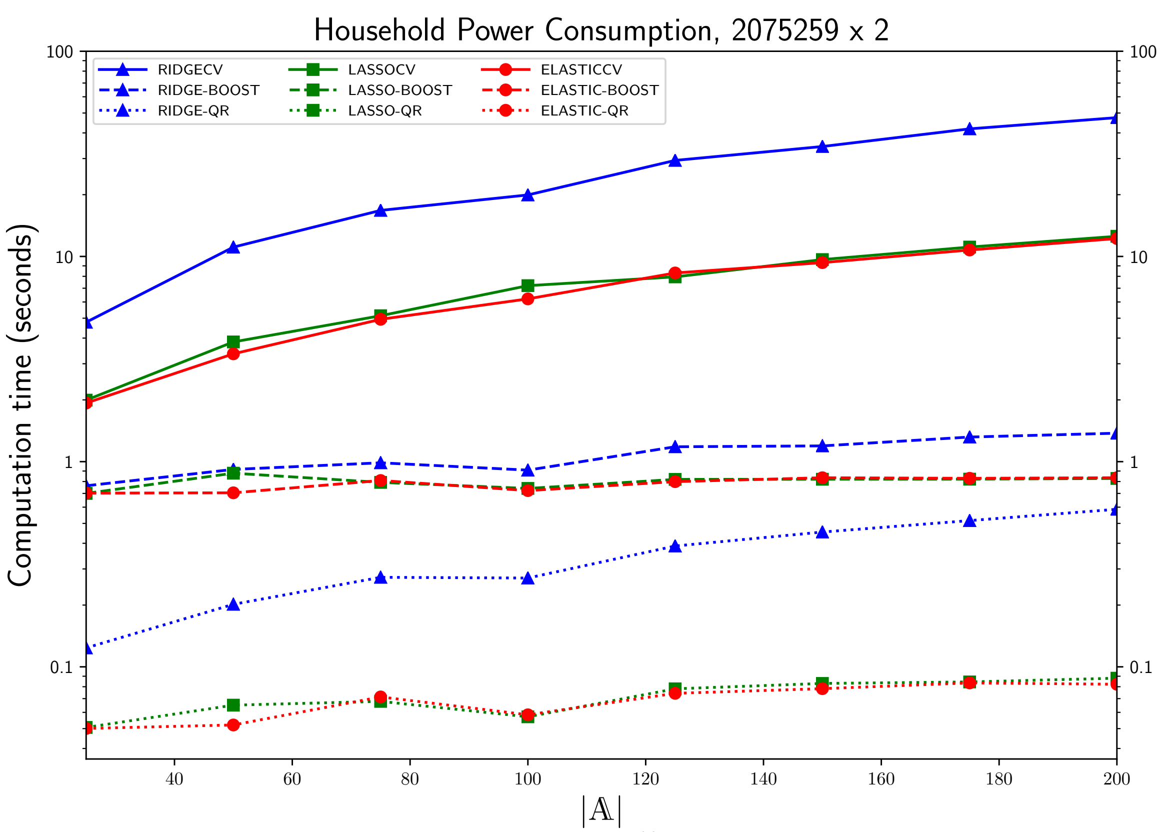

Figure 1 (l)

Figure 1: Sequential training time for various hyper-parameter set size $|\mathbb{A}|$ (k) 3D Road Network dataset (l) Household Power Consumption dataset

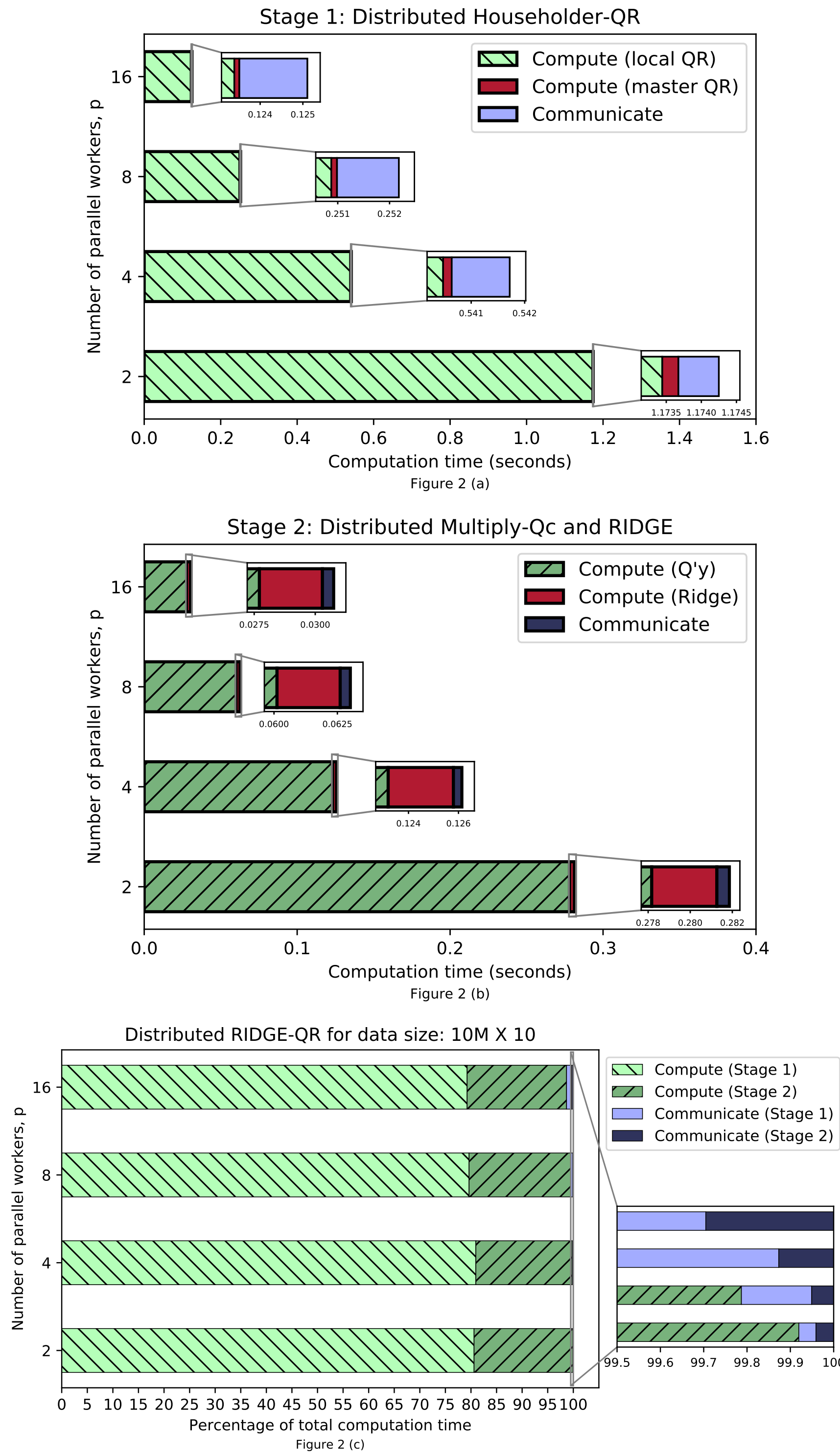

Figure 2: Training time analysis for Distributed Ridge-QR with zoomed insets depicting communication time (a): Stage 1: Distributed Householder-QR timings, (b): Stage 2: Distributed Multiply-Qc and Ridge solver timings, (c): Combined timing percentage spent on each stage for computation and communication

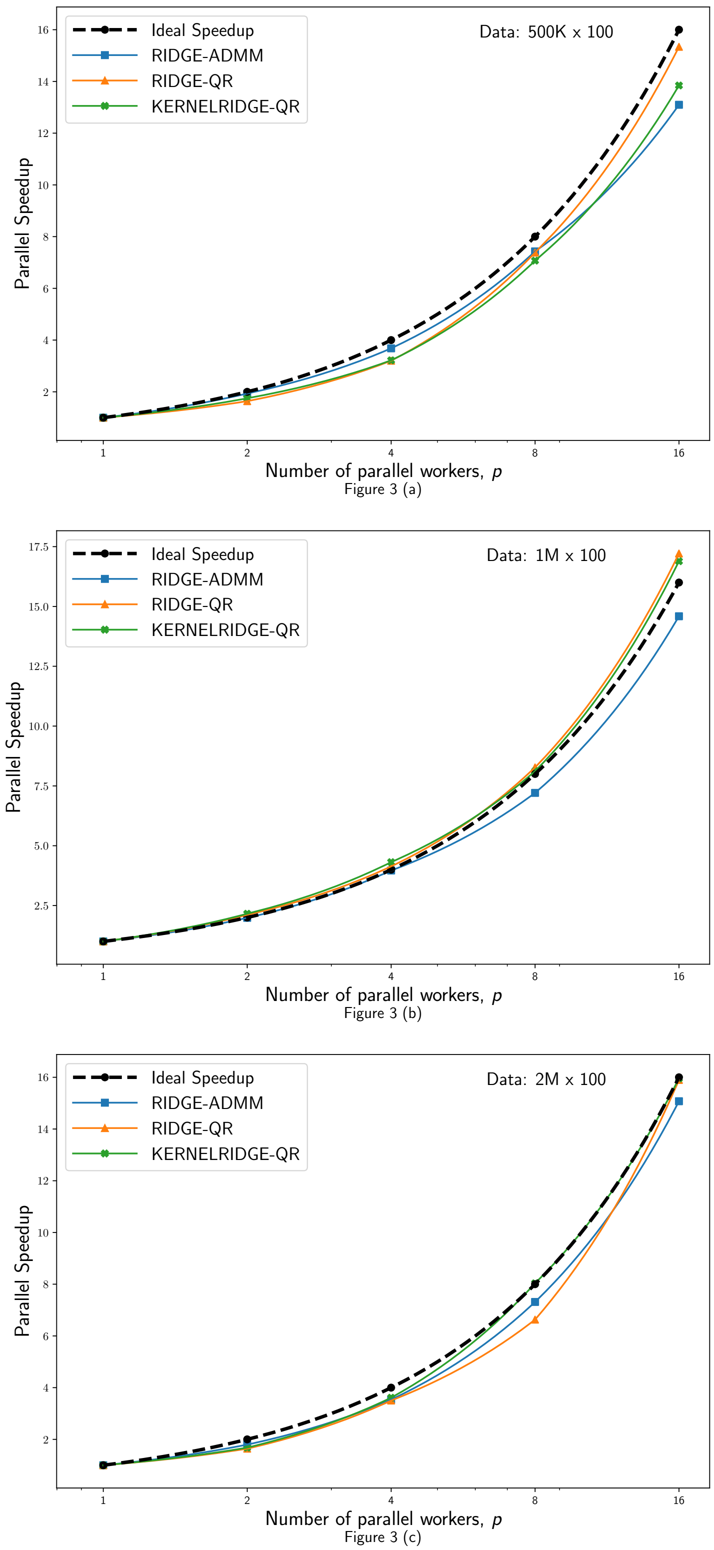

Figure 3: (**Scalability**) Parallel speedup for Distributed Ridgr-QR on synthetic datasets of size (a) $500K \times 100$ (b) $1M \times 100$ (c) $2M \times 100$

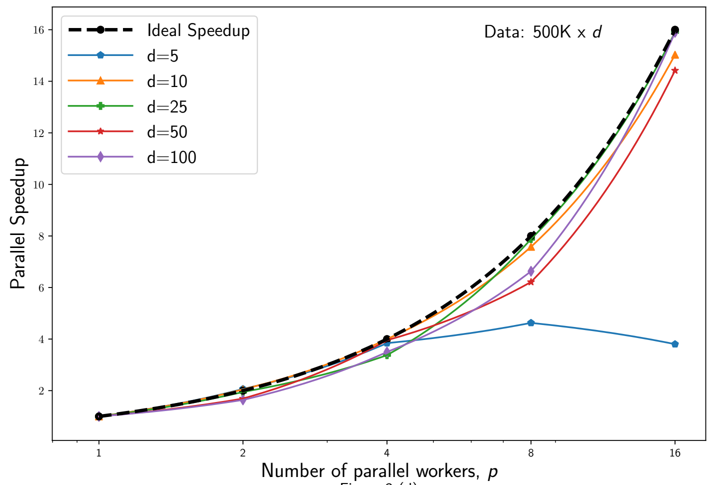

Figure 3 (d)

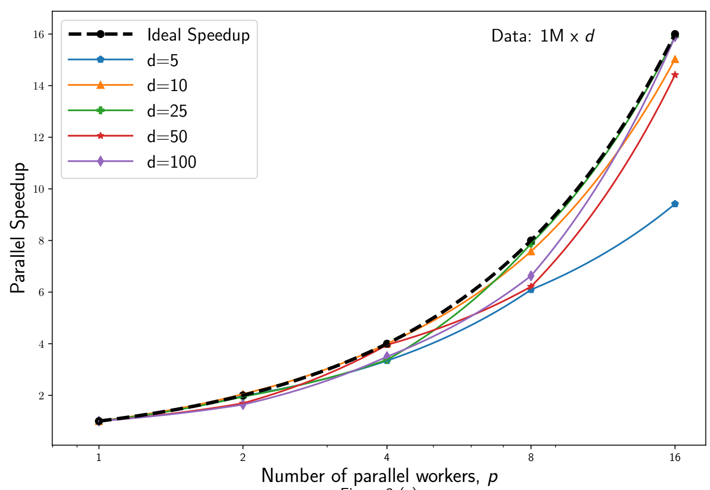

Figure 3 (e)

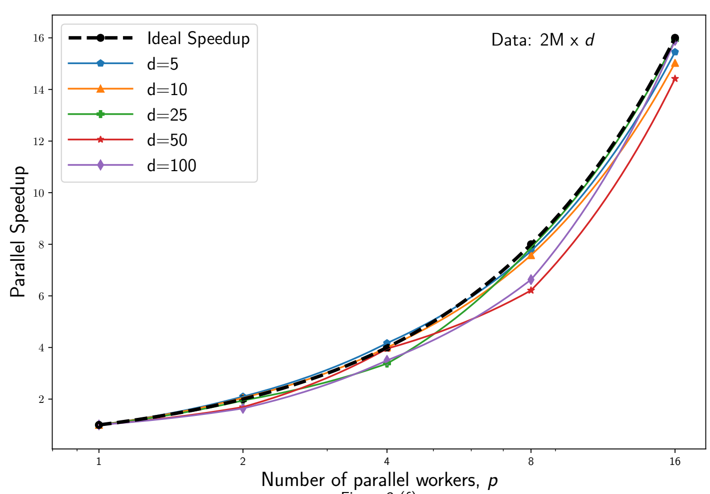

Figure 3 (f)

Figure 3: (**Scalability**) Parallel speedup for DISTRIBUTED RIDGR-QR on synthetic datasets for various feature dimension size $d = \{5, 10, 25, 50, 100\}$ (d) $500K \times d$ (e) $1M \times d$ (f) $2M \times d$

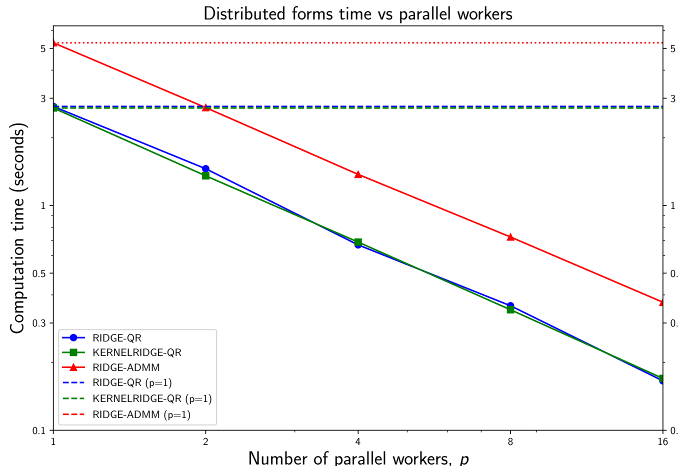

Figure 4 (a)

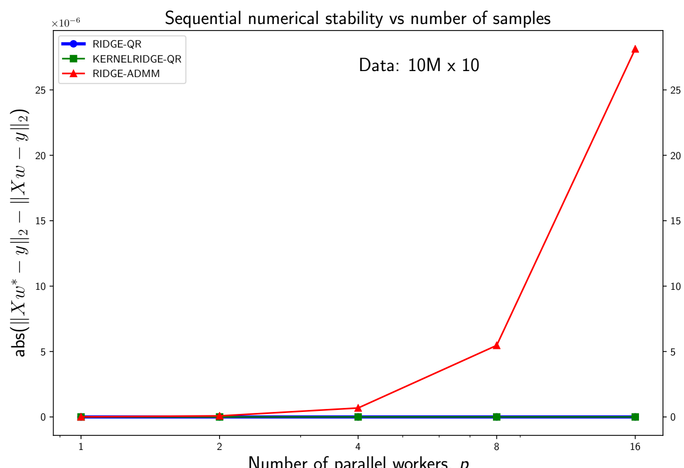

Figure 4 (b)

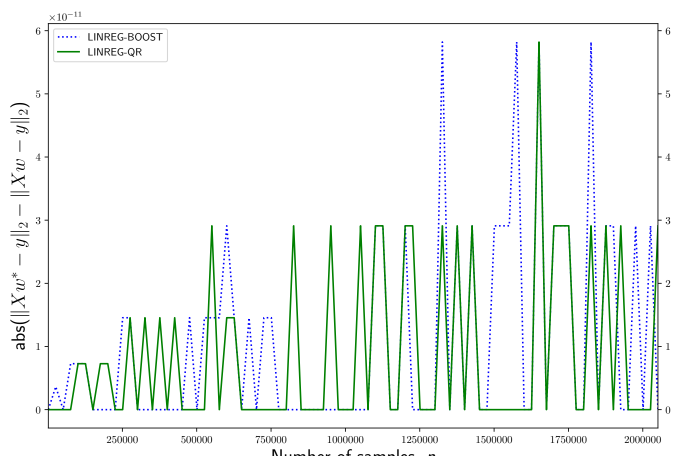

Figure 4 (c)

Figure 4: (a): Comparing distributed implementations of RIDGE-QR, KernelRidge-QR (linear kernel), and Ridge-ADMM for $10M \times 10$ synthetic data based on (a) Computation time (b) Accuracy ($\times 10^{-6}$), w^* comparison of Ridge-QR and Ridge-Boost, w^* is solution from scikit-learn Ridge. (c) Accuracy ($\times 10^{-11}$) comparison of LinReg-QR and LinReg-Boost on Household Power Consumption dataset ($\sim 2M \times 8$), w^* is solution from scikit-learn LinearRegression

C. Algorithms

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

$(\mathcal{V}, R) \coloneqq$ HOUSEHOLDER-QR(X) // see Theorem 2.1, Algorithm 4
$\bar{y} \coloneqq$ MULTIPLY-QC$(\mathcal{V}, y, \text{'T'})$ // implicit Q^Ty, see (Golub & Van Loan, 2012), see Algorithm 5
$R \leftarrow R[0:d,:]$ // $d \times d$ triangular block
$\bar{y} \leftarrow \bar{y}[0:d]$ // top d elements
return $(R, \bar{y})$

Algorithm 2: LMS-QR(X, y, 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, $w \in \mathbb{R}^d$

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

Algorithm 3: DISTRIBUTED LMS-QR(p, X, y, 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, $w \in \mathbb{R}^d$

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

Algorithm 4: $(\mathcal{V}, R) \leftarrow X$, via Householder-QR, refer Theorem 2.1

Input: A matrix $X \in \mathbb{R}^{n \times d}$
Output: Householder reflector set $\mathcal{V}$, Upper trapezoidal matrix $R \in \mathbb{R}^{n \times d}$

for $j \leftarrow 1$ *to* d **do**
  $v_j \leftarrow X(j:n, j)$
  $v_j(1) \leftarrow v_j(1) + sign(v_j(1)) \times \|v_j\|_2$ `// scalar update`
  $v_j \leftarrow \frac{v_j}{\|v_j\|_2}$ `// vector normalization`
  $X(j:n, j:d) \leftarrow X(j:n, j:d) - 2 \times v_j < v_j,\ X(j:n, j:d) >$
  $R = X(j:n, j:d)$
end
$\mathcal{V} \leftarrow [v_1, v_2, \ldots, v_d]$ `// set of d-reflectors`
return $(\mathcal{V}, R)$

Algorithm 5: Computing implicit $Q^T y$ via Multiply-Qc

Input: Householder reflector set $\mathcal{V}$, a vector $y \in \mathbb{R}^n$
Output: $\bar{y} \leftarrow \left(Q^T y\right) \in \mathbb{R}^n$

$c \leftarrow y$
for $j \leftarrow 1$ *to* d **do**
  $c(j:n) \leftarrow c(j:n) - 2 \times v_j(v_j^T\ c(j:n))$
end
$\bar{y} \leftarrow c$
return $\bar{y}$

References

Burges, C. J. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2 (2):121–167, 1998.

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.

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

Si, S., Hsieh, C.-J., and Dhillon, I. S. Memory efficient kernel approximation. *The Journal of Machine Learning Research*, 18(1):682–713, 2017.

Williams, C. K. and Seeger, M. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pp. 682–688, 2001.