Distributed QR decomposition framework for training Support Vector Machines

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

{dass.jyotikrishna, prithvi.sakuru, sarin, rabi }@tamu.edu

37th IEEE ICDCS 2017, Atlanta
Introduction
Some applications of Quadratic Programming (QP) are

1. Least Square approximations
2. Regression Analysis
3. Portfolio Optimization
4. Support Vector Machines
5. Optimal Control
Support Vector Machines (SVM)

We focus on distributed data analytics using SVM

- Supervised machine learning model (data+label)
- Widely used for data classification for its high efficiency
- Popular for multivariate non-linear datasets (kernel SVM)
- Have been extended for tasks like regression analysis (SVR), principal component analysis etc.
SVM as a QP problem

- SVM is a convex optimization problem (QP)
- Solves for maximal separating hyperplane as a classifier

- Maps training vectors into a high dimensional space via a nonlinear function (kernel SVM)
- Hence, solving for dual (rather than primal) form is preferred using ”kernel trick”

Specifically, we focus on the

**two-class soft margin SVM with $l_2$-regularization and $l_2$-loss**
SVM formulation

training dataset, $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, n\}$

input data matrix, $X = \{x_i \in \mathbb{R}^d, i = 1 \ldots n\}$, $d$-dimensional space

class label vector, $y = \{y_i \in \{-1, 1\}, i = 1 \ldots n\}$

dual SVM

$$
\min_\alpha \frac{1}{2} \alpha^T \left( \text{diag}(y) \times K \times \text{diag}(y)^T \right) \alpha + \frac{1}{2} \alpha^T \left( \frac{1}{2C} I_n \right) \alpha + e^T \alpha \tag{1}
$$

subject to $-I_n \alpha \leq 0_n$

where, $\alpha$ is a vector of dual variables

$e = -1_n$

$C > 0$ is penalty parameter for misclassification

$K = \{k(x_i, x_j), \forall i, j = 1 \ldots n\}$ is positive definite matrix (mostly)

$k()$ represents the Mercer kernel function - linear/non-linear
Kernel function:
\[ k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \]
where, \( \phi() \) is a mapping generally not known or inefficient to compute.
Kernel SVM for non-linear data

**Kernel function:**

\[ k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \]

where, \(\phi()\) is a mapping generally not known or inefficient to compute.

However,

\( k(x_i, x_j) \) is known and easier to compute (”Kernel trick”).

- Linear kernel: \( k(x_i, x_j) = \langle x_i, x_j \rangle \)
- Radial Basis Function (RBF) kernel:
  \[ k(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2) \], where, \( \gamma \) is hyperparameter

Measures ”similarity” between two data points in the Feature space.
For large sample size $n$, Kernel methods become unfeasible because

1. $K$ requires $O(n^2)$ memory and
2. it incurs computational cost of $O(n^3)$ to solve such problems
For large sample size $n$, Kernel methods become unfeasible because

1. $K$ requires $O(n^2)$ memory and
2. it incurs computational cost of $O(n^3)$ to solve such problems

Go for **Low Rank Kernel Approximation**!

**Low $p$—rank approximation of $K$**

$$K \approx AA^T, \text{ where, } A \in \mathbb{R}^{n \times p} \text{ and } p \ll n.$$ 

We use MEKA [Si, 2014] for memory efficient and lower approximation error compared to Nyström methods etc.
Recall,

dual SVM

$$\min_{\alpha} \frac{1}{2} \alpha^T \left( \text{diag}(y) \times K \times \text{diag}(y)^T \right) \alpha + \frac{1}{2} \alpha^T \left( \frac{1}{2} C I_n \right) \alpha + e^T \alpha$$

subject to \quad -I_n \alpha \leq 0_n
Recall,

\[ \text{dual SVM} \]

\[
\min_{\alpha} \frac{1}{2} \alpha^T \left( \text{diag}(y) \times K \times \text{diag}(y)^T \right) \alpha + \frac{1}{2} \alpha^T \left( \frac{1}{2C} I_n \right) \alpha + e^T \alpha \\
\text{subject to } -I_n \alpha \leq 0_n
\]

Substitute, \( K \approx AA^T \) and define, \( \hat{A} = \text{diag}(y) \times A \)

\[ \text{approximated dual SVM} \]

\[
\min_{\alpha} \frac{1}{2} \alpha^T \left( \hat{A} \hat{A}^T \right) \alpha + \frac{1}{2} \alpha^T \left( \frac{1}{2C} I_n \right) \alpha + e^T \alpha \\
\text{subject to } -I_n \alpha \leq 0_n
\]
Motivation
Goal

To devise a fast and memory-efficient distributed framework to train large-scale SVM
Goal

To devise a fast and memory-efficient distributed framework to train large-scale SVM

Our Contribution

1. QRSVM: QR decomposition framework for memory-efficient modeling and training of SVM
2. Optimal step size calculation for fast convergence of Dual Ascent method which iteratively solves the SVM problem
3. Distributed QRSVM: designing distributed QR decomposition and parallel Dual Ascent techniques for distributed SVM training
4. Compared training time of distributed QRSVM with competing distributed methods; PSVM and P-packSVM
QRSVM

Memory-efficient modeling and training of SVM
\( \hat{A} \in \mathbb{R}^{n \times p} \) with \( p \ll n \) has a tall and skinny (TS) structure

**QR decomposition**

\[
\hat{A} = QR,
\]

where, \( Q \in \mathbb{R}^{n \times n} \) is Orthogonal matrix

\( R \in \mathbb{R}^{n \times p} \) is Upper Triangular matrix

**Figure: \( \hat{A} \)**

**Figure: \( R \)**

\[Q, \ O(n^2) \rightarrow p-\]

Householder reflector vectors, \( O(np) \)
Recall, approximated dual SVM

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

subject to \(- I_n \alpha \leq 0_n\)

Now, Substitute \(\hat{A} = QR\)
Formulation

Substituting $\hat{A} = QR$

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

subject to $- I_n \alpha \leq 0_n$

Define, $\hat{\alpha} = Q^T \alpha$, $\hat{e} = Q^T e$ and using $Q^T Q = I_n$
Formulation

QRSVM

\[
\min_{\hat{\alpha}} \frac{1}{2} \hat{\alpha}^T \left( RR^T + \frac{1}{2C} I_n \right) \hat{\alpha} + (\hat{e})^T \hat{\alpha} \\
\text{subject to} \quad -Q\hat{\alpha} \leq 0_n
\]
Memory-efficient modeling

Structure of Hessian matrix

\[
\left( \hat{A} \hat{A}^T + \frac{1}{2C} I_n \right) \Rightarrow \left( RR^T + \frac{1}{2C} I_n \right)
\]

Dense \( O(n^2) \)
Non-separable

Sparse \( O(p^2) \)
block diagonal separable
Dual Ascent to solve linearly constrained Optimization problem

Lagrangian $\mathcal{L}$ of QRSVM

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

\textit{where}, $\beta \geq 0_n$ is the Lagrangian dual variable.

Dual Ascent

Dual function: 
\[ g(\beta) = \min_{\hat{\alpha}} \mathcal{L}(\hat{\alpha}, \beta) \]

Dual Problem: 
\[ \max_{\beta} g(\beta) \]
Dual Ascent steps

Gradient method - involves iterating through the following steps until convergence (error in $\beta$ falls below stopping threshold)

**Step 1: Minimization of Lagrangian**

$$\hat{\alpha}^{k+1} = \arg \min_{\hat{\alpha}} \mathcal{L}(\hat{\alpha}, \beta^k)$$

$$= - \left( RR^T + \frac{1}{2C} \times I_n \right)^{-1} (-Q^T \beta^k + \hat{e}) \quad (5)$$

**Step 2: Dual variable update**

$$\beta^{k+1} = \beta^k + \eta(-Q\hat{\alpha}^{k+1}) \quad (6)$$

$\eta > 0$ is the step size, $\beta^0 = 0_n$. 


### QRSVM Workflow

Two stages of QRSVM

1. **QR decomposition**: Computational cost $O(np^2)$
2. **Dual Ascent method**: Computational cost $O(np)/iteration$
Optimal Step Size

Fast convergence of Dual Ascent
Based on optimal synchronization period defined for Lazily Synchronous Dual Ascent method, Theorem 1 [Lee, 2016]

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

$$P^* = \max_{P \in \mathbb{N}} \arg \min \left\{ |1 - \lambda_{\min}(M)P|, |1 - \lambda_{\max}(M)P| \right\}$$  \hspace{1cm} (7)$$

$$M := \eta \left( RR^T + \frac{1}{2c} I_n \right)^{-1},$$

$\eta > 0$ is step size

$\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ eigenvalues of matrix $M$
For any $\eta > 0$, the optimal step size $\eta^*$ can be computed using

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

(8)

where,

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

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

$$\bar{\lambda}^{-1} \approx 1/(\eta C)$$
Distributed QRSVM
Stage 1: Distributed QR decomposition

Partition data, $\hat{A}_i \in \mathbb{R}^{n \times p}$ on $S$ worker nodes

\[ p \ll \frac{n}{S} \implies S \ll \frac{n}{p} \]

**Theorem**

Given, $S$ horizontal partitions of $\hat{A} = \{\hat{A}_i\}, i = 1..S$,

1. $\hat{A}_i = Q_i R_i$ at each worker node $i$
2. **Gather** all $R_i$’s at the Master node
3. $[R_1; ..; R_S] = Q_g R_g$ at Master node

One can represent the factors $Q$ and $R$ of the complete $\hat{A}$ in distributed formulation as

\[
Q = \text{diag}(Q_1, Q_2, .. Q_i .., Q_S) \times Q_g
\]

\[
R = R_g
\]
Stage 1: Distributed QR decomposition

$\hat{A}_1 = \{q_1\} R_1$

$\hat{A}_2 = \{q_2\} R_2$

$\hat{A}_s = \{q_s\} R_s$

$Q_i$ stored as sets of their Householder reflectors, denoted as $\{q_i\}$
Stage 2: Parallel Dual Ascent

Define, \( F = -\left( R_g R_g^T + \frac{1}{2C} I_n \right) \)

**Step 1: Minimization of Lagrangian**

Recall,

\[
\hat{\alpha}^{k+1} = \arg \min_{\hat{\alpha}} \mathcal{L}(\hat{\alpha}, \beta^k)
\]

\[= F^{-1}(-Q^T \beta^k + \hat{e})\]

Define, \( \hat{\beta}^k = Q^T \beta^k \)
Stage 2: Parallel Dual Ascent

Define, \( F = -\left( R_g R_g^T + \frac{1}{2C} I_n \right) \)

Partition \( F \) into \( S \) block-diagonals, \( F_i \in \mathbb{R}^{\frac{n}{S} \times \frac{n}{S}} \)

Figure: Block Separable into \( F_i \)
Stage 2: Parallel Dual Ascent

Define, \( F = - \left( R_g R_g^T + \frac{1}{2C} I_n \right) \)

**Step 1: Minimization of Lagrangian - In Parallel**

At compute node, \( i \)

\[
\hat{\alpha}_i^{k+1} = F_i^{-1} \left( -\hat{\beta}_i^k + \hat{e}_i \right)
\]

where,

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

Step 2: Dual variable update

Recall,

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

Using, $$\hat{\beta}^k = Q^T \beta^k$$
Stage 2: Parallel Dual Ascent

**Step 2: Dual variable update - In Parallel**

At compute node, $i$

$$\hat{\beta}_i^{k+1} = \hat{\beta}_i^k + \eta^*(-\hat{\alpha}_i^{k+1})$$  \hspace{1cm} (10)

$\eta^*$ is the Optimal step size

$\hat{\beta}^k = Q^T \beta^k$
Stage 2: Parallel Dual Ascent- Implementation

- Local update calculations
- $\hat{\beta}_i$ gather to $\hat{\beta}$
- $\hat{\beta} \Rightarrow \beta$
- $\beta$ scatter to $\beta_i$
- Ensure $\beta_i \geq 0$
- $\beta_i$ gather to $\beta$
- $\beta \Rightarrow \hat{\beta}$
- $\hat{\beta}_i$ scatter to $\hat{\beta}$
Experimental Results
Experimental Setup

**Hardware**

- *Ada* Supercomputing Cluster at TAMU
- Intel Xeon E5-2670 v2 (Ivy Bridge-EP), 10-core, 2.5GHz
- 64 GB/node and 16 cores/node
- Message-Passing Interface (MPI), InfiniBand interconnect

<table>
<thead>
<tr>
<th>Dataset</th>
<th>n</th>
<th>d</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9a</td>
<td>32560</td>
<td>123</td>
<td>predict annual income</td>
</tr>
<tr>
<td>covtype</td>
<td>464810</td>
<td>54</td>
<td>predict forest cover type</td>
</tr>
</tbody>
</table>
Convergence

Figure: a9a: k=166, covtype: k=512, threshold=10^{-3}
Scalability of QRSVM: $O(np^2)$

**Figure:** Scales linearly with $n$

**Figure:** Scales quadratically with rank $p$
Optimal Step Size, $\eta^*$

**Figure:** a9a, $\eta^* = 1.9$

**Figure:** covType, $\eta^* = 1.9$
## Distributed QRSVM: Timing Discussions

### Stage 1: Distributed QR

1. **Computation:** $t(p_{localQR}) + t(p_{masterQR})$
2. **Communication:** $t(c_{gatherR})$

### Stage 2: Parallel Dual Ascent

1. **Computation:** $t(p_{pda})$
2. **Communication:** $t(c_{pda})$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$t(p_{meka})$ (in ms)</th>
<th>$t(p_{localQR})$ (in ms)</th>
<th>$t(p_{masterQR})$ (in ms)</th>
<th>$t(c_{gatherR})$ (in ms)</th>
<th>$t(p_{pda})$ (in ms)</th>
<th>$t(c_{pda})$ (in ms)</th>
<th>$t(train)$ (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9a</td>
<td>460</td>
<td>24</td>
<td>4</td>
<td>0.5</td>
<td>1628.1</td>
<td>17.1</td>
<td>1674.2</td>
</tr>
<tr>
<td>covtype</td>
<td>2.1</td>
<td>1.89</td>
<td>0.02</td>
<td>0.04</td>
<td>120.18</td>
<td>0.36</td>
<td>122.50</td>
</tr>
</tbody>
</table>
Distributed QRSVM: Timing Discussions

Stage 1: Distributed QR
1. Computation: $t(p_{\text{localQR}}) + t(p_{\text{masterQR}})$
2. Communication: $t(c_{\text{gatherR}})$

Stage 2: Parallel Dual Ascent
1. Computation: $t(p_{\text{pda}})$
2. Communication: $t(c_{\text{pda}})$
   Gather+Scatter

<table>
<thead>
<tr>
<th>Time details</th>
<th>a9a (in ms)</th>
<th>covtype (in s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(p_{\text{meka}})$</td>
<td>460</td>
<td>2.1</td>
</tr>
<tr>
<td>$t(p_{\text{localQR}})$</td>
<td>24</td>
<td>1.89</td>
</tr>
<tr>
<td>$t(p_{\text{masterQR}})$</td>
<td>4</td>
<td>0.02</td>
</tr>
<tr>
<td>$t(c_{\text{gatherR}})$</td>
<td>0.5</td>
<td>0.04</td>
</tr>
<tr>
<td>$t(p_{\text{pda}})$</td>
<td>1628.1</td>
<td>120.18</td>
</tr>
<tr>
<td>$t(c_{\text{pda}})$</td>
<td>17.1</td>
<td>0.36</td>
</tr>
<tr>
<td>$t(\text{train})$</td>
<td>1674.2</td>
<td>122.50</td>
</tr>
</tbody>
</table>
## Distributed QRSVM: Parameter Discussions

<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>$2^{-1}$</td>
<td>$2^{-1}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$2^{-3}$</td>
<td>$2^3$</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>
Comparison with PSVM and P-packSVM ($S = 16$)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dis-QRSVM</th>
<th>PSVM</th>
<th>P-packSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>covType</td>
<td>2 min</td>
<td>20 min</td>
<td>16 min</td>
</tr>
</tbody>
</table>

Demerits of PSVM and P-packSVM

- PSVM uses Incomplete Cholesky Factorization (ICF) ⇒ Difficult to parallelize and slow ⇒ Unfit for distributed big data analytics
- PSVM training time is $O(n^2)$/iteration ⇒ Limited scalability
- P-packSVM solves *primal* form ⇒ Slow Convergence
Conclusions
Summary

1. Memory-efficient modeling and training for QRSVM
2. Parallel SVM formulation - distributed QR decomposition and Parallel Dual Ascent
3. Optimal Step size calculation for fast convergence and training
4. Performs significantly better than competing algorithms
## Summary

1. Memory-efficient modeling and training for QRSVM
2. Parallel SVM formulation - distributed QR decomposition and Parallel Dual Ascent
3. Optimal Step size calculation for fast convergence and training
4. Performs significantly better than competing algorithms

## Future Possibilities

1. Can be implemented in **clustered embedded systems/Edge-line devices** to solve large-scale problems rather than using supercomputers
2. QRSVM can be extended for **real-time** data analytics
3. QR decomposition technique can be used for other **Kernel based problems** like Support Vector Regression etc.
4. Motivates for designing **hardware accelerators** to further boost the performance in many domain specific scenarios.
Thank You!
MEKA

LSDA