QR DECOMPOSITION FRAMEWORK FOR EFFICIENT IMPLEMENTATION OF LINEAR SUPPORT VECTOR MACHINES USING DUAL ASCENT

A Thesis

by

VENKATA NAGA SAI PRITHVI SAKURU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Rabi N Mahapatra
Co-Chair of Committee,	Raktim Bhattacharya
Committee Member,	Vivek Sarin
Head of Department,	Dilma Da Silva

August 2016

Major Subject: Computer Science

ABSTRACT

Support Vector Machines (SVMs) is a popular method to solve standard machine learning tasks like classification, regression or clustering. There are many algorithms to solve the linear SVM classification problem. However, only a few algorithms are optimized on both per iteration cost and convergence. While fast convergence is essential for solving any optimization problem, per iteration cost is critical in resource-limited environments like dedicated embedded solutions for machine learning problems. In this thesis, we propose a novel approach to solve large-scale linear SVM classification problems. The proposed algorithm has low per iteration cost and also converges faster than existing state-of-art solvers.

There are two significant contributions from this thesis. First, we analyzed and improved the performance of the dual ascent (DA) algorithm, which would serve as the optimizing engine for solving SVM classification problem. An analytical model to evaluate the optimum step size and synchronization period for solving a generic quadratic programming optimization problem using DA is presented. Second, we implement SVM using the improved Dual Ascent algorithm. We also introduce a novel approach to tackle low dimensional classification problems of large data sizes via QR decomposition technique.

ACKNOWLEDGEMENTS

First and foremost I would like to express my gratitude to my advisor Dr. Rabi Mahapatra for the continuous support and motivation throughout the course of research. His experience and knowledge has taught me a lot and made be a better researcher. Besides my advisor, I would like to thank my committee co-chair Dr. Raktim Bhattacharya and committee member Dr. Vivek Sarin for their valuable ideas and inputs without which this research would not have been possible.

I would like to specially thank my fellow labmate Jyotikrishna Dass for all the stimulating discussions, long meetings relating to the research. These discussions prompted me to widen my research from various perspectives. Special thanks to Dr. Kooktae Lee for his insightful thoughts on the research and his motivating interactions.

I would also like to ackowledge Texas A&M High Performance Research Computing (http://hprc.tamu.edu/) for the computing platform and the resources used in performing necessary experiments.

Last but not the least I would like to express my heartfelt thanks to my parents, my sister and my friends for their love, encouragement and support throughout.

NOMENCLATURE

ADMM	Alternating Method of Multipliers
AWS	Amazon Web Services
DA	Dual Ascent Algorithm
DCD	Dual Coordinate Descent
EC2	Elastic Cloud Compute
HPC	High Performance Computing
IB	InfiniBand interconnect
IoT	Internet of Things
LAN	Local Area Network
LSDA	Lazy Synchronized Dual Ascent
MPI	Message Passing Interface
QP	Quadratic Programming
QR	QR decomposition
SMO	Sequential minimization optimization
SVM	Support Vector Machines

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

With the advancement in computing systems, several industries and organizations started to invest heavily in data analytics to develop data-driven solutions to problems such as expenditure balancing, real-time solutions to customer problems, business intelligence and also to explore new horizons to expand businesses. Machine learning has also interleaved itself into other areas of science and engineering. For example, genetic engineers extensively use data analytics for pattern recognition and gene modeling[22]. Computer architecture community is nowadays adopting machine learning techniques for branch prediction[13] and malware detection [15]. Hence, there is a constant need to improve the performance of such data analytic tasks either from an algorithmic perspective or through dedicated computing resources.

Large-scale optimization algorithms are the crux of many statistical and machine learning problems. Many of the optimization algorithms used in SVM modeling are iterative in nature. Several researchers target different aspects of the algorithm to improve the performance further. The classic trade off is the per-iteration cost and rate of convergence [11]. Most of the existing algorithms fall at the two extremities of this paradigm. The low per-iteration cost algorithms are used only for developing an approximate yet reasonable model while the fast convergence algorithms which can generate highly accurate models require powerful computing systems to deal with high computation costs. Hence, new algorithms are needed which can both reduce the per-iteration cost as well as have a quick rate of convergence.

A general shift towards dedicated hardware is observed in the recent years to further improve the performance of the optimization algorithms and thereby providing real-time solutions to these data-intensive problems [2, 27]. While a majority of them target a particular application, a few like [35] target a generic optimization engine and build a framework for modeling hardware. With the increasing computing power and rise in popularity of the Internet of Things (IoT), embedded data processors on smart devices are the need of the future. Such embedded data processors can be used to obtain meaningful information from the raw data collected by IoT sensor devices. Smart filtering mechanisms [3] are required in such scenarios to remove unnecessary/obvious data and only send important data points to the centralized IoT server. These filters can easily be modeled using embedded data processors. Such requirements reinforce the need for algorithms with low computation costs and good convergence rates based on which dedicated hardware can be built.

The future of computing systems is moving from today's symmetric multiprocessors to tightly integrated heterogeneous processor systems and further into exascale computing [19, 31]. The algorithms should, therefore, be adaptable to distribution, decomposition and parallelization with each task solving a subproblem in various multiprocessor architectures. These distributed algorithms rely on synchronization between processors to maintain consensus and this synchronization accounts for the majority of communication overhead in the form of idling across processors. This synchronization overhead in distributed optimization problems can sometimes account for almost 60% of the total execution times[32] and hence, become a bottleneck when scaled to a large number of processors. We present a relaxed synchronization approach to alleviate this communicate bottleneck and thereby improving the performance in distributed environments. As this relaxed synchronization methodology is generic and at an algorithmic level, distributed environments ranging from massive supercomputing clusters to small embedded multiprocessors can leverage its benefits.

In this thesis, we explore the capabilities of relaxed synchronization in case of distributed optimization using dual ascent algorithm. This distributed optimization framework will serve as the computing engine to solve an SVM classification problem. Based on the need and the nature of the problem, the distributed optimization framework can be scaled from a single processor to as large as the hardware can support. In case of single processor system, this relaxed synchronization methodology manifests itself as a technique to compute the *optimal step size*.

The main contributions of this thesis are:

1. Formulation of lazily synchronized dual ascent algorithm(LSDA) for solving a large-scale quadratic programming problem.

2. Experimental validation of the theoretical results to show that the number of iterations, and thereby execution time, can be significantly reduced using optimal step size for an optimization problem. Experimental results from two environments, 1) cloud cluster on Amazon web services(AWS) and 2) supercomputing cluster courtesy of Texas A&M High-Performance Research Computing, are be presented.

3. A novel QR decomposition approach to model the quadratic programming problem in linear large-scale SVM classifications (QR-SVM) and its solution via LSDA algorithm formulated in step (1).

4. Analytical derivation of optimal step size for solving QR-SVM in a single processor environment.

5. Comparision of performance and accuracy of the newly proposed QR-SVM with existing state of the art solver using standard, publicly available benchmarks.

2. RELATED WORK

2.1 Lazily Synchronized Dual Ascent

Dual Ascent is a standard approach for solving constrained optimization problem which dates back to 1960s. Recently, due to the advances in computing systems, there is a renewed interest in proximal techniques like dual ascent(DA), alternating direction method of multipliers(ADMM), Dykstra's alternating projections method and others[9] to solve large distributed optimization problems. Dual Decomposition is a powerful extension of DA algorithm which distributes the optimization problem into sub-problems. These sub-problems take the form of broadcast-gather architecture. Dantzig et.al [14] popularized dual decomposition methods for large-scale linear programming problems. Nedić et.al [36] discussed this approach in their recent survey on distributed techniques. Augmented Lagrangian and method of multipliers improve the dual ascent algorithm making it more robust to solve optimization problems which are not strictly convex[9]. ADMM is a manifestation of the method of multipliers which has been used in several fields to solve distributed optimization problems including SVM [45]. Goldstein et.al. [23] proposed some extensions and parameter selection techniques to improve the performance of ADMM and Alternating Minimization Algorithms(AMA).

The traditional dual decomposition method of dual ascent does distribute the optimization problem. However, it requires us to synchronize after every iteration step. Lazily Synchronized Dual Ascent(LSDA) relaxes this constraint by synchronizing at an interval of P, thereby reducing the communication between processor nodes. Our experiments show that we can reduce the communications up to almost 90% when dealing with well-conditioned optimization problems. While the LSDA algorithm is developed in the spirit of distributed systems, we can extend the same analytical results even for a non-distributed, single processor systems. It can be shown that the analytical derivations in such single processor systems lead to optimal step size selection (analogous to optimal synchronization period in distributed systems) for the optimization problem.

In this thesis, we provide analytical derivation to obtain the optimal synchronization period P for the LSDA algorithm and empirically validate these results in a distributed environment with varying cluster sizes.

2.2 QR-SVM Framework

Support Vector Machines(SVM) close to the current form was introduced by Boser, Guyon and Vapnik [7] in 1992. Over the next few years, the model was developed by introducing the kernel trick, loss functions and slack variables to improve the performance and robustness of the classifiers[5]. However, with the growing dataset sizes, the performance of SVM solvers hit a snag. To counter these performance issues, Vapnik [12] suggested a chunking approach to reducing the dataset sizes by purging non-support vector data points from the training sets. Osuna et.al [18] proposed operating on subsets individually one at a time, while rest of the data points and their weights are left unchanged. One of the most prominent contributions on algorithmic enhancements for SVMs was the sequential minimization optimization (SMO) by Platt [37] which takes the decomposition method proposed by Osuna to the extreme and optimizes only two points at a time. LIBSVM, which incorporates SMO, is a popular machine learning library targeted mainly for beginners in the field of machine learning [10]. There have been several extensions and hybrids of SMO improving the performance and stability further. One of the prominent modifications widely used was proposed by Keerthi et.al. [30].

Recently, much of the research in this area is targetted towards solving linear SVM classifiers rather than non-linear classifiers. As clearly explained in [44], for large scale SVM problems, training a non-linear classifier is a tedious and time-taking process. It has been observed that for several rich dimensional datasets, the accuracy of linear SVM classifiers is on par with that of non-linear SVM models. Also, training linear classifiers is much faster as it does not involve kernel transformations on the data points. Bottou [8] presented a stochastic gradient descent approach for solving linear SVM. Joachims [28] introduced a cutting plane method for solving large-scale linear SVM problems. SVM^{perf} is an implementation of this cutting plane technique. Hsieh et.al [25] presented a coordinate decent approach for solving the dual problem of SVM and showed that they outperform other classifiers when dealing with high dimensional datasets. LIBLINEAR [20] is the publicly available library which implements this dual coordinate descent (DCD) method. However, it was found that the dual coordinate descent method is not stable for non-document datasets especially when the number of dimensions is low [11].

In this thesis, we present a novel QR-SVM approach to solving the dual SVM problem targeting problems which cannot be solved using DCD method, i.e., datasets with low dimensionality.

3. LAZILY SYNCHRONIZED DUAL ASCENT [*]

3.1 Introduction - Dual Ascent

Dual Ascent(DA) is an algorithm to solve linear constrained convex optimization problems. Consider a convex optimization problem

$$\begin{aligned} &\min_{\mathbf{x}} \;\; f(\mathbf{x}) \\ &\text{subject to } A\mathbf{x} = \mathbf{b} \end{aligned} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the primal variable which minimizes the objective function f while satisfying the constraint $A\mathbf{x} = \mathbf{b}$ where $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^n$. It should be noted that the function $f : \mathbb{R}^n \to \mathbb{R}$ is a strict convex function and hence any local minimum is in fact a global minimum.

The Lagrangian for problem (3.1) is given as

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \mathbf{y}^T (A\mathbf{x} - \mathbf{b}) \tag{3.2}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the Lagrangian or dual variable. The size of dual variable $\mathbf{y}$ is dependent on the number of constraints in the original optimization problem, m.

Solving the optimization problem in (3.1) is equivalent to solving its dual problem as given below:

$$\max_{\mathbf{y}} \;\; \left(\inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{y}) \right) \tag{3.3}$$

If $\mathbf{y}^*$ is the optimal solution to the dual problem (3.3) and assuming strong duality

[*] Part of the content in this chapter is reprinted with permission from "A relaxed synchronization approach for solving parallel quadratic programming problems with guaranteed convergence" by Kooktae Lee, Raktim Bhattacharya, Jyotikrishna Dass, V N S Prithvi Sakuru, and Rabi N Mahapatra. 30th International Symposium on Parallel & Distributed Processing (IPDPS), © 2016 IEEE.

holds, the optimal primal variable $\mathbf{x}^*$ can be obtained by solving

$$\mathbf{x}^* = arg \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{y}^*)$$

Finally, to solve the optimization problem for $\mathbf{x}^*$ and $\mathbf{y}^*$ the following gradient steps can be used:

$$\begin{aligned} \mathbf{x}^{k+1} &= \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{y}^k) \\ \mathbf{y}^{k+1} &= \mathbf{y}^k + \eta^k (A\mathbf{x}^{k+1} - \mathbf{b}) \end{aligned} \tag{3.4}$$

where $\eta^k > 0$ is the step size for the k^{th} iteration. These update steps are continued till the dual variable $\mathbf{y}$ converges to an ϵ-accurate solution i.e., $|\mathbf{y}^{k+1} - \mathbf{y}^k| \leq \epsilon$. Notice that the optimization problem considered is an equality constrained problem. By ensuring each of the dual variable $\mathbf{y}_i$ is greater than or equal to 0, the dual ascent algorithm can be extended to inequality constrained ($A\mathbf{x} \leq \mathbf{b}$) optimization problem i.e., the update steps for an inequality constrained optimization problem are as follows:

$$\begin{aligned} \mathbf{x}^{k+1} &= arg \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{y}^k) \\ \mathbf{y}_i^{k+1} &= max(0, \hat{\mathbf{y}}_i^{k+1}) \end{aligned} \tag{3.5}$$

where $\hat{\mathbf{y}}^{k+1} = \mathbf{y}^k + \eta^k (A\mathbf{x}^{k+1} - \mathbf{b})$.

3.1.1 Dual Decomposition

Dual Decomposition is a powerful extension of dual ascent algorithm used to distribute the optimization problem into sub-problems each of which can be solved in parallel. If the convex function f in (3.1) is separable, then the objective of the minimization problem can be written as

$$f(\mathbf{x}) = \sum_i f_i(\mathbf{x}_i)$$

where primal variable $\mathbf{x} = (\mathbf{x}_1; \mathbf{x}_2; ...\mathbf{x}_i; ...)$. Each partitioned sub-vector $\mathbf{x}_i$ can be updated in parallel without any dependency on other sub-vector partitions.

The coefficient matrix in the constraint equation is trivially separable and can be written as

$$A = [A_1, A_2, ...A_i, ...]$$

Hence, the update steps in case of dual decomposition can be formulated as:

$$\begin{aligned} \mathbf{x}_i^{k+1} &= arg \min_{\mathbf{x}_i} \mathcal{L}_i(\mathbf{x}_i, \mathbf{y}^k) \\ \mathbf{y}^{k+1} &= \mathbf{y}^k + \sum_{i=1}^{N} \eta_i^k \Big(A_i \mathbf{x}_i^{k+1} - \frac{\mathbf{b}}{N} \Big) \end{aligned} \tag{3.6}$$

where N is the number of sub-vector partitions of variable $\mathbf{x}$. As clearly observed in (3.8), each of the $\mathbf{x}$ update can be done in parallel by independent worker nodes. However, for dual variable update, the components of $\mathbf{x}$ from each individual worker nodes should be gathered. The newly updated value of dual variable should then again be broadcasted to the worker nodes for the next iteration. Hence, dual decomposition clearly follows a broadcast-gather architecture in distributed environments.

3.1.2 Quadratic Programming Using Dual Ascent

This section explores a specific example of dual ascent; solving a quadratic programming problem using dual ascent. A quadratic programming (QP) problem is an optimization problem which minimizes or maximizes a quadratic objective function subject to linear constraints. A standard QP problem can be formulated as

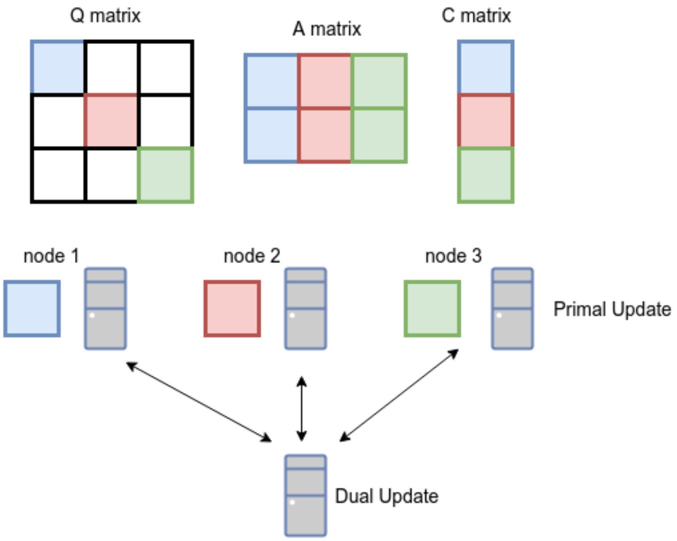

Figure 3.1: Process distribution of a separable QP problem.

$$\begin{aligned} &\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ &\text{subject to } A\mathbf{x} = \mathbf{b} \end{aligned} \tag{3.7}$$

The update steps for solving the QP problem (3.7) using dual ascent are

$$\mathbf{x}^{k+1} = arg\min_{\mathbf{x}_i} \mathcal{L}_i(\mathbf{x}_i, \mathbf{y}^k) = -Q^{-1}(A^T\mathbf{y}^k + \mathbf{c})$$

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \eta^k(A\mathbf{x}^{k+1} - \mathbf{b})$$

To apply dual decomposition, the objective function f should be separable. The objective function in (3.7) is separable if the Q matrix is block diagonal. A block diagonal matrix Q would imply that the components of $\mathbf{x}$ pertaining to one block are independent of the components in the other block and hence can be solved in parallel. The update steps in case of a block diagonal Q matrix are

$$\begin{aligned} \mathbf{x}_i^{k+1} &= -Q_i^{-1}(A_i^T\mathbf{y}^k + \mathbf{c}_i) \\ \mathbf{y}^{k+1} &= \mathbf{y}^k + \Big[\sum_{i=1}^{N} \eta_i \Big(A_i\mathbf{x}_i^{k+1} - \frac{\mathbf{b}}{N}\Big)\Big] \end{aligned} \tag{3.8}$$

where Q_i is the i$^{\text{th}}$ block in the block diagonal matrix Q. Figure 3.1 pictorially shows this distribution.

3.2 Proposed - Lazily Synchronized Dual Ascent

This section details the proposed lazy synchronization technique and provides the analytical analysis of the algorithm for Quadratic Programming (QP) problems. From equations in (3.8), it is observed that the update equation of $\mathbf{y}$ can be interpreted as part of a discrete-time system and the convergence of the iterations is dependent on the stability of this system. Such analogy between discrete-time systems and iterative algorithms can be observed in Lee et.al. [33].

In a typical implementation of equation (3.8), x-updates are computed in parallel and then synchronized to calculate the $\mathbf{y}$-update. This sequence is carried out for every iteration. Lazy synchronization strategy relaxes this requirement and the x-updates are synchronized only at certain time period P. Hence, the update equation of the dual variable $\mathbf{y}$ can now be written as

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \Big[\sum_{i=1}^{N} \eta_i \Big(A_i\mathbf{x}_i^{tP+1} - \frac{\mathbf{b}}{N}\Big)\Big] \tag{3.9}$$

$$\text{where } tP \leq k < (t+1)P, \quad t \in \mathbb{N}_0$$

The total iterations can be grouped into synchronization steps and intermediate

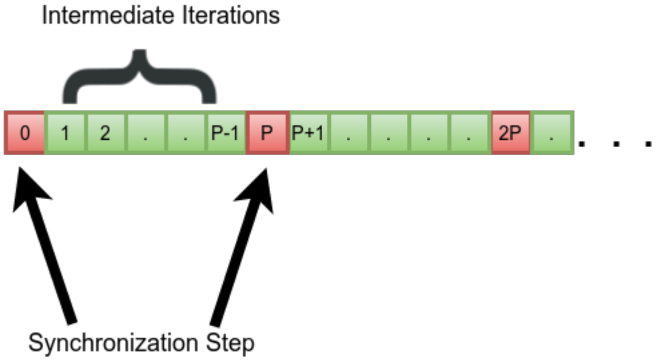

Figure 3.2: Distribution of iterations of dual ascent algorithm with a synchronization period P

steps as shown in figure 3.2. So, for a given intermediate iteration i.e., $tP \leq k < (t+1)P$, $\mathbf{y}$ update is carried out with the most recently updated values of $\mathbf{x}_i$, $\mathbf{x}_i^{tP+1}$. On reaching the next synchronization step, $(t+1)P$, each individual $\mathbf{x}_i$ is computed and communicated across nodes, and the updated values i.e., $\mathbf{x}_i^{(t+1)P+1}$ are now used for the subsequent $\mathbf{y}$ updates during the itermediate iterations.

3.2.1 Stopping Criteria of LSDA

The algorithm is bound to terminate when change in the dual variable is small defined by

$$||\mathbf{y}^{k+1} - \mathbf{y}^k||_2 \leq \epsilon$$

where ϵ is the stopping threshold. From equation (3.9), it can be observed that during intermediate iterations that the change in the dual variable is a constant determined by $\mathbf{x}^{tP+1}$. Hence, the stopping criterion of the LSDA algorithm can be redefined as

$$\Big|\Big|\sum_{i=1}^{N}\eta_i\Big(A_i\mathbf{x}_i^{tP+1}-\frac{\mathbf{b}}{N}\Big)\Big|\Big|_2\leq\epsilon \tag{3.10}$$

Note that the convergence is bound to occur only on the synchronization steps when $\mathbf{x}^{tP+1}$ changes to $\mathbf{x}^{(t+1)P+1}$, as $\mathbf{x}$ is the only variable in equation (3.10). The error during the intermediate iterations is fixed and same as that of the error at the last synchronization step.

3.2.2 Stability of LSDA

Stability of LSDA determines the algorithms ability to converge to the optimal solution. Increasing the synchronization period, P decreases the amount of communication between iterations. However, if the period P crosses a certain threshold, then dual variable $\mathbf{y}$ will start to diverge and the algorithm will fail to converge to the optimum value. The below lemma 1 establishes the stability of LSDA algorithm.

Lemma 1 *[32] The dual variable* $\boldsymbol{y}$ *update for LSDA algorithm is stable if and only if*

$$\rho\Big(I-P\sum_{i=1}^{N}\eta_i(A_iQ_i^{-1}A_i^T)\Big)<1 \tag{3.11}$$

where $\rho(\cdot)$ *represents the spectral radius.*

Proof: Consider a discrete-time dynamic system of the form $\mathbf{y}^{k+1}=A\mathbf{y}^k+\mathbf{b}$, where A is a time-invariant matrix and b is a constant vector. The stability of such a system is determined by the spectral radius of the matrix A. $\mathbf{y}^k$ converges to $\mathbf{y}^*$ if and only if the spectral radius of A is less than 1 i.e., $\rho(A)<1$.

From equation (3.8), $\mathbf{x}_i^{tP+1}=-Q_i^{-1}(A_i^T\mathbf{y}_i^{tP}+\mathbf{c}_i)$. Subsituting $\mathbf{x}_i^{tP+1}$ in equation (3.9), the dual variable update equation can be written as

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \sum_{i=1}^{N} \eta_i \Big(- A_i Q_i^{-1} (A_i^T \mathbf{y}^{tP} + \mathbf{c}_i) - \frac{\mathbf{b}}{N} \Big)$$
$$\text{where } tP \leq k < (t+1)P$$

Towards the end of the iterations in a synchronization period i.e. $k = (t+1)P-1$, the $\mathbf{y}$ update can be analytically written as

$$\begin{aligned} \mathbf{y}^{(t+1)P} &= \mathbf{y}^{tP} + P \sum_{i=1}^{N} \eta_i \Big(- A_i Q_i^{-1} (A_i^T \mathbf{y}^{tP} + \mathbf{c}_i) - \frac{\mathbf{b}}{N} \Big) \\ &= \mathbf{y}^{tP} + P \sum_{i=1}^{N} \eta_i \Big(- A_i Q_i^{-1} A_i^T \mathbf{y}^{tP} \Big) + P \sum_{i=1}^{N} \eta_i \Big(- A_i Q_i^{-1} \mathbf{c}_i - \frac{\mathbf{b}}{N} \Big) \\ &= \mathbf{y}^{tP} - P \sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1} A_i^T \Big) \mathbf{y}^{tP} - P \sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1} \mathbf{c}_i + \frac{\mathbf{b}}{N} \Big) \end{aligned} \tag{3.12}$$

$$\Rightarrow \mathbf{y}^{(t+1)P} = \Big(I - P \sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T) \Big) \mathbf{y}^{tP} - P \sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1} \mathbf{c}_i + \frac{\mathbf{b}}{N} \Big) \tag{3.13}$$

From the above equation (3.13), the stability of LSDA algorithm can be determined as

$$\rho \Big(I - P \sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T) \Big) < 1$$

3.2.3 Convergence of LSDA

While the stability analysis in section 3.2.2 ensures convergence of LSDA algorithm, convergence to the same optimal solution as that of the conventional dual ascent algorithm can be proved through the following proposition.

Proposition 1 *[32] Given a QP problem and ensuring the stability of LSDA as defined in lemma 1 for that QP problem, the dual variable* $\boldsymbol{y}$ *of LSDA algorithm* *converges to the same optimal value as that of conventional DA algorithm.*

Proof: As the stability condition for LSDA algorithm is satisfied, it is bound to converge to a certain value of dual variable $\mathbf{y}$ as $t \to \infty$

$$\mathbf{y}^* = \lim_{t\to\infty} \mathbf{y}^{(t+1)P}$$

From equation (3.13)

$$\begin{aligned}
&\Rightarrow \mathbf{y}^* = \Big(I - P\sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T)\Big)\mathbf{y}^* - P\sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1}\mathbf{c}_i + \frac{\mathbf{b}}{N}\Big) \\
&\Rightarrow P\Big(\sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T)\Big)\mathbf{y}^* = -P\sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1}\mathbf{c}_i + \frac{\mathbf{b}}{N}\Big) \\
&\Rightarrow \Big(\sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T)\Big)\mathbf{y}^* = -\sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1}\mathbf{c}_i + \frac{b}{N}\Big) \\
&\Rightarrow \mathbf{y}^* = -\Big(\sum_{i=1}^{N} \eta_i (A_i Q_i^{-1} A_i^T)\Big)^{-1} \sum_{i=1}^{N} \eta_i \Big(A_i Q_i^{-1}\mathbf{c}_i + \frac{\mathbf{b}}{N}\Big) \qquad (3.14)
\end{aligned}$$

A similar analysis of dual update equation of conventional DA algorithm shown in (3.8) would result in the same optimal value of dual variable $\mathbf{y}^*$. Hence, it can be concluded that both LSDA and conventional DA algorithms converge to the same optimal value of dual variable $\mathbf{y}^*$.

While a direct analytical result of $\mathbf{y}^*$ can directly be obtained from (3.14) iteration are often preferred as calculation of inverse might be expensive and practically infeasible for large QP problems.

3.2.4 Optimal Synchronization Period

Optimal Synchronization Period, P^* corresponds to that synchronization period with would minimize the communication between the nodes while also ensuring fast convergence of the algorithm. Consider the matrix $M = \sum_{i=1}^{N} \eta_i A_i Q_i^{-1} A_i^T$. From lemma 1, it has been established that the stability of the LSDA is dependent on the spectral radius of M and that the spectral radius should be less than 1. It follows that, lower the spectral radius of matrix $(I - MP)$, higher is the stability of the LSDA algorithm and hence, the algorithm is bound to converge faster. Hence, the optimal synchronization period of LSDA is that period P^* where the spectral radius is minimum. It should be noted that the domain of P is the set of Natural numbers i.e., $P \in \mathbb{N}$ and if $P^* = 1$ then the LSDA algorithm behaves similar to the conventional DA algorithm.

Theorem 1 *[32] For a given separable QP problem, the optimal synchronization period P^* for LSDA algorithm is given as*

$$P^* = \max \arg \min_{P \in \mathbb{N}} \ \max\{|1 - \underline{\lambda}(M)P|, |1 - \bar{\lambda}(M)P|\} \tag{3.15}$$

where $M = \sum_{i=1}^{N} \eta_i A_i Q_i^{-1} A_i^T$, $\underline{\lambda}(\cdot)$ and $n\bar{\lambda}(\cdot)$ represent the minimum and maximum eigenvalues of the matrix respectively.

Proof: The optimal synchronization period P^* corresponds to the minimal spectral radius of matrix $(I - MP)$. This can be analytically written as

$$P^* = \arg \min_{P \in \mathbb{N}} \rho(I - MP)$$

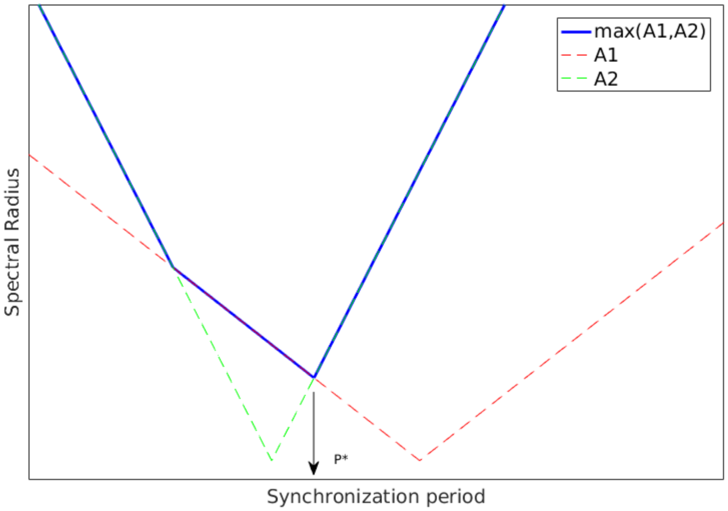

Figure 3.3: Optimal synchronization period, P^* as derived in equation (3.16). Here, $A1 = |1 - \underline{\lambda}(M)P|$ and $A2 = |1 - \bar{\lambda}(M)P|$.

From the definition of spectral radius,

$$
\begin{aligned}
P^* &= \arg\min_{P\in\mathbb{N}} \max_{\mathbf{v}} \left| \frac{\mathbf{v}^T(I - MP)\mathbf{v}}{||\mathbf{v}||^2} \right| \\
&= \arg\min_{P\in\mathbb{N}} \max_{\mathbf{v}} \left| 1 - \frac{\mathbf{v}^T(MP)\mathbf{v}}{||\mathbf{v}||^2} \right| \\
&= \arg\min_{P\in\mathbb{N}} \max_{\mathbf{v}} \left| 1 - \frac{\mathbf{v}^T M\mathbf{v}}{||\mathbf{v}||^2} P \right|
\end{aligned}
$$

where $\mathbf{v}$ is the eignevector of the matrix $(I - MP)$. Given the symmetric semi-positive definite matrix M, the below expression is valid

$$
\underline{\lambda}(M) \leq \frac{\mathbf{v}^T M\mathbf{v}}{||\mathbf{v}||^2} \leq \bar{\lambda}(M)
$$

$$
\Rightarrow \underline{\lambda}(M)P \leq \frac{\mathbf{v}^T M\mathbf{v}}{||\mathbf{v}||^2} P \leq \bar{\lambda}(M)P
$$

Hence, the optimal synchronization period P^* is given as

$$P^* = \arg\min_{P \in \mathbb{N}} \max\{|1 - \underline{\lambda}(M)P|, |1 - \bar{\lambda}(M)P|\} \tag{3.16}$$

The maximum of the possible synchronization period is chosen in such cases to ensure least amount of communication. Figure 3.3 is a graphical presentation of the expression in (3.16).

3.2.5 Theoretical Speedup of LSDA

To determine the theoretical speedup of LSDA algorithm we use the below definitions

- k^*_{DA}: the total number of iterations up to termination of Dual Ascent algorithm.
- k^*_{LSDA}: the total number of iterations up to termination of LSDA algorithm.
- t^p_{DA}: computation time per iteration of Dual Ascent algorithm.
- t^p_{LSDA}: computation time per iteration of LSDA algorithm.
- t^c_{DA}: communication time per iteration of Dual Ascent algorithm.
- t^c_{LSDA}: communication time per iteration of LSDA algorithm.
- T_{DA}: Total execution time of Dual Ascent algorithm.
- T_{LSDA}: Total execution time of LSDA algorithm.

The total execution time of conventional dual ascent algorithm is given as

$$T_{DA} = k^*_{DA}(t^p_{DA} + t^c_{DA})$$

and the total execution time of LSDA algiorithm is given as

$$T_{LSDA} = k^*_{LSDA}\Big(t^p_{LSDA} + \frac{t^c_{LSDA}}{P^*}\Big)$$

We divide the per iteration communication time by the synchronization period P^* because out of the total number of iteration we communicate only at time intervals of P^*. Hence, the speedup is given as

$$\begin{aligned}
\text{Speedup } &= \frac{T_{DA}}{T_{LSDA}} \\
&= \frac{k^*_{DA}(t^p_{DA} + t^c_{DA})}{k^*_{LSDA}\left(t^p_{LSDA} + \frac{t^c_{LSDA}}{P^*}\right)}
\end{aligned} \tag{3.17}$$

Substituting $r_k = k^*_{DA}/k^*_{LSDA}$, $r_p = t^p_{DA}/t^p_{LSDA}$, $r_{DA} = t^c_{DA}/t^p_{DA}$ and $r_{LSDA} = t^c_{LSDA}/t^p_{LSDA}$ in (3.17)

$$\text{Speedup } = r_k r_p \left(\frac{1 + r_{DA}}{1 + \frac{r_{LSDA}}{P^*}} \right) \tag{3.18}$$

From equation (3.18), it can be observed that with an increase in the synchronization period, the speedup also increases assuming the ratio of the number of iterations r_k is fixed with the change in synchronization period. Cluster size N defines the number of subproblems the original QP problem is divided into. With an increase in the cluster size, the amount of computation required on each cluster node decreases while the communication increases as now we need to synchronize between larger number of nodes. Hence, the ratio of communication time to computation time for both DA algorithm (r_{DA}) and LSDA algorithm (r_{LSDA}) increases. But due to the factor P^*, it is observed that the speedup tends to increase with cluster size N as long as the computation time is dominant over communication time. But at a certain cluster size, communication time dominates over the computation time and the speedup thereafter tends to flatten out. This trend can be observed in figure 3.4.

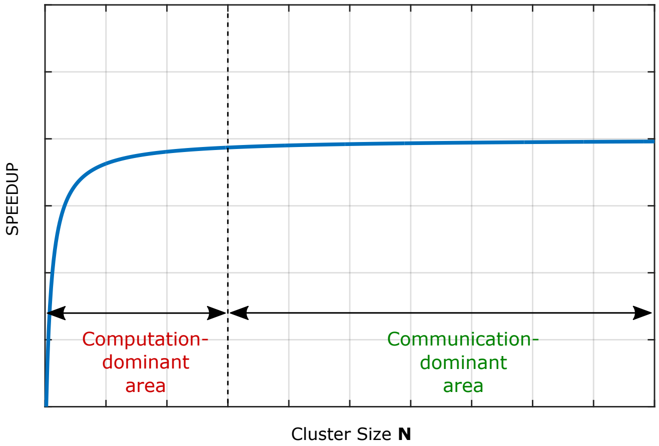

Figure 3.4: Theoretical speedup of LSDA when compared with conventional DA algorithm.

3.2.6 LSDA in a Single Processor Environment

The idea of lazy synchronization is presented mainly targeting distributed environments where the cluster sizes are greater than 1. However, the concept of minimum spectral radius for fast convergence can also be applied to dual ascent algorithm in single processor environments. From equation (3.12), substituting $N = 1$ we get

$$\mathbf{y}^{(t+1)P} = \mathbf{y}^{tP} + P\eta_1\Big(- A_1Q_1^{-1}(A_1^T\mathbf{y}^{tP} + \mathbf{c}_1) - \mathbf{b}\Big) \tag{3.19}$$

Redefining Q_1 as Q, A_1 as A, c_1 as c and considering $P\eta_1$ as $\bar{\eta}$, equation (3.19) is analogus to a conventional DA update step with a step size of $\bar{\eta}$

$$\mathbf{y}^{(t+1)P} = \mathbf{y}^{tP} + \bar{\eta}\Big(- AQ^{-1}(A^T\mathbf{y}^{tP} + \mathbf{c}) - \mathbf{b}\Big)$$

By computing the optimal synchronization period as discussed in section 3.2.4 in this setting, we would obtain optimal step size for the dual ascent algorithm in the form of $\eta^* = P^*\eta$.

3.3 Experimental Results

The experiments to validate the analytical results of LSDA algorithm were performed on two different platforms with varying cluster sizes $N = \{10, 20, 32, 40\}$. The QP problem was fixed throughout the experiments. First, the tests were run on Amazon Cloud Compute cluster platform with conventional gigabit interconnect. This platform was close to everyday commodity computers connected with gigabit ethernet connections. The second set of tests were performed in HPC cluster platform which supports InfiniBand Communication. Both LSDA and DA algorithms converge to the same solution of the dual variable on termination. Irrespective of cluster size and platform LSDA algorithm consistently outperformed DA algorithm while converging to the same ϵ-accurate solution.

3.3.1 Implementation

The LSDA and DA algorithms were implemented in C++11 supported by Armadillo(v5.4002) [39] linear algebra library. The Communication across nodes was handled via Message Passing Interface (MPICHv3)[42]. Algorithm 1 shows the pseudocode for the proposed LSDA algorithm.

The **AllReduce** method is used for synchronization between the nodes. This **AllReduce** operation uses the SUM operator to combine multiple *sumLocal*s to compute *sumGlobal*. **AllReduce** is a blocking operation which implies all the individual nodes wait till the **AllReduce** operation is complete, thereby ensuring guaranteed synchronization when invoked. As clear from the pseudocode line 11, termination of LSDA algorithm is only dependant on the *sumGlobal* value which changes only when the iteration $k = tP$, where $t = 0, 1, 2, 3....$ Hence, the LSDA terminates only during a synchronization step ($k = tP$) and not during intermediate

Algorithm 1 LSDA algorithm

```
function LSDA($Q_i$,$A_i$,$\mathbf{c}_i$,$\mathbf{b}$,$P$,$\epsilon$,$\eta_i$)
    $y \leftarrow \mathbf{y}_0; error \leftarrow \infty; k = 0;$
    $sumLocal \leftarrow \bar{\mathbf{0}}; sumGlobal \leftarrow \bar{\mathbf{0}};$
    while $error > \epsilon$ do
        if k%P == 0 then
            $\mathbf{x}_i = -Q_i^{-1}(A_i^T \mathbf{y} + \mathbf{c}_i)$
            $sumLocal = \eta_i(A_i \mathbf{x}_i + \frac{\mathbf{b}}{N})$
            AllReduce($sumLocal, sumGlobal$)
        $\mathbf{y} = \mathbf{y} + sumGlobal$
        $error = |sumGlobal|_2$
        $k+ = 1$
```

iterations $(tP < k < (t+1)P)$.

3.3.2 Experimental Setup and Hardware

Synthetically generated random matrices with values uniformly distributed between [-1,1] were used as input datasets for QP problem. The problem specifics are

1. Number of instances, n = 200,000.

2. Step size, $\eta = 0.27$.

3. Optimal Synchronization Period, $P^* = 70$.

4. Stopping threshold, ϵ = 1e-5.

5. Cluster Size, N = {10,20,32,40}

Cluster sizes were chosen such that the data can be evenly distributed among the cluster nodes i.e., factors of n.

The experiments were run on two different hardware platforms − 1) Amazon Cloud Platform and 2) HPC Cluster Platform.

3.3.2.1 Amazon Cloud Platform

40 node Amazon Web Services(AWS) Elastic Cloud Compute(EC2) instances were used[26]. The EC2 instances were created using the t2.micro configuration with each instance backed by Intel Xeon processors with a clock speed of 3.33GHz. Each instance was supported by 8GB disk memory with 1GB main memory. The instances were connected using gigabit ethernet and originated from the same data center in Oregon. This hardware platform was aimed to mimic the commodity computers connected using basic LAN.

3.3.2.2 HPC Cluster

EOS supercomputing cluster provided by Texas A&M High Performance Research Computing consists of 8-core 64-bit Nehalem processor. The nodes were connected by InfiniBand (IB) interconnect with a duplex speed of up to 5 GiB/s. Intel MPI was used in this case rather than MPICH as it supports IB interconnect. Though the EOS cluster offers computer nodes up to 314, we restricted the experiments to 40 nodes to ensure a fair comparison with AWS platform. Figure 3.5 shows a block schematic of the EOS cluster.

3.3.3 Results and Discussion

Results include analysis of 1) Synchronization Period, 2) Computation Time, 3) Communication Time and 4) Speedup with varying cluster sizes to compare the performance of LSDA algorithm against DA algorithm.

3.3.3.1 Synchronization Period

The number of iterations skipped between two successive inter-node communication is defined by the synchronization period P.

http://hprc.tamu.edu/

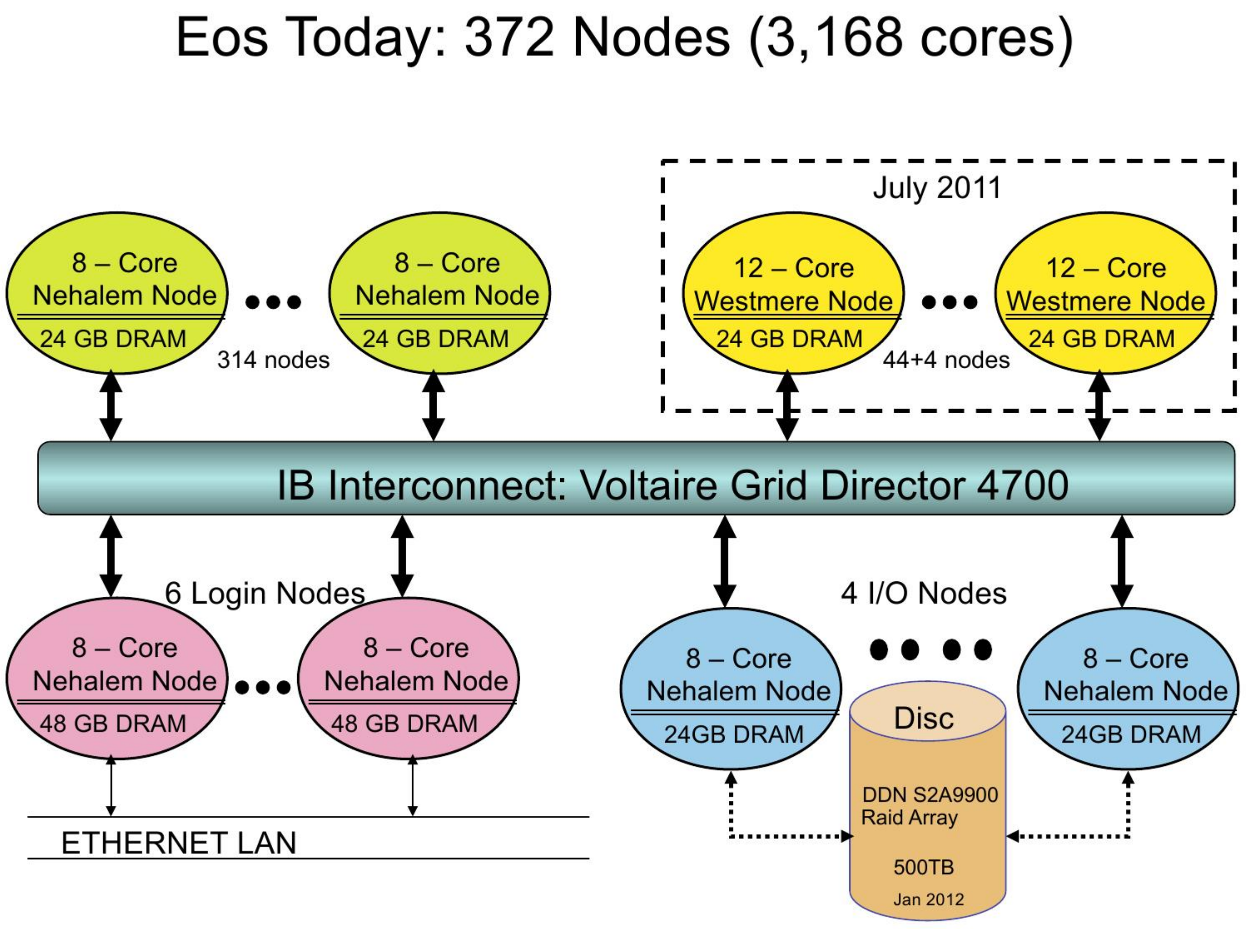

Figure 3.5: Block schematic of EOS cluster. Source: Texas A&M High-Performance Research Computing (http://hprc.tamu.edu/)

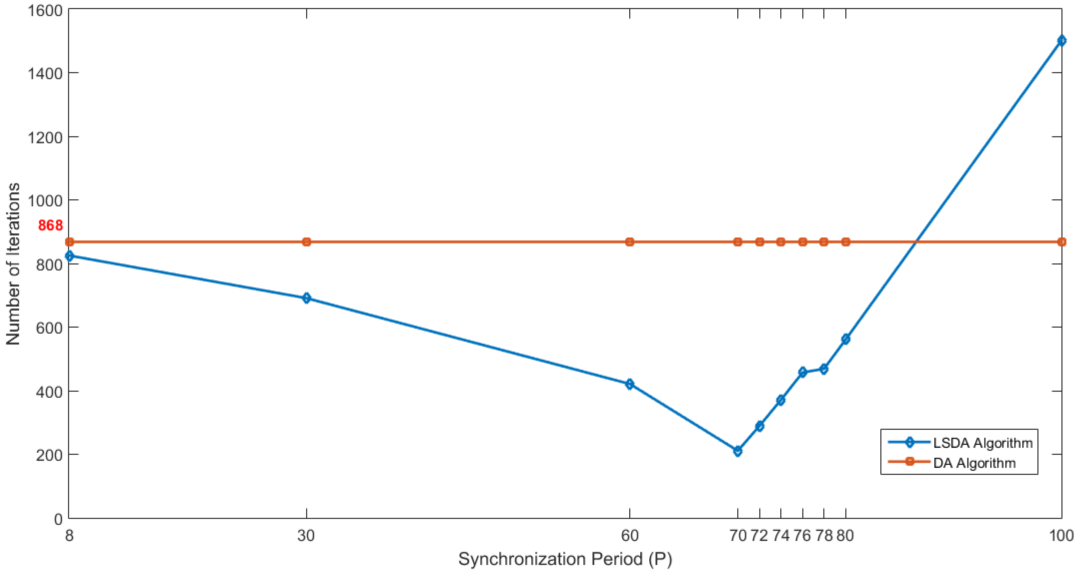

Figure 3.6: Variation of number of iterations to converge with synchronization period.

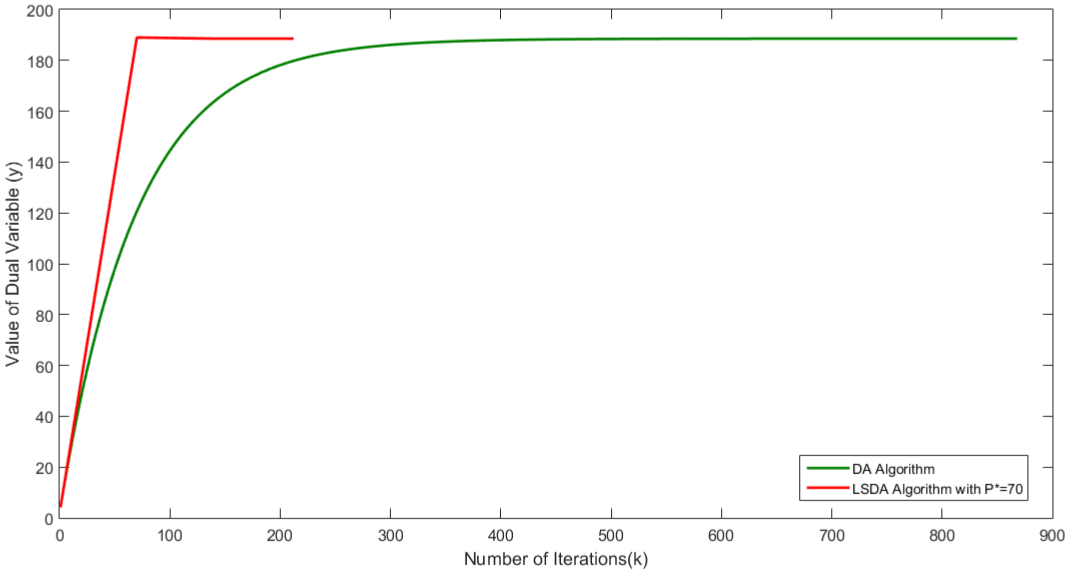

Figure 3.7: Convergence of LSDA algorithm and DA algorithm. LSDA algorithm approaches the solution significantly faster than DA algorithm.

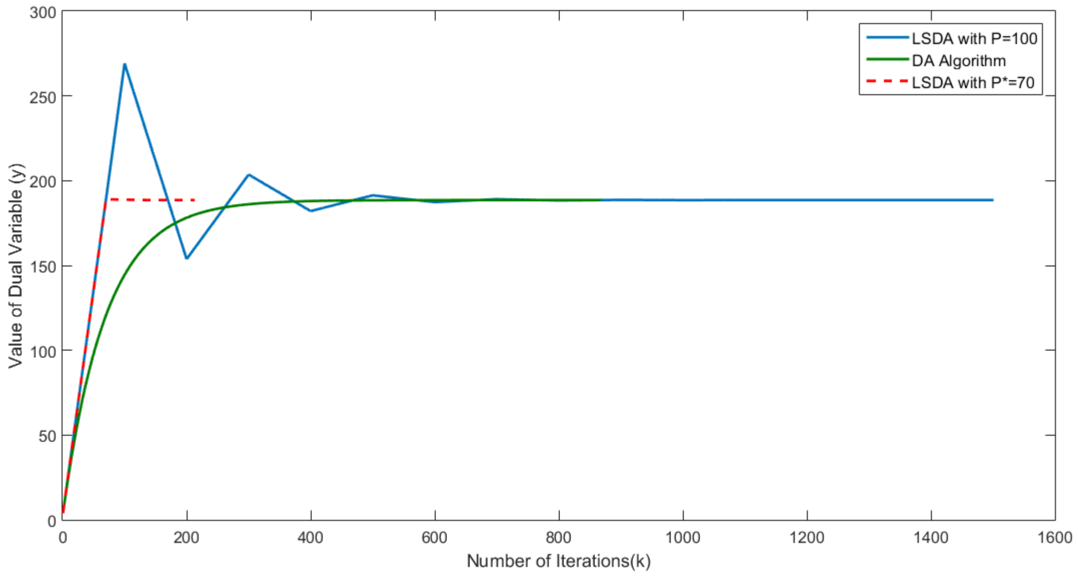

Figure 3.8: Convergence of LSDA algorithm with synchronization periods 70(optimal) and 100. It was observed that for synchronization period 100, the convergence is slower than DA algorithm.

It was observed that synchronization period was the only factor effecting the number of iterations when stopping threshold (ϵ) and step size (η) are fixed. The cluster size should have no effect on the termination of LSDA algorithm as expected. This is because *sumGlobal* in Algorithm 1 would remain the same irrespective of the number of cluster nodes. Figure 3.6 shows the variation of the number of iterations to converge with cluster size. This is close to the theoretical result shown in figure 3.3. It is observed that for the synthetic dataset considered the optimal Synchronization period is 70. Also, from theoretical derivation, it was found that the matrix M for this dataset (from section 3.2.4) is a scalar value of 0.0143 which leads to optimal synchronization period to be 70. This validates the theoretical results on optimal synchronization period.

Figure 3.7 shows the convergence trend of LSDA algorithm (with $P = 70$) and

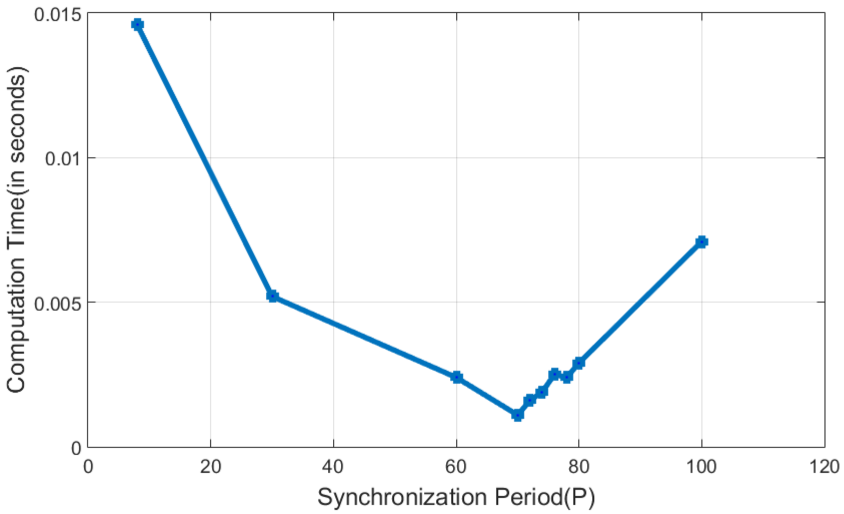

Figure 3.9: Variation of computation time and synchronization period

DA algorithm. The algorithm converges to the optimal solution $\mathbf{y}^*$ in 211 iterations while DA took 868 iterations. This behavior validates that LSDA algorithm is both numerically stable and accurate even with the relaxed synchronizations. Figure 3.8 shows that for synchronization periods much larger than the optimal period, the conventional DA algorithm might perform better than LSDA algorithm.

3.3.3.2 Computation Time

The time spent in calculations other than the **AllReduce** step in Algorithm 1 is considered as Computation time.

Variation with Synchronization Period: For a given cluster size and changing synchronization period, we observe that computation time follows the similar trend as that of the number of iterations. This is evident from figures 3.9 and 3.6. The minimum computation time occurs at a synchronization period of 70 (optimal value). The variation of computation time with cluster size and synchronization period can be seen in figure 3.12.

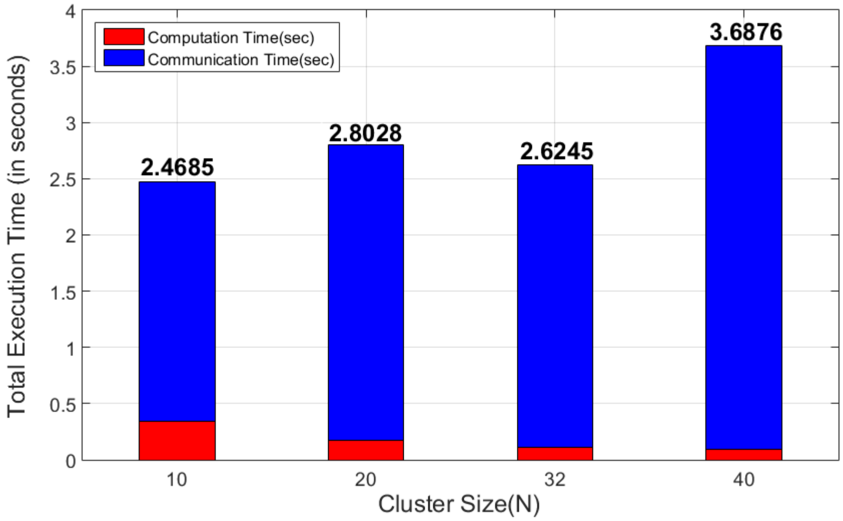

Figure 3.10: DA algorithm in AWS platform: total execution time i.e., sum of computation time and communication time vs cluster size.

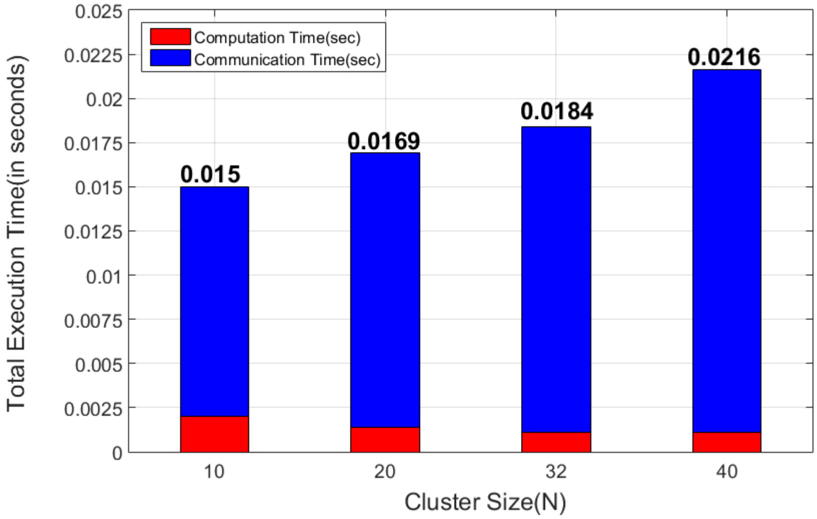

Figure 3.11: LSDA Algorithm in AWS platform: total execution time i.e., sum of computation time and communication time vs cluster size.

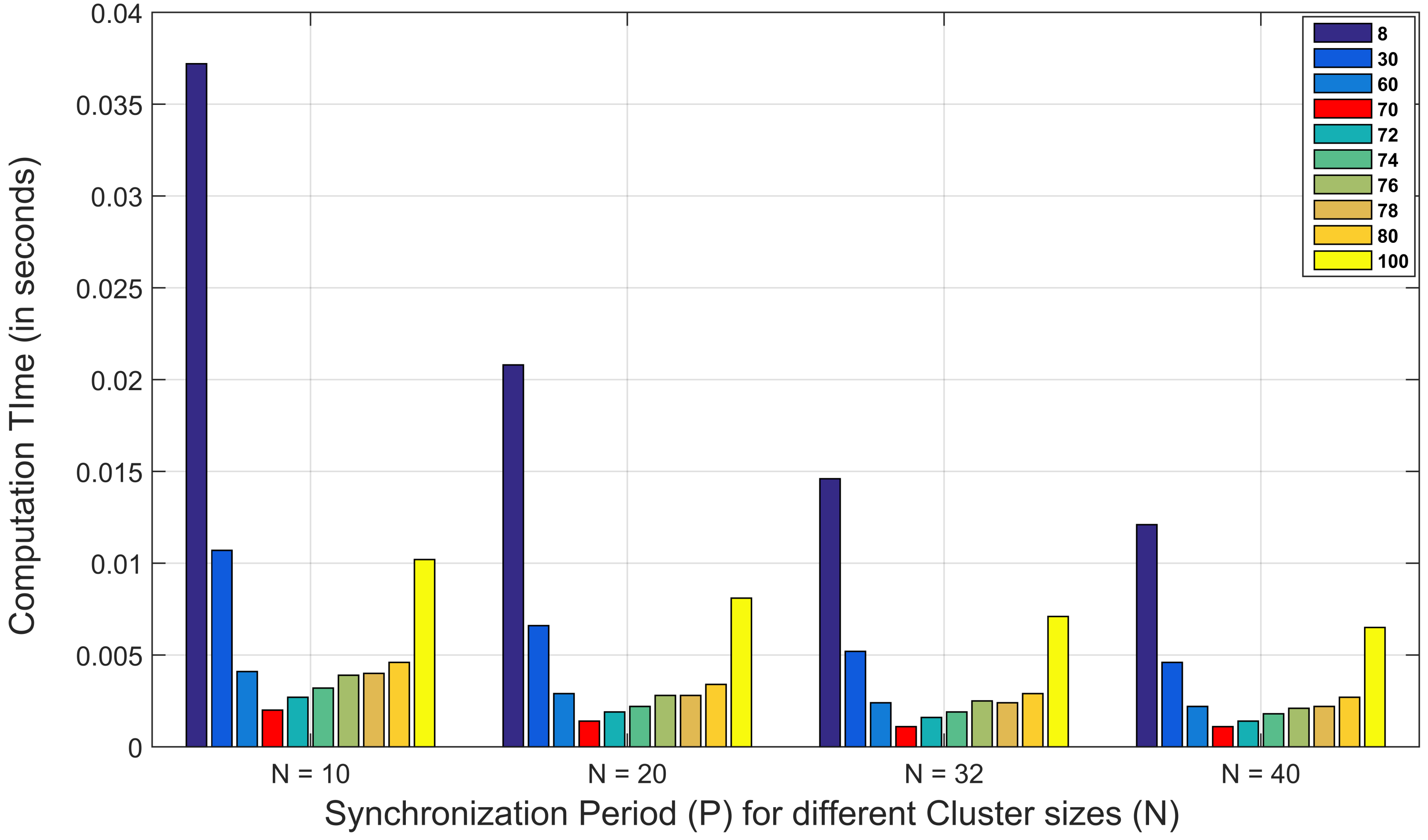

Figure 3.12: Variation of computation time with cluster size (N) and synchronization period (P)

Variation with Cluster Size: Given that the optimal synchronization period P is 70, the computation time was analyzed with changing cluster sizes at this value of P. As the cluster size increases, the data load on each cluster minimizes and hence the computation time decreases as cluster size increases. This is a trend which can be seen both in LSDA and DA algorithm as shown in 3.10 and 3.11. When compared with the computation time of DA algorithm, we see a significant decrease. This decrease is mainly attributed to reduced number of iterations. This trend is observed irrespective of the platform used as evident from figures 3.13 and 3.14.

3.3.3.3 Communication Time

The time spent in the inter-node communication, i.e. synchronization in case of LSDA and DA algorithms, is defined as Communication Time.

Figure 3.11 shows communication time of LSDA algorithm with increasing cluster size N. However, this trend is not strictly followed due to other traffic and noise on the network which is non-deterministic. In HPC platform, we see that the communication time is closer to the expected behavior yet not strictly followed. Figures 3.13 and 3.14 show the communication time in HPC cluster platform.

3.3.3.4 Speedup

Figure 3.15 shows the speedup of execution time which is the sum of computation time and communication time achieved by LSDA algorithm over the DA algorithm. We see that a speedup of around 160×(± 12.5) is obtained in case of AWS platform. It should be noted that the speedup is almost constant. Such flattened speedup is the theoretically expected trend as observed in section 3.2.5. The flattening of speedup shows that the cluster sizes selected fell in the communication dominant region. In the case of HPC cluster platform, we observe a stark difference as the computation time dominates over the communication time. This domination of the computation

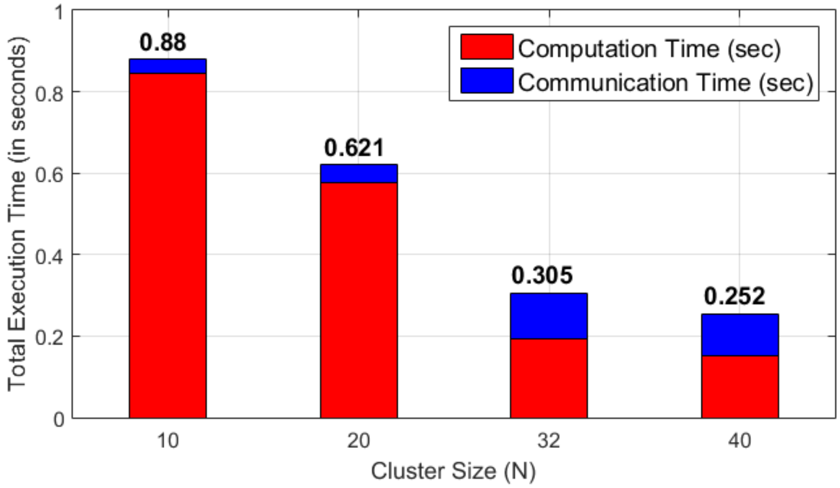

Figure 3.13: DA algorithm in HPC platform: total execution time i.e., sum of computation time and communication time vs cluster size.

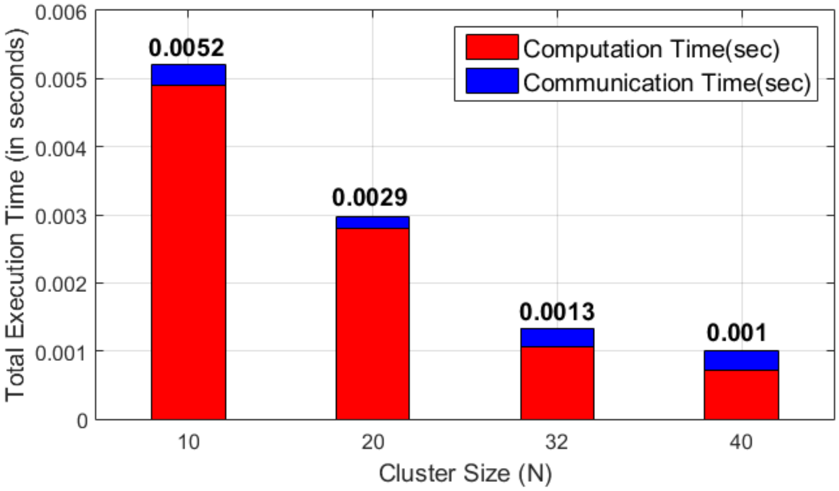

Figure 3.14: LSDA Algorithm in HPC platform: total execution time i.e., sum of computation time and communication time vs cluster size.

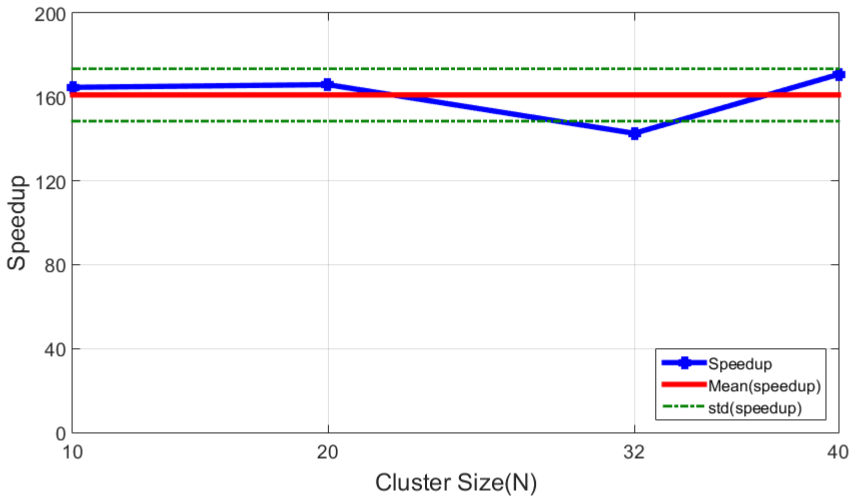

Figure 3.15: Speedup(AWS) of overall execution time of LSDA with respect to DA algorithm.

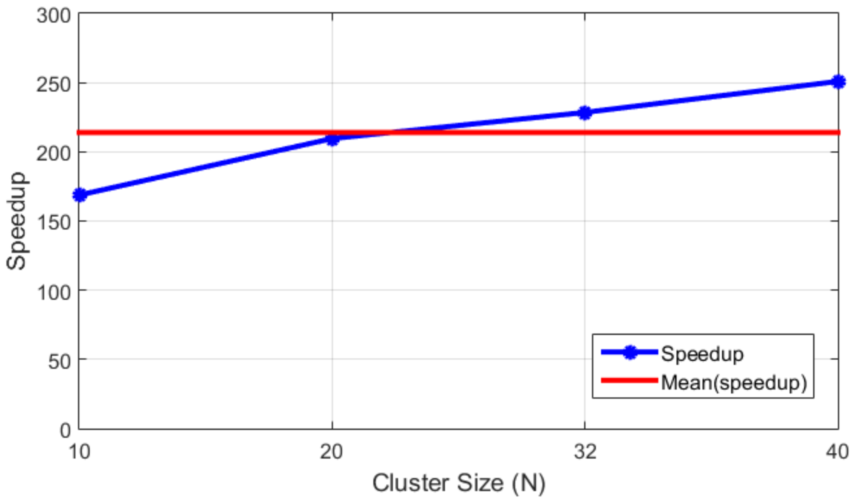

Figure 3.16: Speedup(HPC cluster) of overall execution time of LSDA with respect to DA algorithm.

time over the communication time is mainly due to the high performance computing flavored InfiniBand(IB) interconnect used in the HPC platform for communication. Figure 3.16 shows the speedup of LSDA algorithm over DA algorithm solving the same problem as that of AWS cluster in an HPC cluster platform. The speedup tends to increase as we fall in the computation dominant region, evident from figure 3.14. This validates our theoretical results on speedup in section 3.2.5.

3.4 Conclusion

In this chapter, we introduced a lazy synchronization technique to solve quadratic programming problem using dual ascent algorithm. Analytical proofs show that this approach ensures faster convergence of the optimization problem than conventional DA algorithm. We both analytically and empirically prove that communication time is greatly reduced using LSDA algorithm. For the experimental data used, we show that empirically communication time is reduced by almost 99% as we synchronize only three times in case of LSDA algorithm rather than 868 times in case of DA algorithm. With the optimal synchronization period, we also reduce the total number of iteration to converge to an ϵ-accurate solution. In addition, we also prove both analytically and empirically that irrespective of the cluster size we would still attain a constant speedup.

4. QR - SVM

4.1 Introduction - Support Vector Machines

Support Vector Machines (SVMs) is a popular machine learning classification technique first introduced by Boser, Guyon and Vapnik [7]. The primary objective of an SVM model is to maximize the separation margin between the data points of different classes thereby forming the best possible separation hyperplane. Hence, d-dimensional data points are separated by a $(d-1)$-dimensional hyperplane. This method is a form of supervised learning where the class label of each of the data points is available before the training phase begins. The original SVM was designed for a binary classification problem. To solve a multivariate (more than 2 classes) classification problem two commonly used strategies are one-versus-one and one-versus-all [6, 17] techniques where multiple SVM models are trained either sequentially or in parallel.

Though SVM is commonly used to solve classification problems, the idea, i.e. optimal margin, is not restricted to supervised classifiers alone. There are several variations like support vector clustering [4], support vector regression [16] and one-class SVM [40] which solve clustering, regression modelling and anomaly detection problems respectively.

Figure 4.1 illustrate a simple SVM model of 2-dimensional data where H1 denotes the 1-dimensional maximal margin hyperplane classifying the data. In figure 4.2, hyperplanes H2 and H3 also classify the data; however, they are not the best possible solution with least generalization error. Generalization error in a supervised learning context indicates the accuracy of the model in predicting the results of unseen data i.e., out-of-sample error [1].

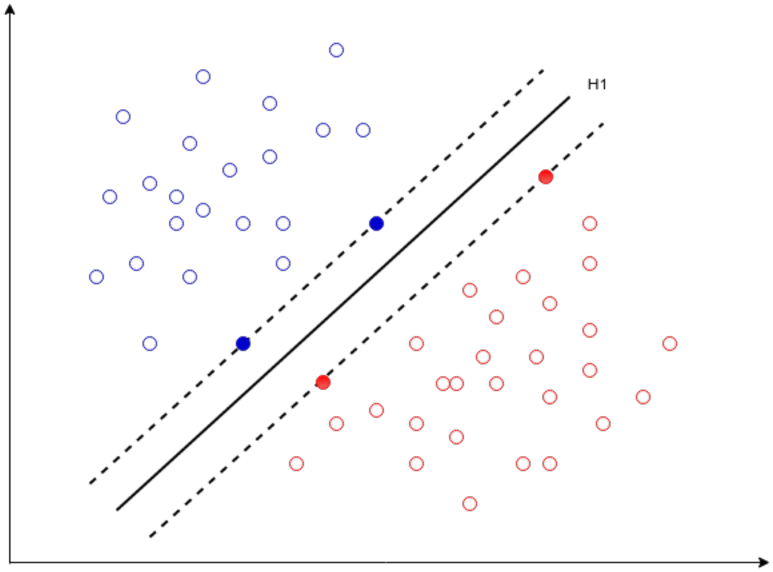

Figure 4.1: Illustration of SVM classifier.

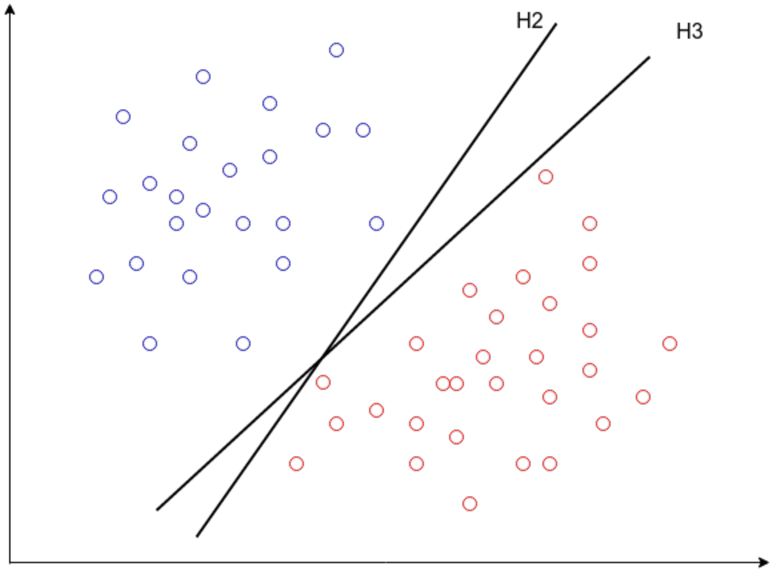

Figure 4.2: Illustration of generic classifiers.

The advantages of using SVM as a classification technique apart from the mathematical simplicity are:

1. Low generalization error is observed as the model tries to find the maximum separation margin. This also accounts for robust results as compared against other classifiers.
2. SVM models can easily handle non-linear data spread using the kernel trick.
3. The support vectors (points which lie on the boundary and determine the hyperplane) can be used to differentiate between "interesting" and "non-interesting" points. These support vectors essentially carry most of the information in the dataset and can be used in incremental learning [41].

4.1.1 Mathematical Formulation

Given n training data points in d-dimensional space belonging to two classes $\{-1, +1\}$ i.e., given data points $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \dots (\mathbf{x}_i, y_i) \dots (\mathbf{x}_n, y_n)$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.The objective is to find a vector $\mathbf{w}$ and a bias b such that

$$\mathbf{w}^T x + b = 0 \tag{4.1}$$

is the maximal margin hyperplane.

The values $\mathbf{w}$ and b are normalized such that the support vectors (data points on the boundary) satisfy the equation

$$|\mathbf{w}^T x + b| = 1 \tag{4.2}$$

The modulus in equation (4.2) is necessary so that the support vectors from both the classes are considered. Given this normalization, the margin between the hyperplane in (4.1) and support vectors reduces to $\frac{1}{||\mathbf{w}||_2^2}$. Hence, the SVM can now be formulated as a maximization problem:

$$\max_{w} \frac{1}{||\mathbf{w}||_2^2}$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } n = 1, 2, ...n$$

The above quadratic optimization can be re-written in a more standard minimization problem as

$$\min_{w} \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{4.3}$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } n = 1, 2, ...n$$

The bias b which is typically associated with the hyperplane can be induced into the vector $\mathbf{w}$ by adding an additional dimensionality to the input data point $\mathbf{x}_i$

$$\mathbf{x}_i \leftarrow [\mathbf{x}_i; 1] \implies w \leftarrow [w; b]$$

Eq (4.3) is often referred to as *hard margin SVM* where the data points are linearly separable in the d-dimensional space. However, this is not the case in most of the real-world classification problems. Hence, *hinge loss* is introduced into the objective function to penalize the misclassifications. Such objective functions are called soft margin SVM. In the case of hard margin SVM (eq (4.3)), there are no misclassifications to begin with, and hence, no loss function is needed.

A soft margin SVM problem is now an unconstrained minimization problem given as

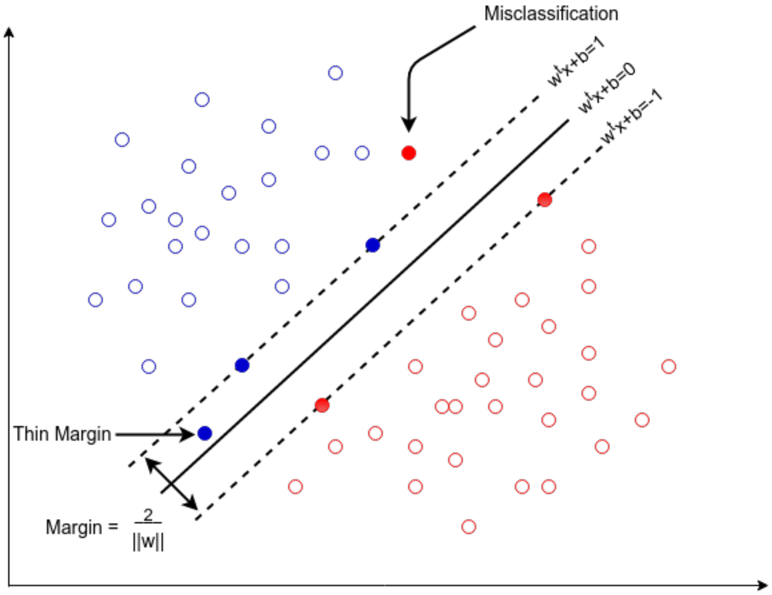

Figure 4.3: Illustration of soft margin SVM.

$$\min_{w} \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n} \xi_i(\mathbf{w};\mathbf{x}_i;y_i) \tag{4.4}$$

where $\xi_i(\mathbf{w};\mathbf{x}_i;y_i)$ represents the loss function associated with the optimization problem and C represents the penalty parameter i.e., how much is a data point penalized for a classification error. Hence, when the penalty parameter C is increased the classifier becomes more strict. Figure 4.3 illustrates the use of soft margin SVM. For a low penalty parameter C, the classifier considers the best margin solution even though there are some data points within the margin. If the penalty parameter was high, the SVM model would not have allowed such points within the margin.

The common loss functions(ξ_i) used in SVM training are l_1-loss(L1-SVM) or l_2-loss(L2-SVM).

l_1**-loss**:

$$max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

l_2**-loss**:

$$max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)^2$$

The dual form of optimization problem in (4.4) is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\hat{Z}\boldsymbol{\alpha} + e^T\boldsymbol{\alpha} \\ \text{subject to } & L \leq \alpha_i \leq U \end{aligned} \tag{4.5}$$

where $\hat{Z} = Z + D$, $\mathbf{e}$ is a vector with of negative ones, L is the lower bound of the dual variable and U is the upper bound. $Z_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ where k represents the kernel function. For linear-SVM, the kernel function k is the dot product of the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Diagonal matrix D, lower bound L and upper bound U are dependent of the type of loss function associated with the SVM problem. In case of L1-SVM, $D = \mathbf{0}_n$, $L = 0$ and $U = C$ while for L2-SVM, $D = diag(1/2C)_n$, $L = 0$ and $U = \infty$.

The dual form in (4.5) might not be as intuitive as the primal SVM. However, there are several advantages of solving the dual form of SVM over the primal SVM. The dual variable $\boldsymbol{\alpha}$ denotes the hidden weight of each data point in determining the classifier. If α_i is zero, it indicates that the data point $\mathbf{x}_i$ is towards the interior and away from the margin and if α_i is greater than zero, it shows that $\mathbf{x}_i$ is a support vector and is either on or close to the margin. As explained earlier, these support vectors are useful in decomposition algorithms [43, 24], incremental learning [38, 41].

Another reason for solving the dual problem is to leverage the kernel trick for non-linear classifiers. SVM is essentially a linear classifier. If the data is not linearly separable, then its mapped to a higher dimensional space where it can be linearly separable and then an SVM model is trained in that high dimensional space. It is not easy to transform data to higher dimensional space and sometimes there can even be infinite number of dimensions. Using the kernel trick, a kernel function k can be found, such that

$$k(\mathbf{x}_i, \mathbf{y}_i) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$$

where $\langle \cdot, \cdot \rangle$ represents the inner product and φ is the transformation to higher dimensional space $\varphi : \mathbb{R}^d \to \mathbb{R}^D$. Hence, even without explicit knowledge about φ a non-linear classifier can still be trained by using the function k on the original set of data points $\mathbf{X}$ in the d dimensional space. Though kernel trick is very useful, for large scale datasets calculating the SVM problem turns computationally infeasible. Hence, we prefer either solving a linear SVM problem in such cases.

4.2 Linear SVM

Linear SVM involves finding the maximal margin hyperplane in the original input data space. The kernel function in the dual form of linear SVM in (4.5) takes the form

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

We can rewrite equation (4.5) specific to linear SVM for $l1$-loss and $l2$-loss as

l_1-**loss**:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T \Big(diag(\mathbf{y}) X X^T diag(\mathbf{y})^T \Big) \boldsymbol{\alpha} + e^T \boldsymbol{\alpha}$$

$$\text{subject to } 0 \leq \alpha_i \leq C$$

l_2**-loss**:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T\Big(diag(\mathbf{y})XX^Tdiag(\mathbf{y})^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}I_n\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha}$$

$$\text{subject to } 0 \le \alpha_i \le \infty$$

where $X = [\mathbf{x}_1^T;\mathbf{x}_2^T;\mathbf{x}_3^T...\mathbf{x}_n^T]$, $\mathbf{y} = \{y_i \in \{-1,1\} | i = 1...n\}$ and C is the penalty parameter. Substituting $\hat{X} = diag(\mathbf{y})X$,

l_1**-loss**:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T\Big(\hat{X}\hat{X}^T\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha} \tag{4.6}$$

$$\text{subject to } \begin{bmatrix} -I_n \\ I_n \end{bmatrix} \boldsymbol{\alpha} \le C \begin{bmatrix} \mathbf{0}_n \\ \mathbf{1}_n \end{bmatrix}$$

l_2**-loss**:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T\Big(\hat{X}\hat{X}^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}I_n\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha} \tag{4.7}$$

$$\text{subject to } - I_n\boldsymbol{\alpha} \le \mathbf{0}_n$$

L2-SVM provides a simpler constraint formulation which specifies each α_i corresponding to each data x_i must be non-negative. Moreover, the optimization problem in L2-SVM is strongly convex and smooth when compared to L1-SVM, which makes it easier to solve. For these specific reasons we elaborate proposed QR-SVM mainly for L2-SVM, however, the similar methodology can be extended to L1-SVM as well.

4.2.1 Challenges

Matrices X and $\hat{X}$ are based on the input of SVM classification problem. The dimension of these matrices is $(n \times d)$ where n is the number of input instances and d is the dimensionality of each data point. In a majority of SVM classification problems, the number of instances is much larger than the dimensionality of a data point, i.e. $d \ll n$. The $\hat{X}\hat{X}^T$ in equations (4.6) and (4.7) is a dense $(n \times n)$ matrix which is tightly coupled and hence can't be decomposed into independent sub-problems of smaller size trivially. Working on the whole $(n \times n)$ data matrix not only leads to high computational complexities but also loading this $O(n^2)$ matrix to memory is practically infeasible. Also, the quadratic coefficient matrix $\hat{X}\hat{X}^T$ is also rank deficient and hence non-invertible. This makes the dual ascent algorithm highly unstable. To address the above issues, we propose QR-SVM framework comprising of 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 relatively faster than the state of the art solver.

4.3 Proposed QR-SVM

Some basic preliminary concepts of QR decomposition and various algorithms used to for this decomposition are detailed in appendix A. In this section we first present the motivation for using QR decomposition in Linear SVM, followed by the formulation and benefits of proposed QR-SVM framework for L2-SVM.

4.3.1 Motivation

In a majority of SVM classification problems, the number of instances is much larger than the dimensionality of a data point, i.e. $d \ll n$. For such SVM problems, the input data set X and the corresponding $\hat{X}$ are tall skinny matrices. On applying

QR decomposition on $\hat{X}$ we get

$$\hat{X}_{n\times d} = Q_{n\times n}R_{n\times d}$$

The R matrix from QR decomposition, is an upper triangular matrix of size $n\times d$. The maximum number of possible non-zero elements in this R matrix is $d(d+1)/2$ making it highly sparse when compared to the original matrix $\hat{X}$ with a maximum of nd non-zero elements as $d \ll n$. So, if we transform the vector $\boldsymbol{\alpha}$ (equations (4.6) and (4.7)) from the original vector space of the SVM problem to a vector space with basis defined by the orthogonal matrix Q, i.e. $\hat{\boldsymbol{\alpha}} = Q^T\boldsymbol{\alpha}$, we will be dealing with a much smaller data matrix in terms of R. This transformation of vector spaces will thereby sparsify the computations involved in solving the transformed SVM problem. As the data size reduction is dependent on the ratio of $d : n$, tall-skinny SVM problems yield high computation benefits from this QR decomposition approach.

4.3.2 QR-SVM Formulation for L2-SVM

QR decomposition on a TS-dense $\hat{X}$ factorizes it into two matrices, namely, Q and R i.e., $\hat{X} = QR$. Here, Q is an orthogonal matrix of size $n \times n$ and R is a upper triangular matrix of size $n \times d$. The quadratic programming (QP) problem in equation (4.7) now becomes

$$\frac{1}{2}\boldsymbol{\alpha}^T\Big(\hat{X}\hat{X}^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}I_n\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha} \tag{4.8}$$

$$= \frac{1}{2}\boldsymbol{\alpha}^T\Big(QRR^TQ^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}I_n\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha}$$

$$= \frac{1}{2}(\boldsymbol{\alpha}^TQ)\Big(RR^T\Big)(Q^T\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}I_n\Big)\boldsymbol{\alpha} + e^T\boldsymbol{\alpha}$$

On substituting $Q^T\boldsymbol{\alpha} = \hat{\boldsymbol{\alpha}}$, cost function of L2-SVM QP problem formulates to

$$\frac{1}{2}\hat{\boldsymbol{\alpha}}^T\Big(RR^T\Big)\hat{\boldsymbol{\alpha}} + \frac{1}{2}\hat{\boldsymbol{\alpha}}^T Q^T\Big(\frac{1}{2C}I_n\Big)Q\hat{\boldsymbol{\alpha}} + \left(Q^T e\right)^T\hat{\boldsymbol{\alpha}} \tag{4.9}$$

$$\text{subject to} \qquad -Q\hat{\boldsymbol{\alpha}} \leq \mathbf{0}_n$$

Defining $\hat{e} = Q^T e$ and using $Q^T Q = I_n$,

$$\frac{1}{2}\hat{\boldsymbol{\alpha}}^T\Big(RR^T + \frac{1}{2C}I_n\Big)\hat{\boldsymbol{\alpha}} + \left(\hat{e}\right)^T\hat{\boldsymbol{\alpha}} \tag{4.10}$$

$$\text{subject to} \qquad -Q\hat{\boldsymbol{\alpha}} \leq \mathbf{0}_n$$

The QP problem in (4.10) can be interpreted as a combination of two QP problems where the first QP problem deals with the first d components of the vector $\hat{\boldsymbol{\alpha}}$ and the second QP problem targets the next $n - d$ components of the vector $\hat{\boldsymbol{\alpha}}$.

4.3.3 Benefits of QR-SVM

From (4.10), it can be observed that by transforming the basis of $\boldsymbol{\alpha}$ to $\boldsymbol{\alpha}_hat$, we were able to move the problem to a vector space where the

In equation (4.10), RR^T is a symmetric sparse matrix of size n where the first $d \times d$ submatrix is dense while rest of the elements in the matrix are all zeros. In other words, coefficient matrix of the quadratic term in equation (4.10) is a block diagonal matrix comprising of two diagonal blocks: 1) a $d \times d$ symmetric and dense submatrix, $(RR^T)_d$+$(1/2C)I_d$ and 2) a diagonal submatrix $(1/2C)I_{n-d}$. The benefits of the proposed QR-SVM formulation are listed below.

1. **Sparsity**: We have transformed a dense $n \times n$ quadratic coefficient matrix $\hat{X}\hat{X}^T + (1/2C)I_n$ in equation (4.8) to a sparse matrix as in equation (4.10) which consists of a small dense $d \times d$ block, as illustrated in Figure 4.4. The

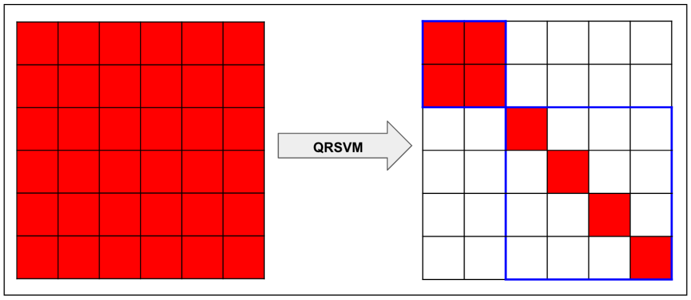

Figure 4.4: QR-SVM technique on L2-SVM transforms a 6×6 dense and non-separable coefficient matrix into a sparse block diagonal matrix, where, the first 2×2 block is full rank and the second 4×4 block is a diagonal submatrix. Dense regions are colored. The two blocks in the transformed matrix on the right are outlined in blue. Here, $n = 6$ and $d = 2$.

sparse coefficient matrix in the proposed QR-SVM consumes $(d^2 + 1)/(n^2)$ fraction of the memory assigned to the original dense coefficient matrix.

2. **Separability**: We have also rendered the aforementioned non-separable quadratic coefficient matrix into a block separable form by using the proposed QR-SVM formulation. One can exploit this separability to independently solve the two sub-problems in parallel at each iteration dual ascent algorithm.

3. **Invertibility**: On applying QR-SVM, the *low-rank* quadratic coeffcient matrix becomes block-separable where the two sub-blocks are now invertible. The first block $(RR^T)_d + (1/2C)I_d$ is *full-rank* in d and the second block $(1/2C)I_{n-d}$ 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 (4.12).

The proposed QR-SVM formulation can be implemented without any additional memory compared to the original problem in equation (4.8). The given input data $\hat{X}$ can be replaced by the set of $d-$ Householder reflectors (i.e. Q) and the matrix R, which together occupy the same $nd-$ memory space as $\hat{X}$. In addition, it is observed that one can recover the values of $\boldsymbol{\alpha}$ (to identify corresponding support vectors) from $\hat{\boldsymbol{\alpha}}$ by simply pre-multiplying $\hat{\boldsymbol{\alpha}}$ with series of $d-$ Householder reflectors with a computational cost of $O(nd)$. 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, one can efficiently compute the normal $\mathbf{w}$ to the linear classifier by directly pre-multiplying R^T to $\hat{\boldsymbol{\alpha}}$. It is worth noticing that calculation of $R^T\hat{\boldsymbol{\alpha}}$ can be simplified to $R_d^T\hat{\boldsymbol{\alpha}}_{\boldsymbol{d}}$ i.e. $O(d^2)$, given the special structure of matrix R. The QR-SVM process workflow is illustrated in Figure 4.5.

4.4 Optimization Using Dual Ascent

As detailed in chapter 3, Dual Ascent (DA) algorithm can be used to solve Quadratic Programming optimization problems if the quadratic coefficient matrix is invertible. Both the standard SVM problem in (4.8) and QR-SVM in (4.10) are QP problems. However, it is not possible to solve the standard SVM problem ((4.8)) directly as quadratic coefficient matrix $\hat{X}\hat{X}^T + (1/2C)I_n$ is rank deficient and not invertible. In case of QR-SVM, the quadratic coefficient matrix in QR-SVM is invertible and hence can be solved by LSDA algorithm. The Lagrangian $\boldsymbol{\mathcal{L}}$ of QP problem in equation (4.10) is written as follows

$$\boldsymbol{\mathcal{L}}(\hat{\alpha}, \beta) = \frac{1}{2}\hat{\alpha}^T\Big(RR^T + \frac{1}{2C}I_n\Big)\hat{\alpha} + (\hat{e})^T\hat{\alpha} + \beta^T(-Q\hat{\alpha}) \tag{4.11}$$

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

Dual ascent update steps for QR-SVM are

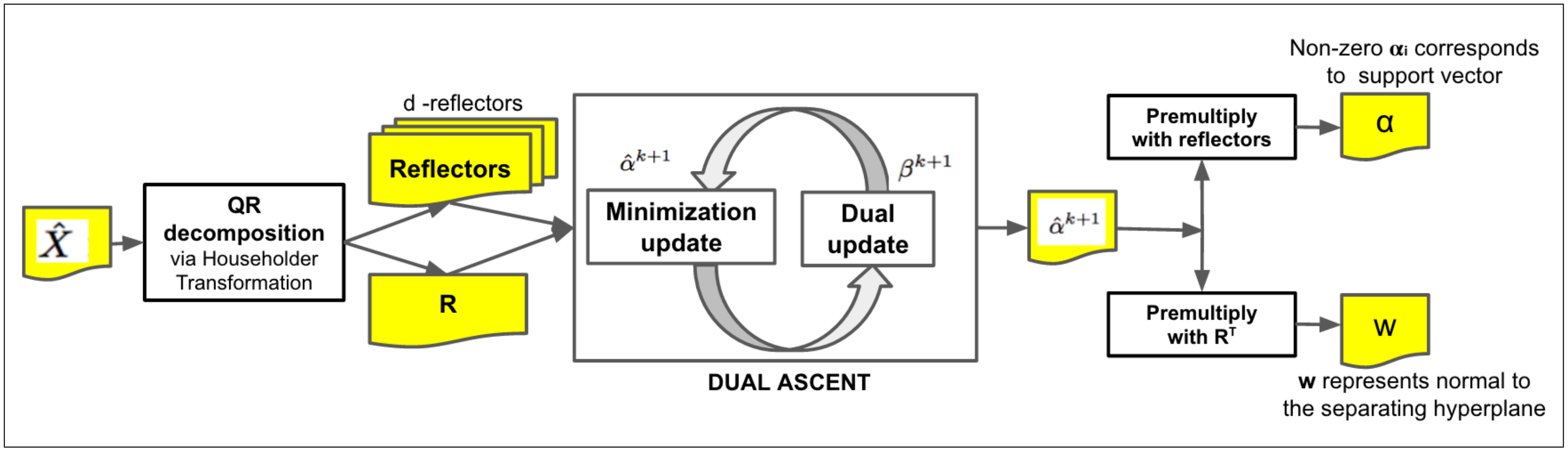
$\hat{X}$
QR decomposition via Householder Transformation
d -reflectors
Reflectors
R
$\hat{\alpha}^{k+1}$
β^{k+1}
Minimization update
Dual update
DUAL ASCENT
$\hat{\alpha}^{k+1}$
Premultiply with reflectors
Premultiply with R^T
Non-zero αi corresponds to support vector
α
w
w represents normal to the separating hyperplane

Figure 4.5: QR-SVM framework comprises of two main stages, namely, 1. *QR decomposition* of the original input matrix $\hat{X}$ into Householder reflectors and a matrix R, and 2. *Dual Ascent* method to solve the QR-SVM problem for obtaining the normal w to the hyperplane and identifying set of support vectors.

Step 1: Minimization of Lagrangian

$$\hat{\alpha}^{k+1} = \arg\min_{\hat{\alpha}} \mathcal{L}(\hat{\alpha}, \beta^k) = -\Big(RR^T + \frac{1}{2C} \times I_n\Big)^{-1}(-Q^T\beta^k + \hat{e}) \tag{4.12}$$

Step 2: Dual variable update

$$\beta^{k+1} = \beta^k + \eta(-Q\hat{\alpha}^{k+1}) \tag{4.13}$$

where $\eta > 0$ is the step size and the superscript $k = 0, 1, 2...$ is the iteration counter. β^0 is initialized to $\mathbf{0}_n$. To satisfy the inequality constraint on each of the dual variable β_i being non-negative , β_i is replaced with $\max(0, \beta_i)$ during every iteration.

While the above update equation (4.12) appear as a single operation to calculate $\hat{\boldsymbol{\alpha}}$, the actual computation can be distributed among available computation nodes and calculated in parallel independent of each other. Given the block diagonal structure of matrix $\Big(RR^T + \frac{1}{2C} \times I_n\Big)$, the update equation (4.12) can be split into two major subproblem where subproblem 1 computes the first d elements of $\hat{\boldsymbol{\alpha}}$ and subproblem 2 computes the rest of the $n-d$ elements of $\hat{\boldsymbol{\alpha}}$. Hence, the update equations of the dual ascent algorithm now become

Step 1: Minimization of Lagrangian

$$\begin{aligned} \hat{\alpha}_1^{k+1} &= -\Big(RR_d^T + \frac{1}{2C} \times I_d\Big)^{-1}(-Q_1^T\beta^k + \hat{e}_d) \\ \hat{\alpha}_2^{k+1} &= -2C(-Q_2^T\beta^k + \hat{e}_{n-d}) \end{aligned} \tag{4.14}$$

Step 2: Dual variable update

$$\beta^{k+1} = \beta^k + \eta(-Q\hat{\alpha}^{k+1}) \tag{4.15}$$

where $\hat{\boldsymbol{\alpha}} = [\hat{\boldsymbol{\alpha}}_{\mathbf{1}}; \hat{\boldsymbol{\alpha}}_{\mathbf{1}}]$ and $Q = [Q_1; Q_2]$; $Q_1 \in \mathbb{R}^{n\times d}$ and $Q_2 \in \mathbb{R}^{n\times(n-d)}$. The update equation of $\hat{\boldsymbol{\alpha}}_{\mathbf{2}}$ in is a simple vector arithematic with can be further distributed trivially.

4.4.1 Optimal Step Size of Optimization Algorithm

The update steps of the dual ascent algorithm in (4.12) and (4.13) can be optimized with respect to step size based on the LSDA formulation based on derivations in Chapter 3, section 3.2.6. The below lemma and corollary will provide the derivation for the optimal step size η^* used in (4.13).

Lemma 2 (Scaling factor for optimal step size) *To ensure the minimum number of iterations involving sequential dual variable update step, the scaling factor $P^\star$ for optimal step size is obtained by*

$$P^\star = \max \quad \arg\min_{P\in\mathbb{N}} \max\{|1-\lambda_{min}(M)P|, |1-\lambda_{max}(M)P|\} \tag{4.16}$$

where, $M := \eta\Big(RR^T + \frac{1}{2C}I_n\Big)^{-1}$, $\eta > 0$ 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 (4.16), we get the following result.

Corollary 1 *For any $\eta > 0$, the optimal step size $\eta^\star$ can be computed using*

$$\eta^\star = P^\star\eta, \quad P^\star \in \mathbb{N} \tag{4.17}$$

where,

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

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

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 $A(\bar{\lambda}^{-1}, \lambda_{max}\bar{\lambda}^{-1} - 1)$.

$$\max\{f_1(P), f_2(P)\} = \begin{cases} 1 - \lambda_{min}(M)P & \text{if } 0 < P \leq A_x \\ \lambda_{max}(M)P - 1 & \text{if } P > A_x \end{cases} \text{,where,} \quad A_x = \bar{\lambda}^{-1}$$

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

$$\arg\min\max\{f_1(P), f_2(P)\} = A_x.$$

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

This value which is lesser than A_x ensures that the scaling factor is optimum, $P^\star$, and will result in optimal step size, $\eta^\star$, leading to stability and convergence of dual ascent method. It is worth noting that given the special structure of M comprising of 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, $\bar{\lambda}^{-1} \approx 1/(\eta C)$ can be used as a good approximation for faster convergence of the dual ascent method.

4.5 Complexity Analysis

The cost of a single iteration of QR-SVM is the combined computation cost of the two update steps defined in equations (4.12) and (4.13). Premultiplying Q (or Q^T) to a vector v by using Householder reflectors requires $O(nd)$ operations, where n is the size of vector v and d is the number of Householder reflectors [29]. The cost of computing each of $(-Q^T\beta^k)$ in equation (4.12) and $(-Q\hat{\alpha}^{k+1})$ in equation (4.13) is $O(nd)$. Given the block diagonal structure of $RR^T + \frac{1}{2C}I_n$, the computation in equation (4.12) can be split into following:

Subproblem 1: The first d components of $\hat{\alpha}^{k+1}$ are computed by solving a system with the $d \times d$ coefficient matrix $RR_d^T + \frac{1}{2C}I_d$. By computing and storing Cholesky factors of this matrix before starting the iterations, the system can be solved in $O(d^2)$ 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 $O(d^3)$.

Subproblem 2: Calculation of the remaining $(n-d)$ components of $\hat{\alpha}^{k+1}$ requires $O(n-d) \approx O(n)$ operations since $d \ll n$.

The overall computation cost of update equation (4.12) is $O(nd + d^2)$. Also, the computation cost of equation (4.13) is trivially $O(nd)$. Combining the above two computation cost for equations (4.12) and (4.13) and including the initial QR decomposition cost, we see that QR-SVM using dual ascent method requires $O(nd^2 + knd)$ operations where k is the number of iterations. The trend is empirically validate in section 4.6.1.2.

4.6 Experiments

The experiments of QR-SVM were performed on a uniprocessor system with LSDA algorithm optimized to work with optimal step size as discussed in sections

3.2.6 and 4.4.1. Given the block diagonal structure of the quadratic coefficient matrix in QR-SVM formulation, the implementation of LSDA algorithm in QR-SVM framework can be extended to multiprocessor systems for additional computational benefits.

The experiments were conducted in three phases. The first phase involved testing the performance of the QR-SVM on synthetic datasets of varying sizes, i.e. to test the *scalability of QR-SVM with data sizes.* The second and third phases involved comparision of QR-SVM with the state of the art solver, LIBLINEAR[20]. In the second phase, we compared the convergence trends of the two algorithms while in the third phase, the performance and accuracies were compared. We also tried to compare with other solvers like SVM^{light}[28] and LIBSVM[10]. However, the time taken to converge for these libraries was too long. Also, it must be pointed out that the LIBLINEAR algorithm was set to terminate at a preset maximum number of iterations. The result at the end of this maximum iteration was used to compare with QR-SVM.

QR-SVM was implemented in C/C++ to ensure a fair comparison with the implementation of LIBLINEAR which is also implemented in C/C++ and is available as an open source library. The linear algebra computations in QR-SVM were handled using armadillo library[39]. Both the programs were run on a single core of an 8-core Intel Nehalem processor with 12GB memory.

4.6.1 Results and Discussion

4.6.1.1 Scalability of QR-SVM with data sizes

Scalability with n: The effect on training time of QR-SVM was analyzed using a synthetic dataset with a fixed dimensionality, d=18 and increasing number of instances, n. It is observed that QR-SVM training time increases linearly with the number of data points as shown in Figure 4.6. As a result, our proposed framework is capable of handling growing data sizes efficiently making it suitable for big data applications.

Scalability with d: With increasing dimensionality or the number of features in a data point, the training time of QR-SVM is observed to grow quadratically as shown in Figure 4.7. This trend validates our discussion in section 4.5. However, as we are addressing the low dimensional problem space through the proposed QR-SVM framework, the benefits from the fast overall convergence rate of the algorithm outweighs the quadratic cost. Using the same benchmarks, we compared the performance of QR-SVM and LIBLINEAR and observed that for a dataset with 100k data points, LIBLINEAR tends to outperform QR-SVM when the number of dimensions increases beyond 700. This trend can be observed in Figure 4.8. It should be noted that this dimensionality threshold (d_t) of 700 is not fixed and is dependent on the number of instances. By increasing the number of instances, the value of d_t tends to increase.

4.6.1.2 Comparison with LIBLINEAR - Convergence of QR-SVM

As shown in Figure 4.9, it was observed that for most of the problems, we can reach a reasonable value of the objective function within 10 iterations of dual ascent method. Figure 4.10 compares the convergence rates of QR-SVM with LIBLINEAR. QR-SVM was able to converge to the optimal cost of the objective function for HIGGS dataset[34] within 80 iterations while LIBLINEAR couldn't converge even after 1500 iterations.

4.6.1.3 Comparison with LIBLINEAR - Performance and accuracy of QR-SVM

A few benchmark datasets were picked up from standard SVM repository[10, 34]. We selected datasets for binary classification problems which were low in dimension-

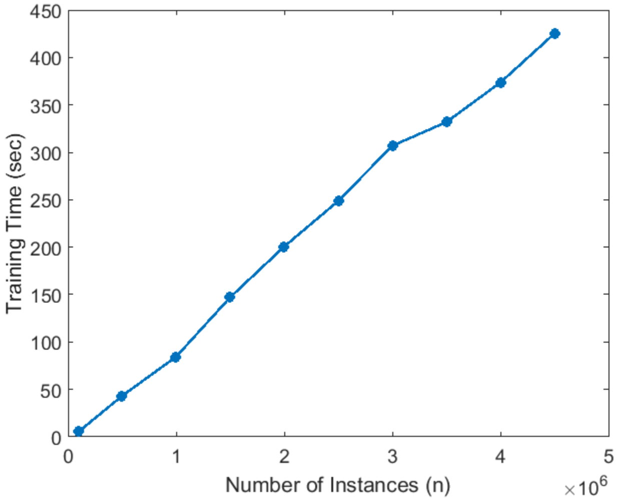

Figure 4.6: QR-SVM scales linearly with number of instances, n in the dataset. A synthetic dataset with fixed dimensionality, d=18 and increasing n was used to test scalability with number of instances.

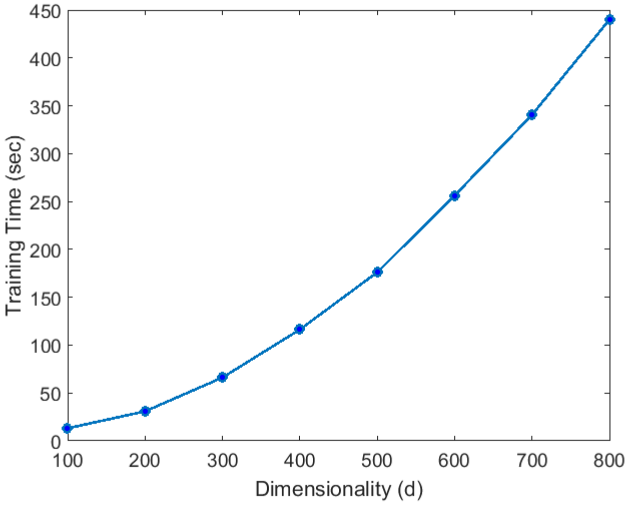

Figure 4.7: QR-SVM scales quadratically with dimensionality, d of the dataset. A synthetic dataset with fixed number of instances, n=100,000 and increasing d was used to test scalability with dimensionality.

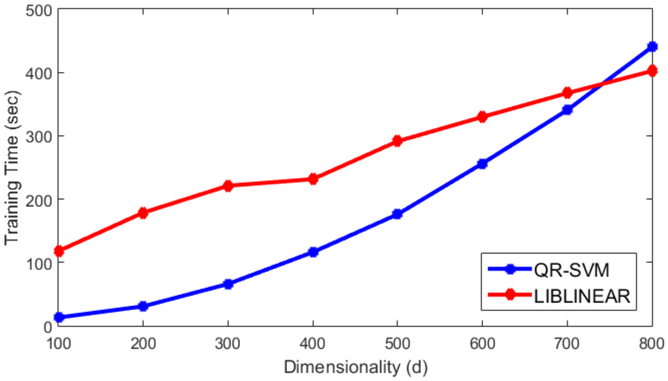

Figure 4.8: Comparision of performance of QR-SVM and LIBLINEAR with dimensionality. A synthetic dataset with number of instances fixed at 100,000 and varying dimensionality was used to make this comparision. QR-SVM outperforms LIBLINEAR till a dimensionality of 700.

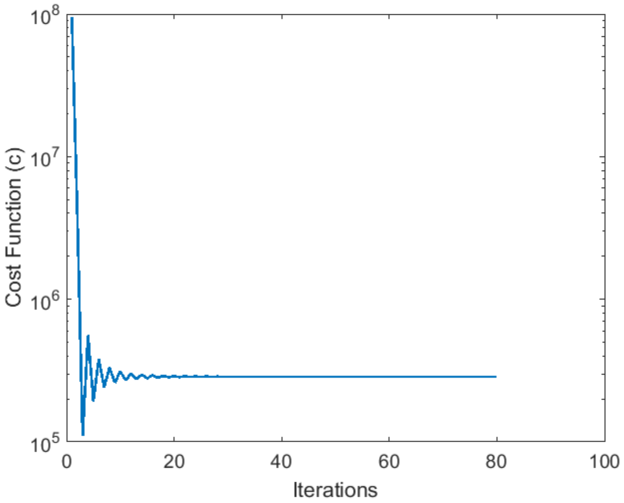

Figure 4.9: Convergence of QR-SVM for HIGGS dataset: using QR-SVM we converge to a reasonable value of the optimal cost within 20 iterations.

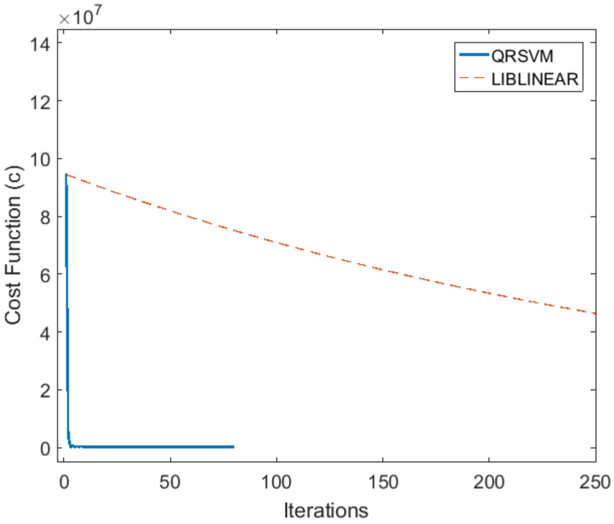

Figure 4.10: Convergence rates of QR-SVM vs LIBLINEAR. QR-SVM converges relatively faster compared to LIBLINEAR. Here, we illustrate for 250 iterations. LIBLINEAR was not able to converge to the optimal cost value in 1500 iterations while QR-SVM converged to the optimum in 80 iterations.

ality but large with respect to the number of data points, i.e., datasets which fit the tall-skinny topology. The selected benchmarks were split into training and testing datasets either as recommended in the repository or with a ratio of 85:15 otherwise. In Table 4.1, we show the QR decomposition time (t_{QR}), dual ascent iterations time (t_{DA}) and the overall training time ($t_{QR} + t_{DA}$) for the selected benchmarks.

Table 4.1: QR-SVM training time details

Dataset	Size	n	d	t_{QR} (s)	t_{DA} (s)	$t_{QR} + t_{DA}$ (s)
skin_nonSkin	3.2 MB	200,000	3	0.50	0.81	1.31
covtype.binary	75.2 MB	500,000	54	14.64	30.61	45.25
SUSY	2.39 GB	4,500,000	18	14.82	365.53	380.35
HIGGS	8.04 GB	10,500,000	28	89.05	642.30	731.35

Table 4.2: Comparison - QR-SVM and LIBLINEAR

Dataset	Training Time QR-SVM	Training Time LIBLINEAR	Speedup	Testing Accuracy QR-SVM	Testing Accuracy LIBLINEAR
skin_nonSkin	**1s**	42s*	42×	**92.42**	77.76
covtype.binary	**45s**	3m 46s*	5×	**63.23**	61.94
SUSY	**6m 20s**	47m 2s*	7×	78.71	**78.74**
HIGGS	**12m 11s**	2h 37m 37s*	13×	**64.06**	63.34

$*$ Convergence not achieved in 1500 iterations.

Table 4.2 presents the comparison between the training time and the testing accuracy of the QR-SVM with those of LIBLINEAR. In this work, our focus is on improving the convergence time of linear SVC problems. It can be observed that the proposed QR-SVM framework allows for fast training of SVC model compared to LIBLINEAR while providing relatively better testing accuracy for linear classifiers. The stopping thresholds of both QR-SVM and LIBLINEAR were set to 0.01 i.e., if c^k is the value of cost function in equation (4.8) at the k$^{\text{th}}$ iteration, then the algorithm will converge at the criterion $|c^{k+1} - c^k| \leq 0.01$. As mentioned earlier, Fan's[20] LIBLINEAR implementation was designed to stop at a preset maximum iteration. We chose the termination count as 1500 for our evaluation. We observed that for all the benchmark datasets used, LIBLINEAR implementation prematurely terminated at the maximum iteration count even before converging to the optimal solution. For the proposed QR-SVM, convergence was achieved much faster in a relatively smaller number of iterations. The convergence comparison for HIGGS dataset is shown in Figure 4.10.

4.7 Conclusion

In this chapter, we proposed a QR decomposition framework for linear support vector classification problems that uses Householder transformation to convert a dense, non-separable and low-rank matrix to a highly sparse and separable matrix, with invertible subproblems. In addition, we provide optimal step size for solving dual ascent method for faster convergence. We experimentally demonstrate that the proposed QR-SVM framework scales linearly with dataset size n making it suitable to handle big data problems with a tall-skinny structure.

5. CONCLUSION AND FUTURE WORK

In this thesis, we presented a novel implementation of SVM classification problems via a QR decomposition framework. In the process of formulating this framework, we also explored the concept of relaxed synchronization and improved the traditional dual ascent algorithm. Below is a summary of the research contributions of this thesis.

1. An analytical model of lazily synchronized dual ascent algorithm for solving quadratic programming problem with the derivation for optimal synchronization period was introduced. The strategy presented not only reduced the communication between distributed compute nodes but also improved the rate of convergence of the algorithm. Experimental results were presented to validate all the theoretical conclusions.

2. A QR decomposition framework for solving large-scale SVM classification problems with low dimensionality was formulated. Leveraging the benefits of LSDA algorithm, we experimentally showed that the QR-SVM formulation outperformed the state of the art solver - LIBLINEAR. A through complexity analysis of this algorithm was also presented which showed that the algorithm's execution time scales linearly with the number of data points in the dataset.

5.1 Future Scope

Currently, the QR-SVM framework has been implemented to run efficiently in uniprcessor systems with a potential parallelization which would yield additional benefits. The next step in the process of improving QR-SVM is to move towards a parallelized version of QR-SVM by distributing the iterative update calculations during the optimization stage.

In the future, we also plan to explore other machine learning tasks which involve solving a QP problem and validate the efficiency of LSDA algorithm in solving those tasks. Eventually, we plan to use LSDA as an optimizing engine and develop dedicated hardware to further improve the performance of solving these data intensive machine learning problems. Such dedicated hardware solutions can be built in two stages

- The optimizer module which is an implementation of LSDA algorithm in a generic form. This module can be reused based on the higher level application the hardware is targetted to solve.
- The application module which is an application specific hardware which is dedicated to solving higher level tasks. In the case of QR-SVM, this module is expected to QR decompose a data matrix into the R matrix and the Householder reflectors, and pass on this information to the optimizer module.

Such a solution is modular as the optimizer module remains fixed irrespective of application the hardware is expected to solve. The application module, on the other hand, is more specific to the task and needs to be swapped in and out based on the high-level application.

REFERENCES

[1] YS Abu-Mostafa, M Magdon-Ismail, and HT Lin. *Learning from data*, volume 4. AMLBook Singapore, 2012.

[2] Shereen Moataz Afifi, Hamid Gholamhosseini, and Roopak Sinha. Hardware Implementations of SVM on FPGA : A State-of-the-Art Review of Current Practice. *International Journal of Innovative Science Engineering and Technology (IJISET)*, 2(11), 2015.

[3] Mohammad A. Alsheikh, Shao W. Lin, Dusit Niyato, and Hp Tan. Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018, 2014.

[4] Asa Ben-Hur, David Horn, Hava T Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[5] Kristin Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1(1):23–34, 1992.

[6] Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, New York, New York, USA, jul 1992. ACM Press.

[8] Léon Bottou. Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT'2010*, pages 177–186, 2010.

[9] Stephen Boyd, N Parikh, B Peleato E Chu, and J Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, jan 2010.

[10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, apr 2011.

[11] Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines. *Journal of Machine Learning Research*, 9:1369–1398, 2008.

[12] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[13] B. J Culpepper and M. Gondree. SVMs for Improved Branch Prediction. *University of California, UCDavis, USA, ECS201A Technical Report*, 2005.

[14] GB Dantzig and P Wolfe. Decomposition principle for linear programs. *Operations research*, 1960.

[15] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13*, volume 41, page 559, New York, New York, USA, 2013. ACM Press.

[16] Harris Drucker, Christopher J C Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support Vector Regression Machines. In M I Jordan and

T Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[17] Kai-Bo Duan and S. Sathiya Keerthi. Which Is the Best Multiclass SVM Method? An Empirical Study. *Multiple Classifier Systems*, 3541:278–285, 2005.

[18] E.˜Osuna, R.˜Freund, and F.˜Girosi. An Improved Training Algorithm for Support Vector Machines. *Neural Networks for Signal Processing VII — Proceedings of the 1997 {IEEE} Workshop*, pages 276˜–˜285, 1997.

[19] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-stover. GPU Cluster for High Performance Computing. *IEEE Supercomputing*, 00(1):47, 2004.

[20] C.-J Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[21] Walter Gander. Algorithms for the qr decomposition. In *Seminar für Angewandte Mathematik: research report*, 1980.

[22] David Goldberg and John Holland. Genetic Algorithms and Machine Learning. *Machine Learning*, 3:95–99, 1988.

[23] T O M Goldstein, Brendan O Donoghue, Simon Setzer, and Richard Baraniuk. Fast alternating direction optimization methods. *SIAM Journal of Imaging Sciences*, 7(3):1588–1623, 2014.

[24] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel Support Vector Machines : The Cascade SVM. *In Advances in Neural Information Processing Systems*, pages 521–528, 2005.

[25] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A Dual Coordinate Descent Method for Large-scale Linear SVM. In

Proceedings of the 25th International Conference on Machine Learning - ICML '08, number 2, pages 408–415, New York, New York, USA, jul 2008. ACM Press.

[26] Amazon Web Services Inc. Amazon elastic compute cloud documentation, 2015.

[27] Kevin M. Irick, Michael DeBole, Vijaykrishnan Narayanan, and Aman Gayasen. A hardware efficient support vector machine architecture for FPGA. *Proceedings of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'08*, pages 304–305, 2008.

[28] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[29] Steven G. Johnson. 18.335J - Introduction to Numerical Methods, Fall 2010. *MIT OpenCourseWare: Massachusetts Institute of Technology*, 2010.

[30] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, 13(3):637–649, mar 2001.

[31] Peter Kogge, K Bergman, S Borkar, Dan Campbell, W Carson, W Dally, M Denneau, P Franzon, W Harrod, K Hill, Jon Hiller, Mark Richards, and Allan Snavely. ExaScale Computing Study : Technology Challenges in Achieving Exascale Systems. *Government Procurement*, TR-2008-13:278, 2008.

[32] Kooktae Lee, Raktim Bhattacharya, Jyotikrishna Dass, VNS Prithvi Sakuru, and Rabi N Mahapatra. A relaxed synchronization approach for solving parallel quadratic programming problems with guaranteed convergence. *Parallel & Distributed Processing (IPDPS), 2016 IEEE 30th International Symposium on, p. to appear, IEEE*, 2016.

[33] Kooktae Lee, Raktim Bhattacharya, and Vijay Gupta. A switched dynamical system framework for analysis of massively parallel asynchronous numerical algorithms. *Proceedings of the American Control Conference*, 2015-July:1095–1100, 2015.

[34] M. Lichman. UCI machine learning repository, 2013.

[35] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A unified template-based framework for accelerating statistical machine learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 14–26. IEEE, 2016.

[36] A Nedic and A Ozdaglar. 10 Cooperative distributed multi-agent. *Convex Optimization in Signal Processing and Communications*, 2010.

[37] John C Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods Support Vector Learning*, 208:1–21, 1998.

[38] S. Ruping. Incremental learning with support vector machines. *Proceedings 2001 IEEE International Conference on Data Mining*, pages 0–1, 2001.

[39] Conrad Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[40] B Schölkopf, J C Platt, J Shawe-Taylor, a J Smola, and R C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, jul 2001.

[41] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling Concept Drifts in Incremental Learning with Support Vector Machines. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 317–321, New York, New York, USA, aug 1999. ACM Press.

[42] The MPI forum. MPI: A Message-Passing Interface Standard. Technical report, University of Tennessee, Knoxville, TN, USA, 1994.

[43] V N Vapnik. *Estimation of Dependences Based on Empirical Data, New York: Springer-Verlag.* 1982.

[44] Guo Xun Yuan, Chia Hua Ho, and Chih Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[45] Caoxie Zhang, Honglak Lee, and KG Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. *International Conference on Artificial Intelligence and Statistics*, 2012.

APPENDIX A

QR DECOMPOSITION

A matrix decomposition technique to transform a matrix A, into two factor matrices Q and R where Q is an orthogonal matrix and R is an upper triangular matrix.

$$A = QR$$

If A is a rectangular matrix of size $n \times d$, then the Q matrix is of size $n \times n$ while the matrix R is of size $n \times d$. Figure A.1 shows the sparsity map of the Q and R matrix structures.

Some of the techniques to compute the QR decomposition of a matrix are the Gram-Schmidt process, the Householder's transformation and Given's rotation. Each of the techniques has their own merits and issues[21]. Householder transformation is particularly chosen for the following merits:

1. Greater numerical stability than Gram-Schmidt process.

2. Smaller memory footprint compared to both Gram-Schmidt and Givens rotation. Using Householder transformation, we can save the Householder reflectors (rather than the explicit Q matrix) and R matrix in the same memory space as that of the original matrix A.

3. Less number of arithmatic operations.

4. Efficient to apply Q and Q^T to a vector (which is the bulk of the calculations in our formulation detailed in chapter 4.5)

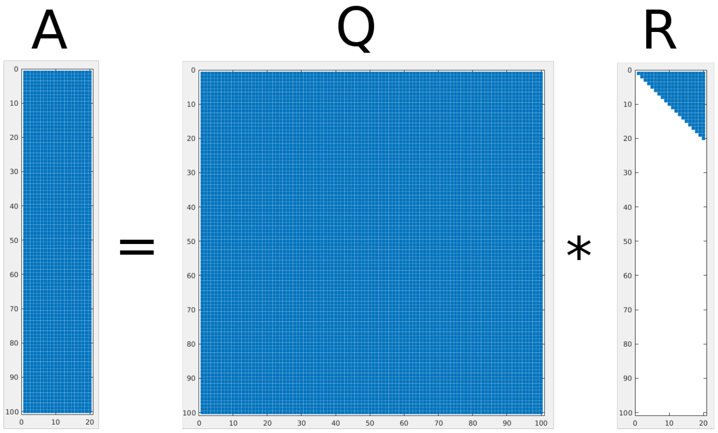

Figure A.1: Sparsity map of QR decomposition.

Algorithm 2 QR Decomposition - Householder

```
for $k = 1 : d$ do
    $v_k = A_{k:n,k}$
    $v_k(1) \mathrel{+}= \text{sign}(x(1))||x||_2$
    $v_k = v_k/||v_k||_2$
    $A_{k:n,k:d} = A_{k:n,k:d} - 2v_k(v_k^T A_{k:n,k:d})$
```

A.1 Householder Transformation

Householder Transformation is a QR decomposition technique which considers each column of the original matrix, A as a vector and transforms it into an upper triangular matrix one column at a time. This upper triangular matrix thus formed is the R matrix while each of the reflector matrices used in the transformation can be combined to form the Q matrix of the QR decomposition. The basic idea is to find the reflector hyperplane corresponding to each of the column vector such that the reflection of the vector will lie on the axis of the first element, thereby ensuring zeros on rest of the elements. A simple illustration can be seen in figure A.2.

For a rectangular matrix with $d < n$, there are d Householder reflectors, i.e. one

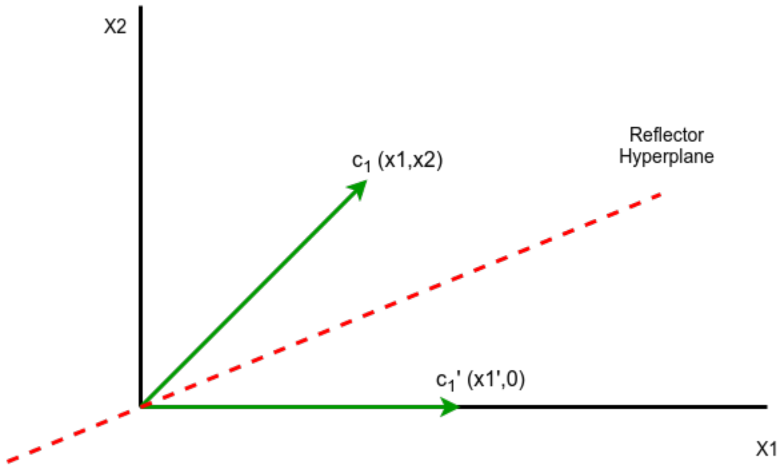

Figure A.2: Householder transformation on a vector in 2D space.

Algorithm 3 Calculation of $Q^T b$ - Householder

```
for $k = 1 : d$ do
    $b_{k:n} = b_{k:n} - 2v_k(v_k^T b_{k:m})$
```

for each of the columns. Algorithm 2 corresponds to the QR decomposition of a matrix using Householder transformations. The computation cost of this algorithm is $O(nd^2)$. Note that the Q matrix is not explicitly computed. We are only computing and storing the Householder reflectors v_k.

After Householder decomposition, the matrix A can be written as a product of multiple reflector matrices and an R matrix as follows

$$A = Q_d Q_{d-1} ... Q_2 Q_1 R$$

Here, each of Q_k is a matrix of size $n \times n$ which can be generated using the k$^{\text{th}}$

Algorithm 4 Calculation of Qb - Householder

```
for $k = d : 1$ do
    $b_{k:n} = b_{k:n} - 2v_k(v_k^T b_{k:m})$
```

Householder reflector as

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & F_k \end{bmatrix}_{n\times n} \quad \text{where } F_k = I - 2\frac{v_k v_k^T}{v_k^T v_k}$$

The Q matrix is never explicitly calculated for most practical cases due to large computation and memory costs. Rather, Q matrix is applied to a vector directly at a reduced cost of $O(nd)$ as shown in algorithms 3 and 4.