Distributed QR decomposition framework for training Support Vector Machines

Jyotikrishna Dass, V.N.S. Prithvi Sakuru, Vivek Sarin and Rabi N. Mahapatra

Department of Computer Science & Engineering
Texas A&M University, College Station, Texas 77840
{dass.jyotikrishna, prithvi.sakuru, sarin, rabi}@tamu.edu

Abstract—Support Vector Machines (SVM) belong to a class of supervised machine learning algorithms with applications in classification and regression analysis. SVM training is modeled as a convex optimization problem that is computationally tedious and has large memory requirements. Specifically, it is a quadratic programming problem which scales rapidly with the training set size rather than the dimensionality of the feature space. In this work, we first present a novel QR decomposition framework (QRSVM) to efficiently model and solve a large scale SVM problem by capitalizing on low-rank representations of the full kernel matrix rather than solving the problem as a sequence of smaller sub-problems. The low-rank structure of the kernel matrix is leveraged to transform the dense matrix into one with a sparse and separable structure. The modified SVM problem requires significantly lesser memory and computation. Our approach scales linearly with the training set size which makes it applicable to large datasets. This motivates towards our another contribution; exploring a distributed QRSVM framework to solve large-scale SVM classification problems in parallel across a cluster of computing nodes. We also derive an optimal step size for fast convergence of the dual ascent method which is used to solve the quadratic programming problem.

I. INTRODUCTION

Support Vector Machine (SVM) is a popular machine learning technique to solve classification and regressions problems. With the growing data sizes, solving large scale machine learning problems using SVM is a tedious task both in terms of computational cost as well as memory resources. Various decomposition techniques like Vapnik’s chunking [1], Osuna’s decomposition [2] and Platt’s SMO [3] have made respectable contribution to addressing these issues.

To efficiently solve large-scale classification problems, research community nowadays is more focused on modeling it as a linear SVM problem which is a simplified optimization framework [4]. In the linear SVM problems, the training data is used in its original feature space, thereby, enabling the adoption of coordinate gradient methods to solve the optimization problem. In comparison, for the non-linear SVM problems, the data is transformed to higher dimensional space by using kernel trick. Since the higher dimensional coordinates are not explicitly computed, applying coordinate gradient methods for non-linear kernel problems is infeasible.

In coordinate gradient methods, a single data point contributes to the gradient shift of the objective function iteratively and hence, they have very low per-iteration computation cost. However, these methods have slow convergence rate when compared to the orthodox gradient methods like dual ascent [5], Zhang et al. [6], Shalev-Shwartz et al. [7], Bottou [8] proposed variations of stochastic gradient descent on the primal SVM form. Joachims [9] proposed the cutting plane method while Smola et al. [10] used bundle methods on dual SVM. Hsieh et al. [11] proposed the dual coordinate descent (DCD) method and showed that it outperformed other dual methods. LIBLINEAR [12] is a state of the art library for solving linear SVM classification (SVC) problems. It is implemented using the DCD method. However, this method was really slow and not completely stable for non-document datasets especially with low dimensions [13].

For solving kernel SVM for non-linear decision boundaries, efficient sequential algorithms have been proposed. The most popular of such algorithms is SMO [3] which now finds its implementation in widely used SVM solver tool LIBSVM [14]. Due to the sequential nature of SMO algorithm, single machine kernel SVM solvers suffer from limited scalability, thereby making those unfit for training big-scale datasets. With growing data sizes, it has become utmost necessary to design parallel and distributed algorithms to train the kernel SVMs.

The state of the art parallel algorithms which have been proposed to solve kernel SVM problems are PSVM [15] and P-pack SVM [16]. PSVM, in fact, has limited scalability due to its quadratic dependence on the sample size. Moreover, PSVM works with kernel matrix approximation using Incomplete Cholesky Factorization (ICF) which lacks theoretical error bounds. Many kernel approximation techniques like Nyström [17], LLSVM [18] and random Fourier features [19] have also been explored to reduce the problem dimension. Such methods produce low-rank approximations of the kernel matrix and have been used to reduce the kernel SVM problem into a linear SVM problem.

Novelty: We propose a QR decomposition framework (QRSVM) to efficiently solve large-scale kernel SVM problems with guaranteed stability of dual ascent method. We use state of the art kernel approximation technique that is memory efficient unlike Singular Value Decomposition (SVD). The low-rank approximation of the kernel matrix is represented in a separable form which makes it fit to be used in the proposed QRSVM framework. Rather than proceeding with the approximated kernel matrix directly, we further decompose it using QR factorization. This leads to memory-
efficient representation of the otherwise dense hessian matrix in SVM problem. The subsequent representation is sparse that it can be block-partitioned for parallel SVM training. The proposed QRSVM scales linearly with the data size which further motivates towards a distributed framework that can work on data that has been partitioned across a cluster of computing nodes. Within this, we first present a distributed QR decomposition technique and then parallelize the iterative update steps of the dual ascent method to solve the parallel kernel SVM problem. We also propose an optimal step size for fast convergence of the dual ascent method.

We evaluate the proposed sequential QRSVM framework in comparison to other state-of-the-art sequential solver LIBLINEAR for HIGGS dataset (10.5 million samples) for relative convergence. We also validate the linear scalability of QRSVM with a number of training samples. For the distributed version of the QRSVM framework, we train non-linear classifiers for benchmarks a9a and covtype and report their computation and communication timings. Finally, we compare the proposed distributed QRSVM with PSVM and P-packSVM.

The rest of the paper is organized as follows. In Section II, we introduce the SVM problem and its formulation for both linear and non-linear classification. Then, we lay down the motivation for the proposed QRSVM framework. The detailed formulation of the QRSVM framework is presented in Section III. Here, we also discuss the benefits achieved by using our proposed framework. Next, in Section IV we list down the update steps of the dual ascent algorithm to train the SVM problem with guaranteed convergence. We also propose an optimal step size formulation for faster convergence of the algorithm in fewer iterations. We wind up this section by analyzing the time complexity of our proposed QRSVM framework. In Section V, we present a distributed version of the QRSVM framework which comprises a distributed QR decomposition technique and a parallel dual ascent algorithm. Section VI reports the experiments and evaluation results. Finally, we conclude in Section VII by laying down the possible future research directions.

II. SVM CLASSIFICATION

A. Generic SVM formulation

In a typical binary Support Vector Classification (SVC) problem, one is given a training dataset \( S = \{(x_i, y_i), i = 1, ..., n\} \) where input data point \( x_i \in \mathbb{R}^d \) feature space and their corresponding data label \( y_i \in \{-1, 1\} \). The objective of the problem is to find the hyperplane with the maximum separation margin. An \( \ell_2 \)-regularized version this problem is

\[
\min_w \frac{1}{2}||w||_2^2 + C \sum_{i=1}^{n} \xi_i(w; x_i; y_i) \tag{1}
\]

where \( w \in \mathbb{R}^d \) represents the normal to the hyperplane separating the data points and \( C > 0 \) is the penalty parameter. \( \xi_i(w; x_i; y_i) \) represents the loss function associated with the optimization problem. The common loss functions are \( \ell_1 \)-loss (L1-SVM) or \( \ell_2 \)-loss (L2-SVM). The scope of this paper includes a detailed discussion of solving L2-SVM via QRSVM framework. L2-SVM uses a squared hinge loss function which is strongly convex and smooth. Hence, it is easily differentiable and can be used in dual ascent method. However, the similar methodology can be extended to L1-SVM.

\[
\ell_2 \text{-loss : } \xi_i(w; x_i; y_i) = \max(0, 1 - y_i w^T x_i)^2
\]

A bias term \( b \) is typically associated with \( w \) representing the separating hyperplane. For following conventions, we include this bias in \( w \) vector itself by adding an additional dimension to the input data instance i.e.,

\[
x_i \leftarrow [x_i; 1] \iff w \leftarrow [w; b]
\]

The optimization problem in equation (1) is referred to as the primal SVM. Often, it is preferable to solve the problem using its dual form which is also a quadratic problem. Advantages for solving the dual form of SVM over the primal form are:

1) In dual form, one can use the “kernel trick” to classify data that is not linearly separable in the original feature space.

2) The dual problem gives the weights for all data points contributing to the classifier, popularly called “support vectors”. Such insights into the given data can not be directly obtained from primal solvers.

3) The loss function present in the primal formulation vanishes from the objective function on using the dual formulation, thereby, making it simpler.

The dual formulation of equation (1) is:

\[
\min_{\alpha} \frac{1}{2} \alpha^T Z \alpha + e^T \alpha \tag{2}
\]

subject to \( L \leq \alpha_i \leq U \)

where, \( Z = (G + D) \in \mathbb{R}^{n \times n} \) is dense and symmetric, \( e = -1_n \), \( L \) is the lower bound of each dual variable \( \alpha_i \) and \( U \) is its upper bound. Elements of \( G, g_{ij} = y_i y_j \kappa(x_i, x_j) \), where \( \kappa() \) represents the Mercer kernel function and \( K = \{\kappa(x_i, x_j), \forall i, j = 1, ..., n\} \) is a positive semi-definite matrix. Diagonal matrix \( D \), lower bound \( L \), and upper bound \( U \) are dependent on the type of loss function associated with the SVM problem. For L2-SVM, \( D = (1/2C)I_n \), \( L = 0 \) and \( U = \infty \).

For a linear SVC problem, the kernel function \( \kappa(x_i, x_j) = \langle x_i, x_j \rangle \) and \( K = XX^T \) where \( X = \{x_i \in \mathbb{R}^d, i = 1, ..., n\} \). L2-SVM formulation in equation (2) using linear kernel can be re-written as:

\[
\min_{\alpha} \frac{1}{2} \alpha^T (XX^T) \alpha + \frac{1}{2} \alpha^T \left( \frac{1}{2C} I_n \right) \alpha + e^T \alpha \tag{3}
\]

subject to \( -I_n \alpha \leq \theta_n \)
where, $\tilde{X} = \text{diag}(y) \times X$ and $y = \{y_i \in \{-1, 1\}, i = 1...n\}$. L2-SVM provides a simpler constraint formulation which specifies that each $\alpha_i$ corresponding to each data $x_i$ must be non-negative. The data points corresponding to positive $\alpha_i$’s are the support vectors. It should be noted that for the class of non-linear SVC problems, the above formulation in equation (3) is not directly possible. We will be focusing on radial basis function (rbf) as the non-linear kernel function i.e.

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \exp(-\gamma \|x_i - x_j\|^2)$$

where, $\phi()$ is a mapping, generally not known, and it transforms the original data $x_i$ from input space to the Reproducing Kernel Hilbert Space (RKHS) (for rbf kernel, the data is transformed to infinite dimension) where it can be linearly separable. Unlike the case for linear SVC, kernel matrix $K$ is not trivially separable for non-linear problems. Moreover, kernel matrix is associated with high computation $O(n^2)$ and dense storage $O(n^2)$ which makes it challenging to scale. A popular solution to these problems is low-rank kernel approximation which speeds up the kernel based solvers while consuming limited memory.

Most of the kernel approximation techniques propose to find the best rank-$p$ approximation $K \approx AA^T$ of the kernel matrix $K$, with $A \in \mathbb{R}^{n \times p}$ and $p \ll n$. An added benefit of such approximation is that the non-linear kernel matrix can now be written in a separable form just like the case for the linear, where, $K = X X^T$. Similarly, we can define $\hat{A} = \text{diag}(y) \times A$ for non-linear SVC problem and rewrite the cost function in equation (3) as

$$\min_{\alpha} \frac{1}{2} \alpha^T \left(\hat{A}^T \hat{A}\right) \alpha + \frac{1}{2} \alpha^T \left(\frac{1}{2C} I_n\right) \alpha + e^T \alpha$$

We use MEKA [20]; memory efficient kernel approximation technique which uses a nearly same amount of storage as other approximation techniques like incomplete Cholesky decomposition [21], Greedy basis selection techniques [22] and Nyström [23] methods, while achieving lower approximation error.

### B. Motivation for QRSVM

Matrices $A, \hat{A} \in \mathbb{R}^{n \times p}$ with $p \ll n$, have a tall and skinny (TS) structure. For SVC problem involving large $n$, $\hat{A} \hat{A}^T$ is a dense coefficient matrix requiring $O(n^2)$ memory which is tightly coupled and hence can’t be decomposed into independent sub-problems of smaller size. Working on the whole $n \times n$ data leads to large computation time during every iteration of the model learning phase. Moreover, $\hat{A} \hat{A}^T$ for SVC is a rank deficient matrix, hence non-invertible. As a result, minimizing the Lagrangian of the dual SVC problem in equation (6) which solves a system of linear equation can become unstable for large $C$. To address the above issues, we propose QRSVM framework comprising a) QR decomposition technique to efficiently transform the dense coefficient matrix into a sparse form and b) dual ascent method to solve the above optimization problem comparatively faster than LIBLINEAR. It should be noted that we can use it for non-linear SVM problems too.

### III. PROPOSED QRSVM

#### A. $\ell_2$-QRSVM formulation

QR decomposition on a TS- dense $\hat{A}$ factorizes it into two matrices, namely, $Q$ and $R$ i.e., $\hat{A} = QR$. Here, $Q$ is an orthogonal matrix of size $n \times n$ and $R$ is an upper triangular matrix of size $n \times p$. The cost function of the non-linear SVC problem in the above equation now formulates to

$$\min_{\alpha} \frac{1}{2} \alpha^T \left(Q R R^T Q^T\right) \alpha + \frac{1}{2} \alpha^T \left(\frac{1}{2C} I_n\right) \alpha + e^T \alpha$$

Defining $\hat{\alpha} = Q^T \alpha$, $\hat{e} = Q^T e$ and using $Q^T Q = I_n$, the L2-SVM quadratic programming (QP) problem becomes

$$\min_{\hat{\alpha}} \frac{1}{2} \hat{\alpha}^T \left(R R^T + \frac{1}{2C} I_n\right) \hat{\alpha} + (\hat{\epsilon})^T \hat{\alpha}$$

subject to $-Q \hat{\alpha} \leq 0_n$.

#### B. Benefits of QRSVM

In equation (4), $RR^T$ is a symmetric sparse matrix of size $n$ where the first $p \times p$ submatrix is dense while the rest of the elements in the matrix are all zeros. In other words, coefficient matrix of the quadratic term in equation (4) is a block diagonal matrix comprising two diagonal blocks:

1. a $p \times p$ symmetric and dense submatrix,

   $$(RR^T)_p + (1/2C) I_p$$

2. a diagonal submatrix $(1/2C) I_{n-p}$.

The benefits of the proposed QRSVM formulation are listed below.

**Sparsity**: We have transformed a dense $n \times n$ quadratic coefficient matrix $\hat{A} \hat{A}^T + (1/2C) I_n$ in equation (3) to a sparse matrix as in equation (4) which consists of a small dense $p \times p$ block, as illustrated in Figure 1. The sparse coefficient matrix in the proposed QRSVM consumes $\frac{p^2+1}{p^2}$ fraction of the memory assigned to the original dense coefficient matrix.

![Fig. 1: QRSVM technique transforms a 6 × 6 dense and non-separable coefficient matrix into a sparse block diagonal matrix, where, the first 2 × 2 block is full rank and the second 4 × 4 block is a diagonal submatrix. Dense regions are colored. The two blocks in the transformed matrix on the Right are outlined in blue. Here, $n = 6$ and $p = 2$.](image)
Separability: We have also rendered the aforementioned non-separable quadratic coefficient matrix into a block separable form by using the proposed QRSVM formulation. One can exploit this separability to independently solve the two sub-problems in parallel at each iteration of the dual ascent algorithm.

Invertibility: On applying QRSVM, the low-rank quadratic coefficient matrix becomes block-separable where the two sub-blocks are now invertible. The first block \((RR^T)_p + (1/2C)I_p\) is full-rank in \(p\) and the second block \((1/2C)I_{n-p}\) is trivially invertible. The invertibility of the quadratic coefficient matrix makes the dual-ascent algorithm stable by allowing the computation of the minimization step in equation (6). Many methods exist to compute QR decomposition of a rectangular matrix, such as the Gram-Schmidt process, Householder transformations, or Givens rotations. Householder transformation [24] technique uses elementary orthogonal Householder matrices to compute QR decomposition. We choose the Householder transformation because it has a far better numerical stability than the Gram-Schmidt process [25]. In comparison to Given rotations, the chosen technique needs fewer arithmetic operations to obtain the same result [24]. Also, our matrix \(\hat{A}\) is a dense matrix for which benefits of Givens rotation can not be leveraged as for sparse matrices. While using Householder transformation, it is to be noted that the orthogonal \(Q\) matrix of size \(n \times n\) is never explicitly computed. Rather, one can simply store the \(p\) Householder reflector vectors. These reflectors are sufficient to generate the \(Q\) matrix, if required, or to apply \(Q\) to a vector. This strategy of storing only the reflectors reduces memory requirement to \(np\) from \(n^2\), which is required for storing the complete \(Q\).

The proposed QRSVM formulation can be implemented without any additional memory compared to the original problem in equation (3). The given input data \(\hat{A}\) can be replaced by the set of \(p\) Householder reflectors (i.e. \(Q\)) and the matrix \(R\), which together occupy the same \(np\) memory space as \(\hat{A}\). In addition, it is observed that one can recover the values of \(\alpha\) (to identify corresponding support vectors) from \(\hat{\alpha}\) by simply pre-multiplying \(\hat{\alpha}\) with series of \(p\) Householder reflectors with a computational cost of \(O(np)\). This cost is a lot cheaper compared to the \(O(n^2)\) computational cost incurred if one directly pre-multiplied the matrix \(Q\) instead. Finally, for the linear SVC problem with low original dimensionality \(d\), one can efficiently compute the normal \(w\) by directly pre-multiplying \(R^T\) to \(\hat{\alpha}\). It is worth noticing that calculation of \(R^T\hat{\alpha}\) can be simplified to \(R^T\hat{\alpha}_p\) i.e. \(O(d^2)\) given the special structure of matrix \(R\). For non-linear SVCS where \(p\)-rank kernel approximation techniques like in [26] are used, one can efficiently find the prediction for a test sample using the simplification \(R^T\hat{\alpha}_p\). The QRSVM process workflow is illustrated in Figure 2.

IV. OPTIMIZATION

Dual Ascent is a gradient method which involves iterating through the update steps until convergence [5]. The Lagrangian \(L\) of QP problem in equation (4) is written as follows

\[
L(\hat{\alpha}, \beta) = \frac{1}{2} \hat{\alpha}^T \left( RR^T + \frac{1}{2C} I_n \right) \hat{\alpha} + (\hat{e})^T \hat{\alpha} + \beta^T (-Q \hat{\alpha})
\] (5)

where, \(\beta \geq 0_n\) is the Lagrangian dual variable.

Dual ascent update steps for QRSVM are as follows.

Step 1: Minimization of Lagrangian

\[
\hat{\alpha}^{k+1} = \arg \min_{\alpha} L(\hat{\alpha}, \beta^k) = \left( RR^T + \frac{1}{2C} I_n \right)^{-1} (-Q^T \beta^k + \hat{e})
\] (6)

Step 2: Dual variable update

\[
\beta^{k+1} = \beta^k + \eta (-Q \hat{\alpha}^{k+1})
\] (7)

where \(\eta > 0\) is the step size and the superscript \(k\) is the iteration counter. \(\beta^0\) is initialized to \(0_n\). To satisfy the inequality constraint on each of the dual variable \(\beta_i\) being non-negative, \(\beta_i\) is replaced with \(\max(0, \beta_i)\) during every iteration.
A. Calculating optimal step size

In this section, we provide a formulation for computing the optimal step size, \( \eta^* \) that will ensure the least number of iterations for the dual update. For this, we extend a theorem from Lee et al. (27, Theorem 1) which states a result for the optimal synchronization period in a given parallel QP problem solved using their proposed lazy synchronous dual ascent (LSDA) technique. In their parallel QP problem, every worker node in a cluster of processors computes its local minimization step, analogous to equation (6) above. For the dual variable update step, these local \( w_i \)'s are aggregated by lazily synchronizing the cluster nodes, once in every optimal synchronization period, \( P^* \). We observe that the above technique can also be interpreted as tightly synchronizing among the various nodes with an optimal step size, where, \( P^* \) can be considered as the optimal scaling factor. We provide the following lemma (extending Theorem 1 in [27]) for the dual ascent method to ensure a minimum number of iterations for the solution of QRSVM to converge.

Lemma 1. (Scaling factor for optimal step size) To ensure the minimum number of iterations involving the dual variable update step, the scaling factor \( P^* \) for optimal step size is obtained by

\[
P^* = \max_{P \in \mathbb{N}} \min \{ |1 - \lambda_{\min}(M)P|, |1 - \lambda_{\max}(M)P| \}
\]

where, \( M := \eta \left( RR^T + \frac{1}{\eta^2} I_n \right)^{-1}, \eta > 0 \) is step size and \( \lambda_{\min}(\cdot) \) and \( \lambda_{\max}(\cdot) \) denote the smallest and the largest eigenvalues of the square matrix \( M \), respectively.

On solving equation (8), we get the following result.

Corollary 1. For any \( \eta > 0, \) the optimal step size \( \eta^* \) can be computed using

\[
\eta^* = P^* \eta, \quad P^* \in \mathbb{N}
\]

where,

\[
P^* = \begin{cases} 
1 & \text{if } 0 < \lambda^{-1} < 2 \\
\lfloor \lambda^{-1} \rfloor & \text{if } \lambda^{-1} \geq 2
\end{cases}
\]

and \( \lambda = (\lambda_{\max}(M) + \lambda_{\min}(M))/2 \)

Proof. On plotting the functions \( f_1(P) = |1 - \lambda_{\min}(M)P| \) and \( f_2(P) = |1 - \lambda_{\max}(M)P| \), we can observe that the intersection of the above two functions occurs at point \( L(\lambda^{-1}, \lambda_{\max}\lambda^{-1} - 1) \) as illustrated in Figure 3.

\[
\max\{f_1(P), f_2(P)\} = \begin{cases} 
1 - \lambda_{\min}(M)P & \text{if } 0 < P \leq L_x \\
\lambda_{\max}(M)P - 1 & \text{if } P > L_x
\end{cases}
\]

where, \( L_x = \lambda^{-1} \)

Minimum value of \( \max\{f_1(P), f_2(P)\} \) occurs at point of intersection \( L \). In other words,

\[
\arg\min \max\{f_1(P), f_2(P)\} = L_x.
\]

Since, \( P^* \in \mathbb{N} \) as per Lemma 1, we must ensure that for \( 0 < L_x < 2 \), optimal scaling factor \( P^* = 1 \). When \( L_x \geq 2 \), \( P^* \) is assigned the highest integral value lesser than \( L_x \), i.e., \( P^* = \lfloor \lambda^{-1} \rfloor \).

This value which is lesser than \( L_x \) ensures that the scaling factor is optimum, \( P^* \), and will result in optimal step size, \( \eta^* \), leading to stability and convergence of dual ascent method. \( \square \)

It is worth noting that given the special structure of \( M \) comprising of the inverse of a block separable sparse matrix (positive semi-definite), \( \lambda_{\min}(M) = 2\eta C/(1 + 2C\lambda_{\max}(RR^T)) \) and \( \lambda_{\max}(M) = 2\eta C \). For practical values of \( C \) in the proposed formulation, \( \lambda_{\max}(M) \gg \lambda_{\min}(M) \). Hence, \( \lambda^{-1} \approx 1/(\eta C) \) can be used as a good approximation for faster convergence of the dual ascent method.

B. Complexity Analysis

The cost of a single iteration of QRSVM is the combined computation cost of the two update steps defined in equations (6) and (7). Premultiplying \( Q \) (or \( Q^T \)) to a vector \( v \) by using Householder reflectors requires \( O(np) \) operations, where \( n \) is the size of vector \( v \) and \( p \) is the number of Householder reflectors [28]. The cost of computing each of \(( -Q^T \beta^k ) \) in equation (6) and \(( -Q\hat{\alpha}^{k+1} ) \) in equation (7) is \( O(np) \). Given the block diagonal structure of \( RR^T + 1/\eta^2 I_n \), the computation in equation (6) can be split into following:

Subproblem 1: The first \( p \) components of \( \hat{\alpha}^{k+1} \) are computed by solving a system with the \( p \times p \) coefficient matrix \(( RR^T + 1/\eta^2 I_p ) \). By computing and storing Cholesky factors of this matrix before starting the iterations, the system can be solved in \( O(p^3) \) operations. Cholesky factorization of the coefficient matrix is one-time calculation that is carried out in the beginning of the dual ascent algorithm at cost of \( O(p^3) \).
Subproblem 2: Calculation of the remaining \((n-p)\) components of \(\hat{\alpha}^{k+1}\) requires \(O(n-p) \approx O(n)\) operations since low-rank \(p \ll n\).

The overall computation cost of update equation (6) is \(O(np + p^2)\). Also, the computation cost of equation (7) is trivially \(O(np)\). Combining the above two computation cost for equations (6) and (7) and including the initial QR decomposition cost, we see that QRSVM using dual ascent method requires \(O(np^2 + knp)\) operations where \(k\) is the number of iterations. The trend is empirically illustrated in Figure 6.

V. DISTRIBUTED QRSVM

With the increase in the quantity of data and challenges associated with it in terms of computation and storage, it has become evident to look for distributed algorithms that can solve parallel SVM problems efficiently. QRSVM scales linearly with the training dataset size \(n\) which makes it scalable to large datasets. In this section, we lay down the formulation of parallel non-linear SVMs and a distributed version of our proposed QRSVM framework to solve the classification problems.

A. Distributed QR decomposition

Using kernel approximation techniques, we represent \(K \approx A A^T\) and \(G \approx A A^T\). In the QRSVM framework, \(\hat{A}\) is decomposed into factors \(Q\) and \(R\). To deal with large data sizes, we partition the approximated tall data matrix \(\hat{A} = [\hat{A}_1; \ldots; \hat{A}_S]\), instance-wise (horizontally) and distribute it equally across \(S\) computing nodes. We chose the cluster size \(S\) such that each partitioned data \(\hat{A}_i \in \mathbb{R}^{\frac{n}{S} \times p}\) maintains the tall and skinny structure as in the original formulation i.e. \(p \ll \frac{n}{S} \implies S \ll \frac{n}{p}\).

We now discuss how to distribute the decomposition of \(\hat{A} = QR\) across cluster of computing nodes such that the resulting matrices \(Q\) and \(R\); representatives of the overall data, can be generated from the partitioned matrices \(\hat{A}_i\).

Theorem 1. For QR decomposition of a matrix \(\hat{A} = QR\), where \(Q\) is an orthogonal matrix and \(R\) is an upper triangular matrix, we can generate \(Q\) and \(R\) from \(S\) horizontal partitions of \(\hat{A} = \{\hat{A}_i\}, i = 1..S\) as follows

\[
Q = \text{diag}(Q_1, Q_2, \ldots, Q_S) \times Q_g \\
R = R_g
\]

where, \(\hat{A}_i = Q_i R_i\) and \([R_1; \ldots; R_S] = Q_g R_g\)

Here, \(Q_g\) and \(Q_s\) are orthogonal matrices and \(R_i\) and \(R_g\) are upper triangular matrices.

Proof. For \(\forall i \in 1..S\) computing nodes, \(\hat{A}_i = Q_i R_i\). We can write the overall \(\hat{A}\) as

\[
\hat{A} = \begin{bmatrix} \hat{A}_1 \\ \vdots \\ \hat{A}_S \end{bmatrix} = \begin{bmatrix} Q_1 & \cdots & Q_S \end{bmatrix} \begin{bmatrix} R_1 \\ \vdots \\ R_S \end{bmatrix}
\]

Since each \(Q_i\) is an orthogonal matrix,

\[\text{diag}(Q_1, \ldots, Q_S)\] is an orthogonal matrix. Since each \(R_i \in \mathbb{R}^{\frac{n}{S} \times p}\) is an upper triangular matrix, \(R_{\text{gather}} = \begin{bmatrix} R_1 \\ \vdots \\ R_S \end{bmatrix}\) is not an upper triangular matrix.

In the master node, we can further decompose

\[R_{\text{gather}} = Q_g R_g\]

such that \(Q_g \in \mathbb{R}^{n \times n}\) is an orthogonal matrix and \(R_g \in \mathbb{R}^{n \times p}\) is an upper triangular matrix.

Now,

\[\hat{A} = \text{diag}(Q_1, \ldots, Q_S) \times Q_g R_g\]

We know that \(\hat{A} \in \mathbb{R}^{n \times p}\) is \(p\)-rank. In other words, \(\hat{A}\) has linearly independent columns which imply its QR decomposition is unique.

Hence, \(R = R_g\) and \(Q = \text{diag}(Q_1, \ldots, Q_S) \times Q_g\)

The implementation of the distributed QR decomposition technique is further illustrated in Figure 4.

Corollary 2. For any vector \(v \in \mathbb{R}^n\), \(\hat{v} = Q^T v\) stored as partitions in the cluster nodes and orthogonal matrices \(Q\) and \(Q_i\), computing \(Q \hat{v}\) and \(Q^T v\) in the distributed QRSVM framework can be formulated as

\[Q \hat{v} = \text{diag}(Q_1, \ldots, Q_S) \times \{Q_g \hat{v}\}\]

and

\[Q^T v = Q^T_g \times \{\text{diag}(Q_1^T, \ldots, Q_S^T) v\}\]
Each of the above computation inherently requires communication (gather and scatter) across the distributed network.

For \( Q \hat{v} \) computation, we first gather all local \( \hat{v}_i \) from the worker nodes and calculate \( \{ Q_g \hat{v}_i \} \) at the master node. Then, we scatter its partition \( \{ Q_g \hat{v}_i \} \) across all the computing nodes in a cluster. Each of which has its local \( Q_i \) from QR decomposition of its partitioned data \( A_i \). The scattered \( \{ Q_g \hat{v}_i \} \) is then pre-multiplied with local \( Q_i \) to compute \( v_i \).

For \( Q^T \hat{v} \) computation, we first calculate local \( \{ Q_i^T v_i \} \) in each worker node \( i \). Then, we gather each of the local values at the master node and pre-multiply it with \( Q_g^T \) to generate \( \hat{v} \). Finally, we scatter it to all the cluster nodes as \( \hat{v}_i \).

Again, it is worth noting that these \( Q_i \)'s and \( Q_g \) are never explicitly calculated, rather, stored as respective sets of \( p \)-Householder reflectors in the computing nodes.

B. Parallel Dual Ascent

With the distributed QR decomposition technique, it is possible to update the dual ascent steps, equations (6) and (7), in parallel across the cluster nodes. From equation (6), let us define the invertible coefficient matrix (substituting, \( R = R_g \))

\[
F = -\left( R_g R_g^T + \frac{1}{2C} \times I_n \right)
\]

which has a sparse and separable structure, well illustrated in Figure 1. As a result of its separability, we can block-partition \( F = F_1 \oplus F_2 \oplus \ldots \oplus F_S \) such that each diagonal block \( F_i \in \mathbb{R}^{\frac{n}{S} \times \frac{n}{S}} \) is allocated to each computing cluster node \( i \). Here, \( \oplus \) represents an operator that diagonally combines the sub-blocks to generate the entire block-diagonal matrix.

Since \( R_g \) is stored at the master node and \( p \ll \frac{n}{S} \) as to maintain the tall and skinny structure of \( A_i \), the dense and symmetric \( p \times p \) block i.e. \( \left( R_g R_g^T \right)_p + (1/2C)I_p \) in coefficient matrix \( F \) becomes a part of the partition \( F_i \) at the master node. It is also worth appreciating that with the distributed QRSVM formulation, there is no need to actually partition and store the other diagonal blocks \( F_i \) in the worker nodes \( i = 2..S \) as these are simply constant diagonal matrices \((-0.5/C)I_{n/S}\).

On parallelizing Step 1 of the dual ascent for iteration \((k+1)\), at compute node, \( i \)

\[
\alpha_{i}^{k+1} = F_i^{-1}(-\hat{\beta}^k_i + \epsilon_i) \quad (10)
\]

where,

\[
F_i^{-1} = \begin{cases} 
F_1^{-1} & \text{if } i = 1 \\
-2C & \text{if } i = 2..S
\end{cases}
\]

and, \( \hat{\beta}^k = Q^T \beta^k \)

Similarly, on parallelizing Step 2 of the dual ascent for iteration \((k+1)\), at compute node, \( i \)

\[
\hat{\beta}_i^{k+1} = \hat{\beta}_i^k + \eta^* (-\hat{\alpha}_i^{k+1}) \quad (11)
\]

Here, we are using the optimal step size \( \eta^* \) defined in equation (9). We also note that by changing the dual variable from \( \beta \) to \( \hat{\beta} \) in the above update steps, we ensure that \( \alpha_{i}^{k+1} \) and \( \hat{\beta}_i^{k+1} \) calculation during each iteration \((k+1)\) occurs locally without requiring any communication/synchronization across the computing cluster nodes. However, after every iteration, the original dual variable \( \beta \) has to be checked for non-negativity as discussed earlier in Section IV. This requires communication (gather and scatter) across the cluster nodes as we transform from \( \hat{\beta} \) to \( \beta \) using \( Q \hat{\beta} \) and then transform back to \( \hat{\beta} \) (to be used in Step 1 in the next iteration) using \( Q^T \beta \) after checking (and zeroing) the negative \( \beta \) values.

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

The distributed QRSVM framework is implemented in C/C++. The linear algebra computations in the framework were handled using Armadillo library [29] integrated with LAPACK/BLAS. The programs were run in Ada super-computing cluster of Texas A&M HPRC with InfiniBand interconnect. Each of the 792 general compute nodes in this super-computing cluster is a 10-core Intel Xeon E5-2670 v2 (Ivy Bridge) processor with 64GB memory. We use Message-Passing Interface (MPI) [30] as inter-node communication platform.

We use binary classification datasets freely available on LIBSVM datasets repository

1.  
   - a9a: The machine learning task here is to predict whether income exceeds $50K/yr based on census data. Data sparsity is 11.3%. \( n_{\text{train}} = 32560 \), \( d = 123 \)
   - covtype.binary: One needs to predict the forest cover type (class 2 vs other 5 classes) using cartographic variables only. We use the [0,1] scaled version of the dataset having sparsity of 22%. \( n_{\text{train}} = 464810 \), \( d = 54 \)

B. Convergence and Scalability of QRSVM

Convergence: We show the convergence of the basic QRSVM framework on single-machine and compare it with state-of-the-art linear solver library, LIBLINEAR (double coordinate descent). We choose HIGGS\(^1\) dataset (\(n=10.5M\) and \(d=28\)) in linear SVM formulation here to simply demonstrate the superior convergence of our approach compared to the linear SVM solver. As shown in Figure 5a, it was observed that for most of the problems, QRSVM reaches a reasonable value of the objective function within 20 iterations of dual ascent method. Figure 5b compares the convergence rates of QRSVM with LIBLINEAR. It must be pointed out that the LIBLINEAR algorithm implementation in [12] is set to

\(^1\)https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
Fig. 5: Convergence of QRSVM for HIGGS dataset: (a) Using QRSVM we converge to a reasonable value of the optimal cost within 20 iterations. (b) QRSVM converges relatively faster compared to LIBLINEAR. Here, we illustrate for 250 iterations. LIBLINEAR was not able to converge to the optimal cost value in 1500 iterations while QRSVM converged to the optimum in 80 iterations.

terminate at a preset maximum iteration count (default 1000). In our experimentation, we chose this preset value to 1500. The result at the end of this maximum iteration was used to compare with QRSVM. QRSVM was able to converge to the optimal cost of the objective function for HIGGS dataset within 80 iterations while LIBLINEAR couldn’t converge even after 1500 iterations.

**Scalability with n:** The effect on training time of QRSVM was analyzed by increasing number of instances, \( n \) in a synthetic dataset. Its kernel approximation, \( p=18 \) was kept fixed. It is observed that QRSVM training time increases linearly with the number of data points as shown in Figure 6a. As a result, our proposed framework is capable of handling growing data sizes efficiently making it suitable for big data applications.

**Scalability with p:** With increasing the rank of the approximated kernel, the training time of QRSVM is observed to grow quadratically as shown in Figure 6b. This trend validates our discussion in Section IV-B. However, as we are addressing the low-rank approximation of the input problem space through the proposed QRSVM framework, the benefits from the fast overall convergence rate of the algorithm outweighs the quadratic cost.

**C. Distributed QRSVM: Timing Discussions**

**Computation time, \( t(p) \):** The proposed distributed QRSVM framework for parallel SVM training has the following major computational requirements: Low-rank representation of the kernel matrix using MEKA [20] which has time complexity proportional to the memory for storing the approximated kernel matrix. We denote this time as \( t(p_{mek}) \). The distributed QR decomposition stage shown in Figure 4, partitioned data \( A_i \) is locally decomposed into \( Q_i \) and \( R_i \). We denote its worst case computation time as \( t(p_{localQR}) \) for any given cluster node. At the master node in the above stage, the gathered data \( R_{gather} \) is further \( R_f \) decomposed whose computation time we denote as \( t(p_{masterQR}) \). Finally, in the parallel dual-ascent (pda) stage, we solve the update steps in equations (10) and (11) locally in the cluster nodes. The computation time for the update steps is denoted as, \( t(p_{pda}) \). All the above computation timings for the chosen benchmarks are listed in Table I.

**Communication time, \( t(c) \):** The distributed QRSVM framework solves the kernel SVM problem in the process of which it needs to communicate data across the distributed network. It has two necessary communication requirements. The first need for communication across the distributed cluster occurs when local \( R_f \) are gathered at the master node during the distributed QR decomposition stage. This is also depicted in Figure 4. Let us denote this communication time as \( t(c_{gatherR}) \). The second requirement to communicate happens in the parallel dual-ascent (pda) stage. As discussed in Section V-B, bulk of the communication happens after every iteration in the form of \( gather \) and \( scatter \). This is necessary to ensure convergence of the algorithm. Let us denote this communication time as \( t(c_{pda}) \). The overall SVM training time, \( t(train) = t(p) + t(c) \) for both the datasets is also shown in Table I.

**Optimal step size:** For different values of \( C \) and step size \( \eta \), we derive the optimal step size using equation 9 for training the models for the benchmark datasets, a9a and covtype. Figure 8 plots the training time for the various step sizes and we chose the \( \eta^* = 1.9 \) as optimal that provides the fastest time. Table I lists the above parameters, \( C \) and \( \eta^* \). Figure 7 shows the convergence of distributed QRSVM as function of training error (log) over number of iterations.
Fig. 6: Scalability of QRSVM (a) with \( n \): A synthetic dataset having a fixed approximated kernel rank-\( p = 18 \) and increasing \( n \) was used to test scalability with number of instances. (b) with \( p \): A synthetic dataset with fixed number of instances, \( n = 100,000 \) and increasing \( p \) was used to test scalability with rank-\( p \) (dimensionality).

D. Comparison with PSVM

We compare the proposed distributed-QRSVM framework with state of the art kernel SVM solver, PSVM [15]. PSVM uses incomplete Cholesky decomposition to approximate the kernel matrix whereas we use the memory-efficient MEKA [20]. Both PSVM and the proposed framework optimize the dual objective/cost function making our comparison fair. As for PSVM parameters, we used the default setting. The dual residual threshold for convergence was set to 0.001 which we also keep as the stopping threshold for QRSVM. The upper limit of 10000 iterations was maintained for PSVM and the suggested \( m^* = m^{0.5} \) was used. As mentioned in [15], the value of \( m \) approximation was chosen to ensure a balance between accuracy and efficiency.

We observe that the distributed version of the proposed QRSVM framework trains the SVM model for the covtype.binary dataset in around 2 minutes using \( S = 16 \) processors, whereas, PSVM under the same setup converges in around 20 minutes. Another distributed kernel solver, P-packSVM [16] solves the same dataset 1.24x faster (for P-pack, \( r = 100 \) model) than PSVM using 16 processors as reported in Table II in [16]. Due to unavailability of open source code for P-packSVM, we were unable to accurately report its training time for the above dataset on our experimental infrastructure. However, we can estimate that for 16 processors in our infrastructure, P-packSVM would have clocked around 16 minutes (1.24x better than PSVM). Our framework, in comparison to P-packSVM, has converged in 2 minutes.

VII. CONCLUSION

We proposed a QR decomposition framework for solving kernel support vector classification problems. The framework uses Householder transformation to convert a dense and low rank matrix to a highly sparse and separable matrix, with invertible block-partitions. In addition, we provide optimal step size for solving dual ascent method for faster convergence. We empirically demonstrate that the proposed QRSVM framework scales linearly with dataset size making it suitable to handle large data problems. We further present a distributed QRSVM

---

<table>
<thead>
<tr>
<th>TABLE I: Distributed-QRSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time details</td>
</tr>
<tr>
<td>( t(p_{meka}) )</td>
</tr>
<tr>
<td>( t(p_{localQR}) )</td>
</tr>
<tr>
<td>( t(p_{masterQR}) )</td>
</tr>
<tr>
<td>( t(p_{gather}) )</td>
</tr>
<tr>
<td>( t(p_{pda}) )</td>
</tr>
<tr>
<td>( t(c_{pda}) )</td>
</tr>
<tr>
<td>( t(train) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>a9a</th>
<th>covtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank, ( p )</td>
<td>40</td>
<td>64</td>
</tr>
<tr>
<td>( C )</td>
<td>( 2^{-10} )</td>
<td>( 2^{-10} )</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>( 2^{-3} )</td>
<td>( 2^{-1} )</td>
</tr>
<tr>
<td>approx. ( K_{error} )</td>
<td>0.51</td>
<td>0.58</td>
</tr>
<tr>
<td>#processors, ( S )</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>stopping threshold</td>
<td>( 10^{-3} )</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>optimal step size, ( \eta^* )</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>#iterations, ( k )</td>
<td>166</td>
<td>512</td>
</tr>
</tbody>
</table>

---

(a) Scales linearly with number of instances

(b) Scales quadratically with rank-\( p \) approximation.

Fig. 6: Scalability of QRSVM (a) with \( n \): A synthetic dataset having a fixed approximated kernel rank-\( p = 18 \) and increasing \( n \) was used to test scalability with number of instances. (b) with \( p \): A synthetic dataset with fixed number of instances, \( n = 100,000 \) and increasing \( p \) was used to test scalability with rank-\( p \) (dimensionality).

Fig. 7: Training error trend during model learning phase for the datasets. a9a training takes 166 iterations to converge and covtype takes 512 iterations. Stopping error threshold: \( 10^{-3} \) and #processors=16

---

VII. CONCLUSION

We proposed a QR decomposition framework for solving kernel support vector classification problems. The framework uses Householder transformation to convert a dense and low rank matrix to a highly sparse and separable matrix, with invertible block-partitions. In addition, we provide optimal step size for solving dual ascent method for faster convergence. We empirically demonstrate that the proposed QRSVM framework scales linearly with dataset size making it suitable to handle large data problems. We further present a distributed QRSVM
framework to accelerate the training of kernel SVM model by parallelizing the dual-ascent algorithm. As an extension of this work, it will be interesting to work on reducing the computation at parallel dual ascent stage involving model-update iteration to further accelerate the SVM training. Possible solutions to this can be either exploring asynchronous computation at parallel dual ascent stage involving model-update iteration to further accelerate the SVM training. Possible solutions to this can be either exploring asynchronous communication protocols or designing an efficient hardware accelerator for applications in Internet of Things (IoT) and smart embedded systems.

ACKNOWLEDGMENT

The authors would like to thank Texas A&M High Performance Research Computing (http://hprc.tamu.edu/) for providing the computing resources to run the benchmarks.

REFERENCES


Fig. 8: Optimal step size: For both datasets, optimal step size is observed to be 1.9


