HIGH-PERFORMANCE COMPUTING FOR SUPPORT VECTOR MACHINES

LICENTIATE THESIS

HIGH-PERFORMANCE COMPUTING FOR SUPPORT VECTOR MACHINES

SHIRIN TAVARA
Informatics

UNIVERSITY
OF SKÖVDE

Shirin Tavara, 2018

Licentiate Thesis

Title: High-Performance Computing For Support Vector Machines

University of Skövde, Sweden
www.his.se

Printer: BrandFactory AB, Göteborg

ISBN 978-91-984187-8-1

Dissertation Series No. 26 (2018)

ABSTRACT

Machine learning algorithms are very successful in solving classification and regression problems, however the immense amount of data created by digitalization slows down the training and predicting processes, if solvable at all. High-Performance Computing (HPC) and particularly parallel computing are promising tools for improving the performance of machine learning algorithms in terms of time. Support Vector Machines (SVM) is one of the most popular supervised machine learning techniques that enjoy the advancement of HPC to overcome the problems regarding big data, however, efficient parallel implementations of SVM is a complex endeavour. While there are many parallel techniques to facilitate the performance of SVM, there is no clear roadmap for every application scenario.

This thesis is based on a collection of publications. It addresses the problems regarding parallel implementations of SVM through four research questions, all of which are answered through three research articles. In the first research question, the thesis investigates important factors such as parallel algorithms, HPC tools, and heuristics on the efficiency of parallel SVM implementation. This leads to identifying the state of the art parallel implementations of SVMs, their pros and cons, and suggests possible avenues for future research. It is up to the user to create a balance between the computation time and the classification accuracy. In the second research question, the thesis explores the impact of changes in problem size, and the value of corresponding SVM parameters that lead to significant performance. This leads to addressing the impact of the problem size on the optimal choice of important parameters. Besides, the thesis shows the existence of a threshold between the number of cores and the training time. In the third research question, the thesis investigates the impact of the network topology on the performance of a network-based SVM. This leads to three key contributions. The first contribution is to show how much the expansion property of the network impact the convergence. The next is to show which network topology is preferable to efficiently use the computing powers. Third is to supply an implementation making the theoretical advances practically available. The results show that graphs with large spectral gaps and higher degrees exhibit accelerated convergence. In the last research question, the thesis combines all contributions in the articles and offers recommendations towards implementing an efficient framework for SVMs regarding large-scale problems.

HIGH-PERFORMANCE COMPUTING FOR SUPPORT VECTOR MACHINES

LICENTIATE THESIS

HIGH-PERFORMANCE COMPUTING FOR SUPPORT VECTOR MACHINES

SHIRIN TAVARA
Informatics

UNIVERSITY
OF SKÖVDE

Shirin Tavara, 2018

Licentiate Thesis

Title: High-Performance Computing For Support Vector Machines

University of Skövde, Sweden
www.his.se

Printer: BrandFactory AB, Göteborg

ISBN 978-91-984187-8-1

Dissertation Series No. 26 (2018)

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude towards my main supervisor Alexander Schliep for providing excellent feedback and professional discussions besides his friendly attitude. I would also like to thank my co-supervisor, Alexander Karlsson for his scientific insights. Next, I thank Lars Niklasson, Håkan Sundell, Anders Dahlbom, Eva Söderström, and all my colleagues and friends at University of Borås, Skövde and particularly at Chalmers, who made the working environment a pleasant place to be.

Foremost, I would like to thank my husband, Niclas, for his infinite support, love, and encouragement academically and spiritually throughout writing this thesis and my life in general. I am grateful for his care and patience during my irregular working hours and the lack of availability. I am very thankful to my mother, Hamideh, and brother, Shahram, for their endless love and support especially during my PhD study. Without the precious support of my family, it would not be possible to conduct this research.

This work is funded mainly by the University of Borås and was founded partly by the University of Skövde.

PUBLICATIONS

The publications listed below are part of the licentiate thesis. The author is the sole author of the first publication and the main driver of the second and third publications listed below.

RESEARCH ARTICLES

1. Tavara, Shirin (2018). “Parallel Computing of Support Vector Machines: A Survey.” In: *ACM Computing Surveys (CSUR)*.
2. Tavara, S, H Sundell, and A Dahlbom (2015). “Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM.” In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p. 177.
3. Tavara, Shirin and Alexander Schliep (2018). “Effect Of Network Topology On The Performance Of ADMM-based SVMs.” In: *High-Performance Machine Learning Workshop (HPML)*.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

In recent years, we have seen a dramatic increase in the amount of data created by digitalization. Over 90% of digital data in the world was generated over the years 2015-2017, i.e., 2.5 quintillion bytes of data every day (Domo, 2018a). It is estimated that by 2020, 1.7 MB of digital data will be generated per person/per second (Domo, 2018b). Social media and search engines are examples of fields with exponentially growing data. Consequently, the need for tools that can automatically learn, analyze, and predict has grown. Machine learning provides such tools that can automatically understand data and draw conclusions as needed. In order to harness the power of machine learning methods for large-scale problems, High-Performance Computing (HPC) (Yang and M. Guo, 2005) and particularly parallel programming are necessary.

One of the machine learning methods that have been most widely used is Support Vector Machines (SVM) (V. N. Vapnik, 1999). SVM can take advantage of parallelism, however an efficient parallel implementation of SVM is a complex endeavor (H. P. Graf, Cosatto, et al., 2004). While there are many parallel techniques to improve the performance of SVM, there is no clear roadmap for every application scenario. Existing parallel approaches may become inefficient for solving large-scale problems or using a large number of computing nodes, i.e., they may not scale well to a large number of samples or processors. Problems such as communication overhead, computationally dependent steps, and memory limitations degrade the efficiency of parallelism (G.-X. Yuan, Ho, and C.-J. Lin, 2012). Besides, coding part of a parallel SVMs algorithm is difficult and requires considerable skills (Byun and Lee, 2002a). Due to the high space and time complexity of SVM algorithms, it is important to identify and use appropriate algorithms and efficient heuristics that fit the characteristics of the given problem along with the parallel settings. To identify the appropriate parallel approaches that have potentials to obtain the peak performance for the given large-scale problem, a key issue is to thoroughly study the efficiency of existing parallel approaches. Although some attempts have been conducted to identify the parallel approaches (Tyree et al., 2014; Byun and Lee, 2002a), to our knowledge, there is no work that thoroughly reviews and identifies the parallel algorithmic approaches along with parallel tools for efficient implementations of SVM. Distributed optimizations, parallel incremental learning and a parallel cascade of SVMs are examples of algorithmic approaches. Distributed HPC architecture, distributed big data architecture, shared memory and GPU-based parallelism are examples of parallel tools. Another key issue in the application of machine learning algorithms is to supply efficient implementations making the theoretical advances practically available. This helps to identify the limitations of theoretical principles in practice and new settings. For instance, the high dimensionality of problems and a large number of training samples that lead to computationally expensive tasks are some of the hurdles in the implementations of machine learning algorithms. In order to implement efficient algorithms and to harness the power of parallelism, the role of empirical studies and investigation of efficient methods/heuristics in parallel settings is important.

This thesis aims to address the issues regarding parallel implementations of SVM for

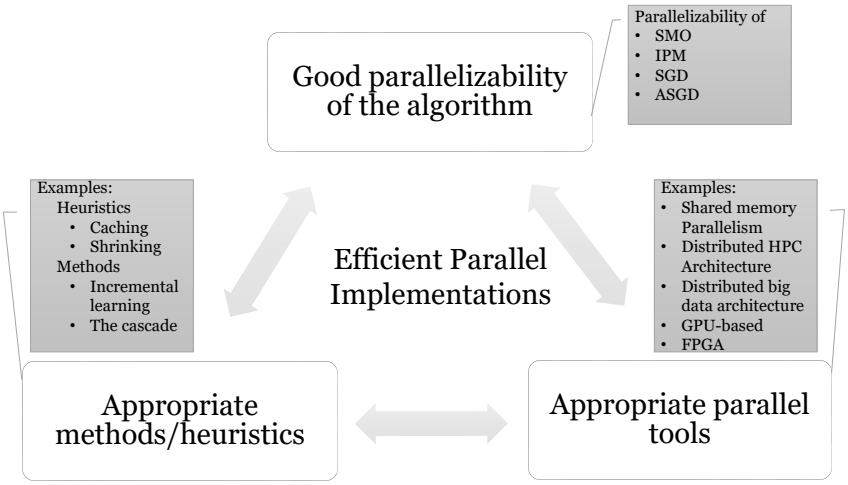


Figure 1.1: Important elements of efficient parallel implementations

large-scale problems in practice. In this regard, the important elements of efficient implementations of SVM have been identified. These elements are shown in Figure 1.1. In addition, two empirical studies are designed and conducted to address issues regarding the overhead and efficient communication between computational nodes. Beside, corresponding implementations are provided to make theoretical advances practically available.

1.1 PARALLEL SVMS - CHALLENGES

Simple SVM classification problems comprise two processes, training and predicting. The training phase of SVM involves a dense Quadratic Programming (QP) problem (Woodsend and Gondzio, 2009a) and solving such a QP is computationally expensive since it involves computations of a large Hessian/kernel matrix. Kernel evaluations include computationally expensive tasks, i.e., matrix-vector and matrix-matrix multiplications, gradient function and optimality condition updates. The old and general-purpose QP solvers are no longer suitable for solving the QP resulting from SVMs since many of QP solvers require computing and storing the kernel matrix in the memory, which is not always possible due to the memory limitations. One efficient approach to improve the performance of SVM algorithms is parallelism, however, the parallel approach used in the general-purpose QP solvers may not take the advantage of the special characteristics of SVMs (see article I) or they may not be easily parallelizable. Besides, the immense size of real-world data can cause problems as follows,

- **Memory.** The whole data may not fit into the memory or inefficient memory access slows down the training and testing phases.

- **Speedup.** The matrix operations might take too long time to be performed due to the computationally expensive tasks, e.g., matrix-vector, matrix-matrix multiplications, and overheads.

- **Scalability.** Algorithms may not scale well to a large number of processors or a large number of samples.
- **Accuracy**. Approximation methods for reducing the size of the problem or dividing the original problem into several small problems may lead to poor classification accuracy.

Besides the important points in parallelism mentioned above, several common heuristics, e.g., caching (Matsushima, Vishwanathan, and Smola, 2012), shrinking (Joachims, 1998), data reordering (Durdanovic, Cosatto, and H.-P. Graf, 2007) and data movement (Liao et al., 2009), are used to further improve the performance of SVM algorithms. As mentioned in article I, one of the challenges in parallel implementations of SVMs is how and to what extent these heuristics can be used.

1.1.1 RESEARCH PROBLEM

The immense amount of data generated by digitalization necessitates using HPC tools and particularly parallel computing for improving the performance of machine learning algorithms. The well-studied tools in this regard are shared memory parallelism (H. Zhao and Magoules, 2011; Didiot and Lauer, 2015; Eitrich and Lang, 2005; Marzolla, 2011; Gonçalves, Lopes, and Ribeiro, 2012; P. Chang, Bi, and Feng, 2014; Eitrich and Lang, 2006), distributed HPC architectures (Brugger, 2006; L. Cao et al., 2006; Athanasopoulos et al., 2011; Narasimhan et al., 2014; Vishnu et al., 2015; Ferreira, Kaszkurewicz, and Bhaya, 2006), hybrid shared-distributed memory parallelism (Doan, Do, and Poulet, 2013; Woodsend and Gondzio, 2009b), distributed big data architectures (H. Zhao and Magoules, 2011; Caruana, M. Li, and Qi, 2011; Alham, M. Li, and Liu, 2014; Q. He et al., 2011), Field Programmable Gate Arrays (FPGA) (Papadonikolakis, Bouganis, and Constantinides, 2009) and most popular GPU-based parallelism (Do, Nguyen, and Poulet, 2008; Peng, Zhang, and Y. Zhao, 2011; Carpenter, 2009a; Athanasopoulos et al., 2011). Nevertheless, the efficient parallelization of SVM algorithms is far beyond an easy task (H. P. Graf, Cosatto, et al., 2004) due to challenges as follows,

1. **Dependencies between the computation steps.** This concerns the parallelizability of SVM algorithms. For instance, in order to obtain the results in the current step of the Sequential Minimal Optimization (SMO) algorithm (Platt, 1998), the result from the previous step is required (H. P. Graf, Cosatto, et al., 2004). As another example, an SVM algorithm may be implemented in a way that the modern processor technologies, e.g., SSE, AVX, or optimization options are not applicable (Tavara, Sundell, and Dahlbom, 2015). In most cases, re-implementing the algorithms is a hurdle and not straightforward. Therefore, it is important to identify the algorithms that have good parallelizability.
2. **High latency in memory access.** Careful access of data from memory, in which it follows the appropriate pattern of the underneath architecture, is significant. An access pattern that does not follow the architecture leads to high latency and if an algorithm is a memory bound, the memory access cost will be high and then multithreading leads to memory contention (You, Song, et al., 2014). For instance, GPUs have certain restrictive memory access patterns which can cause high latency if the patterns are not followed properly (Cotter, Srebro, and Keshet, 2011).
3. **Limited memory.** This concern mostly regards to the shared memory and GPU-based parallelism, albeit it can also relate to the distributed memory parallelism for

large-scale problems in which the data do not fit in the available memory. Approximation methods play an important role to reduce the size or dimension of the data, however, they may cause deterioration of the classification (You, Song, et al., 2014).

4. **Overheads.** Communication and synchronization overheads are hurdles in parallel implementations of machine learning algorithms (E. Y. Chang, 2011). Many parallel implementations exhibit accelerated performance in each core while combining or exchanging results from several cores leads to higher overhead, hence lower speedups. In this regard, how to exchange data and results between distributed cores affects the efficiency of parallelism. Therefore, the network topology of nodes plays an important role in the efficiency of parallel implementations (Chow et al., 2016; H. T. Cao et al., 2016).
5. **Dedicated hardware.** Dedicated hardware for performing computationally expensive tasks has been used for parallel implementations of machine learning algorithms. FPGA for example leads to accelerated performance, however expertise in the dedicated architecture/hardware is required to efficiently perform the tasks. Furthermore, the hardware may not be always available. This motivates developing parallel implementations on standard workstations.

Consequently, how and which HPC and parallel tools along with algorithmic approaches can be employed will impact the performance of the algorithm in question and the applied computing power. To approach these issues and overcome some of the mentioned hurdles, this thesis aims to 1) investigate the efficiency of HPC and parallel tools on the performance of SVM algorithms (addresses challenge 1-5), 2) to improve the performance of a SVM algorithm by conducting simple and important changes that allows using modern processor technologies, besides introducing a threshold between time and the number of cores (addresses challenge 1-4), 3) study the impact of network topology in a distributed consensus-based SVM in order to get the maximum possible advantage of parallelism (addresses challenge 1-4), and 4) suggest constructive approaches towards developing an efficient parallel framework.

1.2 RESEARCH QUESTIONS

In order to address the problems mentioned in 1.1.1, the main objectives of this thesis are elaborately presented in the form of four research questions all of which are answered by three research articles.

1. *What are the efficient HPC and parallel programming tools for improving an SVM algorithm regarding large-scale problems?*
   Today, while there are many HPC and parallel computing tools employed for improving the performance of machine learning algorithms, there is no clear roadmap or suggestion for every application scenario. Besides, parallel settings may not fit or have the maximum possible performance on different algorithms, e.g., an Sequential Minimal Optimization (SMO) (Platt, 1998) algorithm for solving SVM problems may not be as efficient as an Interior Point Method (IPM) (E. Y. Chang, 2011) algorithm in parallel distributed settings. This causes inefficient use of computing powers without significant gains, i.e., using few computing cores may have the same performance improvement as using many cores. This research question aims to present three key characteristics as follows,
   a Which SVM algorithms do fit the parallel settings better?
   b What are the efficient HPC tools and parallel programming and their pros and cons in the context of SVM?

   c What are potential avenues for future research in order to achieve high-performance SVMs?

2. *How can simple changes such as memory allocation, de-allocation, and alternating problem size and the number of computing cores impact the performance of an IPM-based SVM implementation in terms of the training time and the classification accuracy in parallel settings?*
   Based on the Amdahl's law (Null, Lobur, et al., 2014), the maximum speedup of a program using parallel computing with multiple processor is limited regardless of the number of processors. The maximum speedup is limited by sequential parts in the program, therefore parallelizing even small fractions of sequential parts can lead to significant improvement. Several SVM algorithms suffer from sequential parts, e.g., SMO. The sequential parts cannot always be performed in parallel, due to the dependencies between sequential steps in the algorithms. However, redesigning the algorithms can solve this issue in many cases. IPM-based SVMs have shown high-performance in parallel and distributed settings (E. Y. Chang, 2011; Woodsend and Gondzio, 2009a; T. Li et al., 2013; Jin, Cai, and Xiaola Lin, 2013; Chatterjee, Fermoyle, and Raghavan, 2010; Gertz and Griffin, 2010). The following research questions present some key characteristics as follows,

   a How can small changes, e.g., memory allocation, de-allocation, and using modern processor technologies, lead to significant improvement on the performance of an IPM-based SVM in terms of time and the classification accuracy?
   b How does the number of computational cores relate to the training time in an IPM-based SVM?
   c How does the size of the approximated kernel matrix in an IPM-based SVM affect the choice of the other parameters as well as the accuracy?

3. *What are the impacts of network topology on the performance of SVM algorithms in terms of convergence?*
   Network topology and the connectivity of the underlying graph have an impact on the performance of network-based SVMs in terms of convergence and the number of iterations until convergence. This research question investigates how network topology affects a distributed SVM algorithm. The elaborated sub-questions are as follows,

   a How much does the expansion property of expander graphs influence the convergence of a distributed consensus ADMM-based SVM algorithm?
   b Which topology is preferable in the context of expander graphs?
   c How to supply an implementation making these theoretical advances practically available?

4. *What recommendations of possible avenues for future research can be given for development of an efficient parallel framework that takes the maximum possible advantage of parallelism for solving SVM problems?*
   This research question aims to provide recommendations that facilitate the development of an efficient parallel framework for solving SVM problems. Potential recommendations can help scientists and developers to use the efficient approaches based on their demands and preference. Besides, these recommendations can make it possible to develop a framework that gathers all the efficient approaches under the same roof. These recommendations are given based on the findings in research questions 1, 2, and 3.
   Figure 1.2 shows the research design of the thesis regarding the research problem, research questions, objectives, and the corresponding research articles.

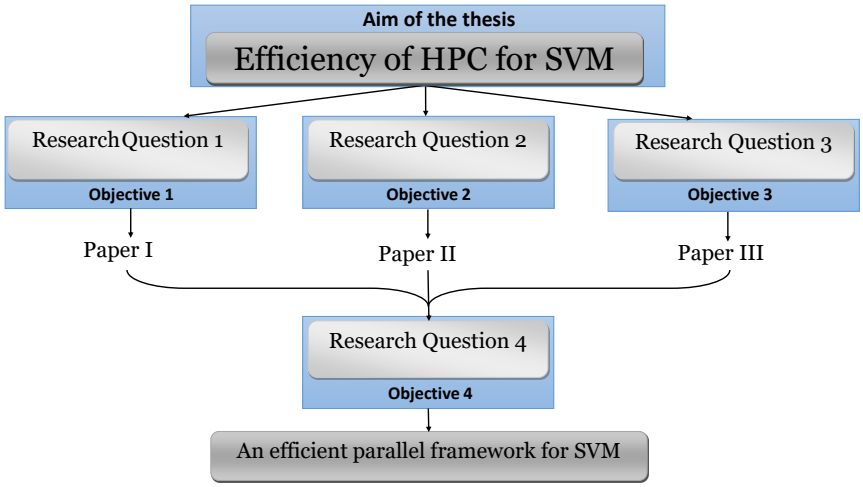


Figure 1.2: The research design regarding research problem, research questions, objectives, and research articles

1.3 CONTRIBUTION TO THE PUBLICATIONS

In this section, the author's contribution to each publication presented in this thesis is briefly described. The author has been the main driver of all the publications listed in this thesis, including defining the research problems, planning and designing the empirical studies and the experiments, conducting the experiments, analyzing the data and results, and writing the articles. Detail of the contributions is as follows,

1. **Article I.** Tavara, Shirin. (2018) Parallel Computing of Support Vector Machines: A Survey. Accepted in ACM Computing Surveys (CSUR) journal, 2018.
   - I have reviewed over 200 relevant articles to conduct the survey. I have had the sole responsibility of summarizing, structuring, analyzing, and comparing methods in the subject area. Besides, I have had the responsibility of identifying the gaps in the existing literature and suggesting constructive recommendations.
2. **Article II.** Tavara, Shirin, Sundell Håkan, and Dahlbom, Anders. (2015) Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Vol. 1, p. 177-183. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015, July. Printed in the United States of America: CSREA Press, 2015.
   - I am the main driver of this article and my contribution to this article includes; planning and designing the empirical study and the experiments, performing the experiments, analyzing data and results, and writing the article. My former supervisor and co-supervisors provided expertise in the field and feedback.
3. **Article III.** Tavara, Shirin, Schliep, Alexander. (2018) Effect Of Network Topology On The Performance Of ADMM-based SVMs. High Performance Machine Learning Workshop, HPML 2018, September 24, Lyon, France.

- I am the main driver of this article and my contribution to this article is 1) the planning and designing of the empirical study and the experiments, 2) performing the experiment, and 3) writing the article. My current supervisor has provided expertise in expander graphs along with giving the constructive feedback to improve the quality of the article.

1.4 SCOPE OF THIS WORK

This thesis focuses on a particular machine learning algorithm, SVM. This algorithm is one of the most popular, successful, and widely used algorithms for classification and regression algorithms (Byun and Lee, 2002b; V. Vapnik, 2013; Bin, Yong, and Shao-Wei, 2000). Some of the special characteristics of SVM are as follows,

- The strong statistical theory underneath SVM (V. Vapnik, 2013)
- Good generalization performance on unseen data (Byun and Lee, 2002b)
- The contribution of only a few samples in the decision boundary (Carpenter, 2009b). This makes SVM to be a powerful technique to handle large-scale problems
- Performing nonlinear mapping without any knowledge of the mapping function (Ivanciuc, 2007a)
- A wide and successful range of applications (Ivanciuc, 2007a).

Other machine learning methods can follow some of the approaches used in SVM to improve the performance of the algorithm in parallel settings, albeit some of the approaches are adjusted to fit the special characteristics of SVM to reach the pick performance.

1.5 THESIS OUTLINE

This thesis includes five chapters. The introduction of the research area, including the challenges in the parallel implementations of SVM, is described in Chapter 1 and it is continued by addressing the research problem. In addition to that, the chapter includes the description of the research questions as the objectives of this thesis. Besides, the contribution of the author to the presented publications along with the scope of the thesis is presented in this chapter. A brief overview of the employed algorithmic techniques, methods, and parallel models are presented in Chapter 2. The research methodology of the thesis is described in Chapter 3. It includes the literature review process and the material, followed by the description of our conducted empirical studies and experiments. It also briefly discusses the research design used in this thesis. The results of all the research articles and how they answer the corresponding research questions are presented in Chapter 4. Finally, a brief summary of research articles and their contributions to the research questions are mentioned in Chapter 5. The chapter further includes the concluding remarks and the contribution of the thesis in the field of machine learning. Moreover, it mentions the suggested possible avenues for future research.

CHAPTER 2
BACKGROUND

This chapter briefly summarizes the background knowledge used in the research presented in this thesis.

2.1 SUPPORT VECTOR MACHINES

SVM is one of the popular and successful supervised learning methods that are widely used for classification and regression problems (Cortes and V. Vapnik, 1995). SVM fits data by maximizing the margin around the seperator. This leads to good generalization performance regarding new unseen data, i.e., training data are classified into classes with the given labels with a separator that is farthest as possible to both classes of points and this generalize well to new data. The maximum margin of the separator of classes leads to the strong upper bound of the generalization error. SVM minimizes this upper bound (Byun and Lee, 2002a).

In a simple binary classification in SVM, the machine is trained to find a linear separator that classifies the given training data into two classes with known labels, this is the training phase. After the machine is trained, the model is extracted to predict which class label a new unseen data belongs to, this is the testing or predicting phase.

SVM has special characteristics that are used to implement efficient parallel algorithms in terms of time and memory. One characteristic is that the solution to a classification problem is obtained by only a few samples called Support Vectors (SV) (V. Vapnik, 2013; Carpenter, 2009a) that determine the maximum margin separating hyperplane (Ivanciuc, 2007b). Another characteristic of SVM is to perform the nonlinear mapping without knowing the mapping function using predefined functions called kernels for calculating the inner product of mapping functions (Ivanciuc, 2007b). Other characteristics of SVM are the simple structure of constraints and the especial definition of the kernel function in a linear case, i.e., the inner product is a simple dot product (Zanghirati and Zanni, 2003).

Hard-margin linear SVM. In this case, a linear separator classifies data into given classes with the maximum margin from the closest points (see Figure 2.1). The closest data points to the separator are SVs. A linear separable SVM or a hard-margin linear SVM solves the optimization problem,

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 \quad \text{for} \quad i = 1, 2, ..., N. \tag{2.1}$$

Here, $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}_i$ is a vector of observations, y_i is the class labels, b is the bias parameter.

Soft-margin linear SVM. Real-world data are not always completely separable. In this case, a linear separator separates data into the given classes with a maximum margin while minimizing the misclassification errors (Platt, 1998)(see Figure 2.1 b). To do so,

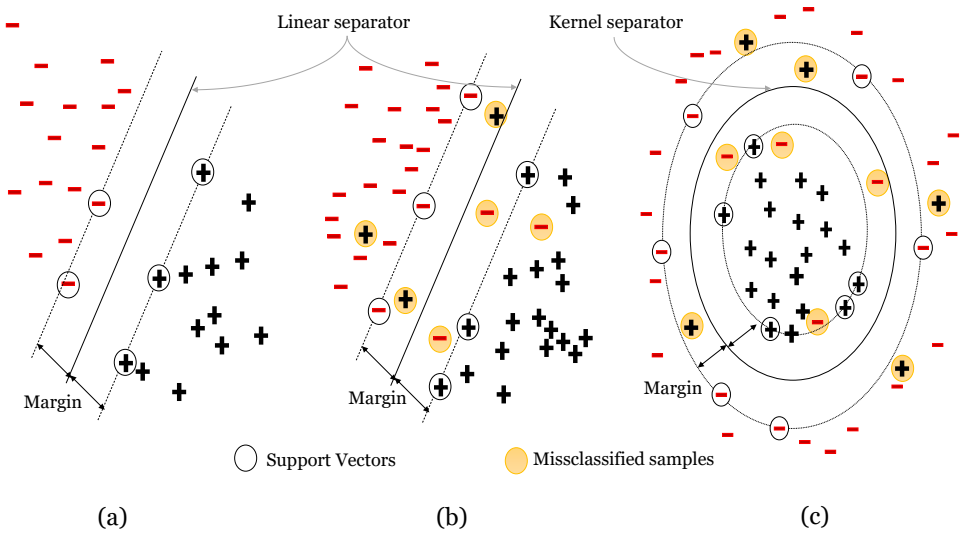


Figure 2.1: a) Hard-margin linear SVM classifier, b) Soft-margin linear SVM classifier, and c) Kernel SVM classifier

positive slack variables, ξ_i, enter the primal optimization problem, i.e.,

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \xi_i \quad \& \quad \xi_i \geqslant 0 \quad \text{for} \quad i = 1, 2, ..., N. \tag{2.2}$$

Kernel SVM classifier. A linear classifier cannot always classify real-world data (Byun and Lee, 2002a). In this case, data can be mapped into a higher dimensional space using a non-linear transformation function,Φ, and then in the feature space data can be linearly separated. The non-linear transformation is done through a function called the kernel function. Figure 2.1 c shows a non-linear kernel SVM classifier. SVM is a good example of kernel methods that uses a kernel trick in which an inner product of the mapping function is replaced by a kernel function. A non-linear soft-margin SVM solves the same primal optimization problem as in (2.2), however in order to use a kernel trick, it transforms the primal optimization into the Lagrange dual optimization (2.3),

$$\max \quad \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \quad \text{s.t.} \quad \sum_{i=1}^{N} y_i\alpha_i = 0 \quad \& \quad 0 \leqslant \alpha_i \leqslant C \quad \text{for} \quad i = 1, 2, ..., N. \tag{2.3}$$

Here, $\boldsymbol{\alpha}$ is the vector of Lagrangian multipliers and $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_N\}$, $\mathbf{1}^T$ is a vector of ones and Q is a matrix of size $N \times N$, where $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$.

Kernel functions. In order to compute matrix Q, it is sufficient to compute the inner product of $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$ without knowing the $\Phi(\mathbf{x})$ function. This is done through a pre-defined kernel function, $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. The kernel matrix measures the similarity or the distance between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ (Platt et al., 1998). Examples of well-known kernel functions are presented in table 2.1.

The kernel function defines the feature space where the training samples are classified, therefore the selection of an appropriate kernel function is important (T. Li et al., 2013).

Kernel Function	Inner Product	Kernel Type
Linear kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	linear
Gaussian/RBF	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$	non-linear
Polynimial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + const)^d$	non-linear
Laplacian	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|)$	non-linear

Table 2.1: Examples of well-known kernel functions

Multi-class classification. SVM can solve multi-class classifications either by embedding it in the optimization problem or through decomposing the multi-class classification into a series of binary class classifications (Doan, Do, and Poulet, 2013). The latter group is popular and includes the methods as follows,

- *OVO*. In One-Versus-One (OVO) (Hwu, 2011) classification, also known as one-against-one, all binary combinations of classes are created. This means that if N different classes are available for the classification, then $N(N-1)/2$ classifiers are built (Doan, Do, and Poulet, 2013). For instance, if 5 different classes with labels $1, 2, 3, 4, 5$ are available for the multi-class classification, then $5(5-1)/2$ binary classifiers, $(1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5)$ and $(4,5)$, are built.
- *DAGs*. Directed Acyclic Graphs (DAGs) (Vural and Dy, 2004; Doan, Do, and Poulet, 2013) is a directed graph that combines the results of OVO classifiers (Vural and Dy, 2004).
- *OVA*. In One-Versus-All (OVA) (Hwu, 2011), also known as one-against-all, the samples of a specific class (class i) will be considered as the positive class and all the remaining samples will be considered as the negative class. This leads to generating N different classifiers (Doan, Do, and Poulet, 2013). For instance, if 5 different classes with labels $1, 2, 3, 4, 5$ are available for the multi-class classification, then 5 binary classifiers, $(1, \text{all others}), (2, \text{all others}), (3, \text{all others}), (4, \text{all others})$ and $(5, \text{all others})$ are built.

Multi-class classifications are computationally expensive and lead to long training time due to involving many classes in the classification and passing through the training data many times (L. Cao et al., 2006). For a large number of classes, OVO is more computationally expensive than the OVA (Doan, Do, and Poulet, 2013), since OVO builds $N(N-1)/2$ classifiers versus N classifiers in the OVA.

2.2 INTERIOR POINT METHOD

Different mathematical solvers are utilized to solve the primal (2.2), the dual (2.3), or the primal-dual optimization problem resulting from SVM. In this context, an Interior Point Method (IPM) (E. Y. Chang, 2011) has shown high-performance in parallel and distributed settings (E. Y. Chang, 2011; Woodsend and Gondzio, 2009a; T. Li et al., 2013; Jin, Cai, and Xiaola Lin, 2013; Chatterjee, Fermoyle, and Raghavan, 2010; Gertz and Griffin, 2010). IPM starts from an initial point located in the interior feasible region and moves towards the optimal point(s) in an iterative manner. One of the advantages of IPM is its high degree of inherent parallelism compared to other solvers. However, IPM may suffer from numerical unstableness or it may require computing the inverse of a

large matrix resulting from SVM which is computationally expensive. To overcome these hurdles, IPM uses 1) Cholesky Factorization (CF) (K. Zhu et al., 2008) for achieving numerical stableness and 2) Incomplete Cholesky Factorization (ICF) (K. Zhu et al., 2008) for reducing the size of the corresponding large matrix. To do this, CF factorizes matrix $Q \in R^{n \times n}$ in (2.3) into a lower triangular matrix, i.e., $Q = LL^T$, where $L \in R^{n \times n}$. ICF is a truncated form of CF, i.e., $Q \approx \hat{L}\hat{L}^T$, where $\hat{L}$ is a $n \times p$ sparse lower triangular matrix close to L, where p is the rank of $\hat{L}$. In ICF approximation, only p column vectors are calculated which makes this approximation quick and economical to compute since $p \ll n$. However, calculating the appropriate column rank value, p, is non-trivial. A lower value of p degrades the accuracy and a higher value of p increases the computational time. In article II, we study the trade-off between the class-prediction accuracy and time efficiency for different p settings and different kernel functions. Furthermore, we study the correlation between the choice of p and the hyperparameters C and the γ value in SVM, with respect to the effect of the Gaussian and Laplacian kernels on the class-prediction accuracy and the training time. In Gaussian and Laplacian kernel functions (see table 2.1), γ determines the influence of training samples on the decision boundary. A large value of γ determines that the decision boundary depends only on the training points close to the boundary and the points far from the boundary do not have an influence on the boundary while a low value of γ shows that even far away training points have the high influence on defining the decision boundary. This means that The γ parameter has an inverse relationship with the radius of influence of support vectors (scikit-learn, 2015). The hyperparameter C makes a balance between the misclassification and the maximum margin. A low value of C makes the decision surface smooth while a large value of C gives freedom to the model to choose more support vectors among the samples that results in more precise and accurate classification of the training samples (scikit-learn, 2015). One of the common ways to find a suitable hyperparameter C and γ is cross-validation (Hsu, C.-C. Chang, C.-J. Lin, et al., 2003), however finding the best value of these parameters are still unclear. In article II, we have improved an IPM-based SVM implementation, PSVM software (E. Y. Chang, 2011), in terms of the training time and the classification accuracy (see empirical study 1).

2.3 ALTERNATING DIRECTION METHOD OF MULTIPLIERS

SVM classification problems can be formulated as the optimization problem

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{N} f_i(\boldsymbol{w}). \tag{2.4}$$

The optimization problem, (2.4), may not be solvable for large-scale problems on a single node. To overcome this hurdle, distributed optimization settings play an important role. To solve the optimization problem in a distributed manner, the training data is stored and processed on a connected network of computational nodes which of all solve the optimization problem collaboratively. In the network, the distributed optimization problem is treated as a consensus optimization in which interconnecting nodes need to reach a consensus to obtain the global optimal. The distributed consensus optimization problem can be solved using an arbitrary connected network of computational nodes (Q. Li et al., 2017). How these nodes connect and communicate with each other impact the performance of the underlying optimization problem.

To solve the optimization problem (2.4) in an aforementioned consensus and distributed

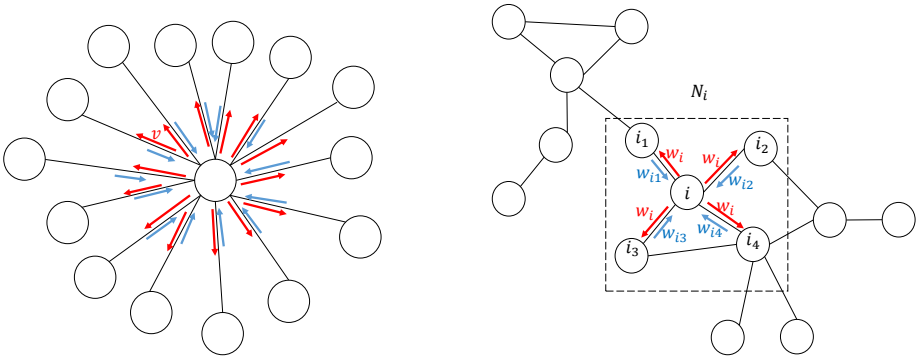


Figure 2.2: Centralized (left) and decentralized (right) settings in ADMM. In the left figure, the outgoing arrows from the middle node show that the global variable is broadcast to all nodes and incoming arrows show that the local variables are sent from all nodes to update the global variable. In the right figure, this is done locally in the neighborhood of each node.

manner, it is re-formalized as the optimization form

$$\min_{\boldsymbol{w}_i} \quad \sum_{i=1}^{N} f_i(\boldsymbol{w}_i) \quad \text{s.t.} \quad \boldsymbol{w}_i = \boldsymbol{v} \quad \text{for} \quad i = 1, 2, ..., N. \tag{2.5}$$

Here, the optimization problem is implemented on a network of N nodes and $\boldsymbol{v}$ is the consensus variable across the nodes, i.e., each local variable $\boldsymbol{w}_i$ is forced to agree with the global consensus variable $\boldsymbol{v}$. A consensus optimization can be solved in a centralized (Boyd et al., 2011) or decentralized (Forero, Cano, and Giannakis, 2010a) settings. This is shown in figure 2.2.

In the centralized settings, the local constraints are forced to satisfy the global constraints. This is shown in equation (2.5) in which each node's local constraint $\boldsymbol{w}_i$ should satisfy the global constraint $\boldsymbol{v}$. In this case, a node failure may hurt the functionality of the parallel implementation. Besides, the communication between all nodes to reach the agreement with the global consensus causes the overhead. To solve this issue, decentralized settings have drawn attention in which computational nodes only communicate with their one-hop neighboring nodes and they only need to agree to the consensus variable in their neighborhood. The decentralized consensus optimization problem transforms (2.5) into form

$$\min_{\boldsymbol{w}_i} \quad \sum_{i=1}^{N} f_i(\boldsymbol{w}_i) \quad \text{s.t.} \quad \boldsymbol{w}_i = \boldsymbol{w}_j, \quad \text{for} \quad i = 1, 2, ..., N, \quad j \in S_i. \tag{2.6}$$

Here, S_i is the one-hop neighborhood of node i and $S_i \subseteq I := \{1, 2, ..., N\}$.

In order to facilitate the decoupling of $\boldsymbol{w}_i$ from its neighboring variable $\boldsymbol{w}_j$ in a decentralized and distributed setting, an auxiliary variable w_{ij} can be introduced which imposes

the consensus between node i and j (Q. Li et al., 2017; Forero, Cano, and Giannakis, 2010a). In this case, the constraints of optimization problem (2.6) are transformed into

$$\min_{\boldsymbol{w}_i, \boldsymbol{w}_{ij}} \quad \sum_{i=1}^{N} f_i(\boldsymbol{w}_i) \quad \text{s.t.} \quad \boldsymbol{w}_i = \boldsymbol{w}_{ij}, \quad \boldsymbol{w}_{ij} = \boldsymbol{w}_j \quad \text{for} \quad i = 1, 2, ..., N, \quad j \in S_i. \tag{2.7}$$

Several distributed optimization methods such as Alternating Direction Method of Multipliers (ADMM) (Forero, Cano, and Giannakis, 2010b), distributed gradient descent (Tao, Wu, and X. Lin, 2014), dual averaging methods (Duchi, Agarwal, and Wainwright, 2012), belief propagation (Bickson, Yom-Tov, and Dolev, 2008), can solve the optimization problem (2.6) (Q. Li et al., 2017). Among all these methods, ADMM has been popular since it performs well in the distributed and particularly decentralized settings (Q. Li et al., 2017). Besides, it has properties such as 1) robustness, 2) scalability, 3) easily distributable and parallelizable, 4) reduced communication overhead, 5) decentralized network operation, 5) convergence guarantees to the centralized settings, and 6) privacy preservation (Forero, Cano, and Giannakis, 2010b). The robustness of ADMM refers often to no requirement of differentiability of the objective function. ADMM solves the optimization problem (2.7) in an iterative manner in which the local primal and Lagrange multiplier of node i are updated using ADMM instructions. To do so, ADMM uses the augmented Lagrangian function

$$\begin{aligned} L(\boldsymbol{w}_i, \boldsymbol{w}_{ij}, \alpha_{ijk}) = \sum_{i=1}^{N} f_i(\boldsymbol{w}_i) + \sum_{i=1}^{N} \sum_{j \in S_i} \boldsymbol{\alpha}_{ij1}^T (\boldsymbol{w}_i - \boldsymbol{w}_{ij}) + \sum_{i=1}^{N} \sum_{j \in S_i} \boldsymbol{\alpha}_{ij2}^T (\boldsymbol{w}_{ij} - \boldsymbol{w}_j) \\ + \frac{1}{2} \sum_{i=1}^{N} \sum_{j \in S_i} \|\boldsymbol{w}_i - \boldsymbol{w}_{ij}\|^2 + \frac{1}{2} \sum_{i=1}^{N} \sum_{j \in S_i} \|\boldsymbol{w}_{ij} - \boldsymbol{w}_j\|^2. \end{aligned} \tag{2.8}$$

Here, $\boldsymbol{\alpha}_{ij1}$ and $\boldsymbol{\alpha}_{ij2}$ are the Lagrange multipliers regarding constraints $\boldsymbol{w}_i = \boldsymbol{w}_{ij}$ and $\boldsymbol{w}_{ij} = \boldsymbol{w}_j$, respectively. ADMM solves the optimization problem (2.7) through the distributed iterations (Forero, Cano, and Giannakis, 2010a)

$$\{\boldsymbol{w}_i(t+1)\} = \arg\min_{\boldsymbol{w}_i} L(\{\boldsymbol{w}_i\}, \{\boldsymbol{w}_{ij}(t)\}, \{\boldsymbol{\alpha}_{ijk}(t)\}), \tag{2.9}$$

$$\{\boldsymbol{w}_{ij}(t+1)\} = \arg\min_{\boldsymbol{w}_{ij}} L(\{\boldsymbol{w}_i(t+1)\}, \{\boldsymbol{w}_{ij}\}, \{\boldsymbol{\alpha}_{ijk}(t)\}), \tag{2.10}$$

$$\boldsymbol{\alpha}_{ij1}(t+1) = \boldsymbol{\alpha}_{ij1}(t) + \eta(\boldsymbol{w}_i(t+1) - \boldsymbol{w}_{ij}(t+1)) \quad \forall i \in I, \forall j \in S_i, \tag{2.11}$$

$$\boldsymbol{\alpha}_{ij2}(t+1) = \boldsymbol{\alpha}_{ij2}(t) + \eta(\boldsymbol{w}_{ij}(t+1) - \boldsymbol{w}_j(t+1)) \quad \forall i \in I, \forall j \in S_i. \tag{2.12}$$

Here, η is ADMM's tuning constant and $\eta > 0$. For detailed information, refer to Boyd et al. (2011) and Forero, Cano, and Giannakis (2010a).

Equation (2.9) can be solved through its dual which forms the dual optimization resulting from SVM and that can use a kernel function for calculating the inner product of the mapping function. In order to update the Lagrangian multiplier $\boldsymbol{\alpha}$, $\boldsymbol{w}_i$ is retrieved from the dual of (2.9) and to reduce the dimensionality of the feature space Forero et al. (Forero, Cano, and Giannakis, 2010a) implement the consensus of the local classifiers on a subspace of reduced rank.

ADMM guarantees the convergence of convex functions using the quadratic penalty term $\|w_{ij} - w_j\|^2$ which is the case in SVMs. The convergence rate of ADMM is $O(1/t)$ for convex functions, where t is the iteration number (B. He and X. Yuan, 2012; Deng et al., 2017). Note in practice the convergence rate of ADMM is still not well-understood (França and Bento, 2017).

2.4 NETWORK TOPOLOGY

Network topology and the connectivity of the underlying graph have an impact on the performance of network-based distributed algorithms in terms of convergence and the number of iterations until convergence (see article III). In article III, we explore the impact of network topology and the connectivity of the underlying graph on the performance of ADMM-based SVMs in terms of convergence. Here, we briefly describe the basics of the network topology. A network of distributed nodes can be represented as an undirected and connected graph, $G(V, E)$ with V as nodes and $E \subseteq (V \times V)$ as the corresponding edges between the nodes with no multiple edges between any two nodes. The network topology of graph G is shown by the corresponding adjacency matrix $A(G) = [a_{ij}]_{n\times n}$. An element of the adjacency matrix, a_{ij}, is 1 if there is an edge between node i and node j and 0 otherwise, i.e.,

$$a_{ij} = \begin{cases} 1 & \text{for} (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

The Laplacian matrix of an adjacency matrix, denoted $L(A) = [l_{ij}]_{n\times n}$, and has -1 for connected pair nodes and the degree of each node on its diagonal, i.e.,

$$l_{ij} = \begin{cases} -1 & \text{for} (i,j) \in E \quad \& \quad i \neq j \\ k_i & \text{for} i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

Here, k_i is the degree of node i.

The connectivity of a graph can be defined by its spectral gap. In order to know the spectral gap of a graph, we need to know the eigenvalues regarding the adjacency matrix of the graph and/or the eigenvalues of the Laplacian matrix of the adjacency matrix. By definition, the adjacency matrix of an undirected graph is symmetric and has real eigenvalues. The eigenvalues of $A(G)$ satisfy

$$k_{max} = \mu_1^{'} \geqslant \mu_2^{'} \geqslant ... \geqslant \mu_n^{'}, \tag{2.15}$$

and the eigenvalues of $L(A)$ satisfy

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n \leq 2k_{max}. \tag{2.16}$$

Here, k_{max} is the largest degree of all nodes.
The first smallest eigenvalue of $L(A)$ is trivial and it is not interesting, i.e., $\lambda_1 = 0$ and corresponds to the first largest and trivial eigenvalue of $A(G)$, i.e., $\mu_1^{'} = k_{max}$. Here, by the largest eigenvalue, we mean the largest absolute value since the eigenvalues of $A(G)$ can be negative, i.e., in some cases $|\mu_n^{'}| = k_{max}$ (Chow et al., 2016).

The spectral gap or algebraic connectivity of a graph is related to the non-trivial eigenvalues of $A(G)$ or $L(A)$, i.e., it relates to $\mu_2 = max\{\mu_2^{'}, |\mu_n^{'}|\}$ in $A(G)$ or λ_2 in $L(A)$ (Donetti, Neri, and Muñoz, 2006). A large spectral gap leads to the better connectivity of the underlying graph. How to design well-connected graphs that have high spectral gaps is an interesting topic discussed in the following section.

2.4.1 EXPANDER GRAPHS

A group of well-studied connected graphs are expander graphs, in which any subset of graph nodes expands through all nodes in a robust manner, i.e., any subset of the graph

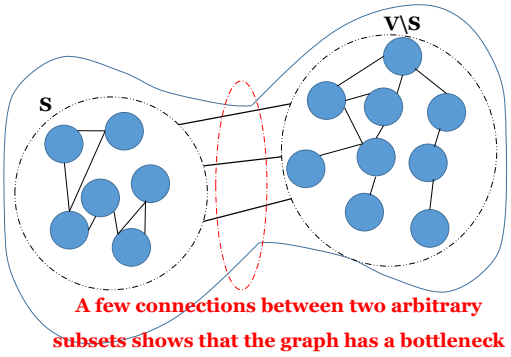


Figure 2.3: Poor expandability of a graph since few edges connect the two sets. Circles are the nodes and lines are the edges in the graph. S and $V \setminus S$ are two large arbitrary sets of nodes.

nodes efficiently connects to many nodes. Properties such as effective communication, high- and well-connectivity, and sparseness make expander graphs good choices for designing efficient networks. Expansion properties of this type of graphs provide good insights about the structure and connectivity of the underlying graph (Malliaros and Megalooikonomou, 2011). The expansion property of an expander graph can be measured by an expansion factor, also known as the Cheeger or isoperimetric constant (Donetti, Neri, and Muñoz, 2006), and it shows that whether the graph has bottlenecks, i.e., whether there are two large subsets of vertices connected by only a few edges. A large Cheeger constant indicates many edges between the two large subsets of vertices. In contrast, a small constant shows that there is a bottleneck between the two subsets of vertices which are connected with only a few edges. Figure 2.3 shows a graph with poor expandability. The Cheeger constant of graph G is denoted $h(G)$ and it is defined as follows,

$$h(G) = min_{S \subseteq V, |S| \leq \frac{|V|}{2}} \frac{|\partial S|}{|S|}. \tag{2.17}$$

Here, $\partial S = \{(e, e^{'}) \in E : e \in S, e^{'} \in V \setminus S\}$. The Cheeger constant is related to the spectral gap by Cheeger inequalities,

$$\frac{\lambda_2}{2} \leq h(G) \leq \sqrt{2d\lambda_2}. \tag{2.18}$$

Here, $G(V, E)$ is a graph with V as nodes and E as the edges between the nodes and λ_2 is the spectral gap. The expansion properties can be enhanced towards increasing the spectral gap. In this regard, d-regular random graphs in which each node is connected to d other nodes are expanders if and only if the corresponding spectral gap is lower bounded (Donetti, Neri, and Muñoz, 2006). In this thesis, we have used this type of expander graphs, i.e., d-regular, to study the impact of network topology on a distributed SVM implementation.

2.5 PARALLEL TOOLS

The most widely used parallel programming tools in machine learning are as follows; shared memory parallelism, distributed HPC architecture, distributed big data architec-

ture, FPGA, and the most popular GPU-based parallelism. Here, we briefly summarized their approaches in machine learning and particularly in SVM.

Shared-memory parallelism. In this type of parallelism, the training task is split across multi-threads which of all access the full data from a single instance in memory (Díaz-Morales, Harold Y Molina-Bulla, and Navia-Vázquez, 2011). This leads to a negligible communication overhead since all computing cores can access the data from the shared memory, however, for large-scale problems, training data might not fit into the limited shared memory. One of the simple and common tools used for a parallel implementation of SVM is OpenMP that supports shared memory multiprocessing. One of the key issues in the shared memory parallelism is to handle the synchronization between threads, i.e., in the cases that concurrent threads conduct the read and the write operations at the same time in the same memory location (Chandra, 2001).

Distributed HPC architecture. In the distributed memory parallelism, data are moved from the address space of one process to the address space of another process using cooperative operations in each process. In Message Passing Interface (MPI) (Nocedal and Wright, 2006), this is done by the send and receive message passing. MPI can be used both in the shared memory and distributed memory parallelism. However the application of MPI in the shared memory parallelism is rare. Unlike the shared memory parallelism, insufficient memory is not an issue even for large-scale problems in the distributed memory parallelism. In this type of parallelism computing nodes with additional memory can be added on demand. One of the key issues in the distributed memory parallelism is to reduce the communication and synchronization overheads (Matloff, 2011). This can be done by the effective input/results communications between the computational nodes that fits better the distributed architecture. Consequently, network topology can impact the performance of the machine learning algorithm in question.

Distributed big data architecture. One of popular distributed big data architecture used in SVM is MapReduce that allows processing large data on the parallel and distributed systems (Dean and Ghemawat, 2008). Map-reduce refers to two tasks, so-called map and reduce functions and both are written by the user. The map function takes the input data and processes the data to produce key/value pairs. Map-reduce library groups the output coming from the map functions and send them to the reduce functions. The reduce function combines key/value pairs with identical keys coming from map functions to create the final results (Dean and Ghemawat, 2008). One of the key issues in map-reduce is the choice of the number of map and reduce functions (W. Guo et al., 2015).

GPU-based parallelism. Today, Graphics Processing Units (GPU) contain immense number of cores that can be used to harness their computational power for solving computation intensive tasks. For instance, NVIDIA has newly announced the NVIDIA TITAN V GPU for a desktop PC which has 5120 CUDA cores (Corporation, 2018). General purpose computing on GPUs is an efficient parallel tool thanks to the characteristics as follows; the computational capabilities, relatively low cost, quick and high-performance improvements of GPUs per year. In the last decade, the general purpose computing power of GPUs has been very popular for parallel implementations of SVM regarding large-scale problems. Some of these parallel implementations have achieved over 1000 times speedups against the sequential implementations (Lu et al., 2014).

FPGA. The computationally expensive tasks in machine learning can be implemented in hardware using FPGAs (Reyna-Rojas et al., 2003; Venkateshan, Patel, and Varghese, 2015; Biasi, Boni, and Zorat, 2005; H. P. Graf, Cadambi, et al., 2009; Papadonikolakis, Bouganis, and Constantinides, 2009). FPGAs are digital integrated circuits that con-

tain programmable blocks of logic and programmable interconnects between the blocks (Maxfield, 2011). One of the reasons for a growing interest in employing dedicated architectures for computing intensive operations is that those architectures can be designed with the aim of reducing power dissipation. For instance, more compact circuits can be built for the fixed-point arithmetic (H. P. Graf, Cadambi, et al., 2009). With this choice of arithmetic, faster computations trade off accuracy, albeit a slight deterioration of the accuracy might be acceptable. One of the key issue regarding FPGA implementations is the limited RAM blocks (Papadonikolakis, Bouganis, and Constantinides, 2009). A dedicated coprocessor consisting of a grid of cores can compute several columns of a large kernel matrix in parallel (Venkateshan, Patel, and Varghese, 2015). Therefore, one may need to perform adjustments and modifications to fit the corresponding algorithm for the hardware characteristics. For instance, one may use strategies to reduce the number of iterations at the expense of more cost per iteration and the cost per iteration can be reduced using FPGA at each iteration. In order to take the advantage of FPGAs, one should think about the availability of the cached kernel values and a fast convergence criterion (Venkateshan, Patel, and Varghese, 2015).

FPGA versus GPU. A comparision between an implementaion of SVM on a didicated high-performance architecture using FPGA and the GPU-based parallelism shows that FPGA outperforms GPUs only for the data that fit in the FPGA's RAM blocks and not otherwise (Papadonikolakis, Bouganis, and Constantinides, 2009). This is due to transfer of the data from the global memory on the device (i.e., GPU) to the shared memory on CPU which causes high overhead for the GPU-based parallelism (Papadonikolakis, Bouganis, and Constantinides, 2009).

CHAPTER 3
RESEARCH METHODOLOGY

This chapter briefly describes the research methodology used in this thesis.

3.1 LITERATURE REVIEW

The foundation of our research has been built based on an extensive reviewing of over 200 relevant articles. The literature review helped me to 1) gain deeper knowledge in the field of my research, 2) know what has already studied in the field, 3) identify important approaches that have potential to reach the peak performance, 4) address the existing gaps, 5) initiate the research ideas later used in this thesis, and 6) come up with constructive recommendations for future works. I categorized the reviewed approaches and tools based on their contribution in the field of parallelism. The summary of the literature review is going to be published as a survey in ACM computing surveys (article I).

Material. The material for the survey was gathered based on reviewing the publications and information in technical books, journals, conference proceedings, technical reports, authentic websites and libraries used for parallel implementations of SVMs. The selection of journal and conference articles was based on searching articles in well-known and authentic databases. Parallel SVM implementations that use parallel algorithmic approaches, parallel models and frameworks were chosen regardless of their applications. In addition to the parallel approaches, the heuristics and strategies that improve the performance of the SVM algorithms with respect to the four focus lines memory, speedup, scalability, and accuracy have been explored (figure 4.2). While the parallel implementations of SVMs using FPGA were briefly mentioned, customized and dedicated hardware for computational purposes have not been the focus of our research in this thesis. A review of publications regarding the sequential implementations of SVMs was excluded from this survey. As a result of literature review, I identified the important elements of efficient parallel implementations of SVM, i.e., algorithmic approaches along with efficient parallel tools and heuristics that have potential to reach pick performance. This is shown in figure 1.1. The parallel SVM algorithms were identified and categorized and within each category, parallel models used for the parallel implementations of SVMs have been reviewed (for more detail see article I).

3.1.1 RELATED RESEARCH

This subsection comprises the related research for each article included in this thesis.
Article I. As an initial step, it is important to identify the appropriate parallel SVM approaches that have potential to obtain peak performance for solving large-scale problems. To do so, we explored surveys which have thoroughly studied the efficiency of existing parallel SVM approaches. For instance, Tyree et al. (Tyree et al., 2014) categorize the approaches used for the parallelization of SVM into the implicit and explicit

parallelization. In the implicit parallelization (Tyree et al., 2014), an algorithm is written as a series of operations that can be computed using highly optimized parallel libraries, e.g., Intel Math Kernel Library (MKL) (2011-2013, 2011-2013) and cuBLAS (Corporation, 2015). In the explicit parallelization (Tyree et al., 2014), programmers parallelize the computationally expensive tasks using parallel programming, e.g., shared memory, distributed memory, and GPUs. The review conducted by Tyree et al. covers most of the existing parallel implementations of SVM and it shows the importance of the implicit parallelism. Despite many advantages of their review, it fails to consider the important parallel algorithmic approaches and parallel tools, e.g., the cascade and map-reduce, and it only considers SMO-type optimization methods. As another example, Lu et al. (Lu et al., 2014) review the mathematical optimization approaches used for accelerating the training process of SVM. The survey only considers the GPU-based parallel implementations of SVM. Unlike the aforementioned approaches, Masih and Tenwani (Masih and Tanwani, 2014) have conducted a generic review of data mining techniques in the distributed settings. In spite of many interesting issues presented in the survey, it lacks to consider the important algorithmic approaches to take the advantages of parallelism and it only mentions very generic advantages of some of the parallel tools without any focus on SVM or other data mining techniques. There are many interesting surveys that have reviewed SVM approaches, i.e., (Byun and Lee, 2002a), however they lack considering the parallelism. To our knowledge, there is no survey that thoroughly reviews the parallel algorithmic approaches along with parallel tools for implementations of SVM. The lack of a thorough survey of parallel SVM approaches including parallel algorithms and parallel tools motivated us to conduct an extensive survey that 1) reviews the efficient parallel approaches in SVM both from parallel algorithmic and parallel tools point of view, and 2) studies various optimization methods, including SMO, IPM, SGM, ASGM, and other methods, used for solving the optimization problem resulting from SVM (the motivation of article I). Note, the Primal Estimated sub-Gradient SOlver (Pegasos) (Shalev-Shwartz et al., 2011) is considered as the state of the art solver for solving the primal optimization resulting from SVM, however, due to limited space in the survey and the sequential implementation of Pegasos we have not mentioned it in article I. Although there are a few numbers of parallel implementations of Pegasos (Do and Tran-Nguyen, 2016; Takác et al., 2013) that we have not mentioned in article I, there are several parallel sub-gradient solvers that have used Pegasos only for the benchmarking purpose and they have not implemented Pegasos in parallel.

Article II. In the last decade, many parallel SVMs have been implemented to take the advantage of parallelism and they have achieved considerable performance in terms of accuracy, speedup, scalability and memory. One of the parallel algorithmic approaches that has drawn attention is IPM that solves the optimization problem resulting from SVM using Cholesky factorization to reduce the problem size. One example of this kind is the parallel SVM developed by Chang et al. (E. Y. Chang, 2011). They improve the scalability of an Interior Point Method (IPM)-based SVM (PSVM) and reduce the memory use. To do this, they have employed a row-based and approximate matrix factorization. Besides, to perform the distributed parallelism, only the important data have been loaded to each computing machine (E. Y. Chang, 2011). Although PSVM has exhibited high-performance using distributed parallelism, it still deals with the communication and synchronization overheads, thereby we further improved PSVM through the investigation of 1) simple changes that can lead to significant improvement in terms of time and the classification accuracy, 2) the impact of the problem size on the important parameters, 3) existence of a threshold between the training time and the number of computing cores. We further studied and improved PSVM in terms of time and accuracy (the motivation of article II).

Article III. As the result of the advance of distributed memory parallelism, network-based algorithms have drawn attention for implementing parallel and large-scale SVMs. In this regard, network topology and the connectivity of the underlying graph are the key issues in the network-based distributed algorithms. They impact the performance of the algorithm in question in terms of convergence and the number of iterations until convergence. One example of this kind is shown by Cao et al. (H. T. Cao et al., 2016) in which the impact of network topology is studied in solving only linear equations. Another example is presented by Francca et al. (França and Bento, 2017) in which the impact of network topology is explored in an ADMM framework with a specific optimization problem not related to SVMs. The importance of network topology motivated us to study the effect of network topology on the performance of a network-based ADMM-based SVM algorithm considering a non-linear classification. Our contribution is summarized in 1) studying the effect of a family of well-known high connected graphs, expander graphs, on the performance of a decentralized ADMM-based SVM algorithm, 2) supplying an implementation making the theoretical advances practically available, and 3) to solve the problem in a parallel setting. Note we have not compared our work with the work conducted by França and Bento, 2017 due to the optimization problem not related to SVM since our main focus is on SVM related algorithms.

3.2 EMPIRICAL STUDIES

In order to make theoretical advances practically available along with analyzing and evaluating the theoretical principles in practice and parallel settings, we have designed empirical studies. This type of studies including quantitative methods is one of the most common research methods used in the field of machine learning. In particular, we have conducted empirical studies to 1) test our hypothesis addressed in research question II and III, 2) derive knowledge from the designed experiments, and 3) analyze and interpret the results from conducted experiments. Two empirical studies are presented in article II and III.

Material. We have designed experiments for two empirical studies in article II and III with datasets gathered by LIBSVM Data (C.-C. Chang and C.-J. Lin, 2011) from several machine learning data repositories such as UCI (Dheeru and Karra Taniskidou, 2017). UCI is one of the most common data repositories used by many academic works in the field of machine learning and it is cited over 1000 times in articles in the field of computer science (Dheeru and Karra Taniskidou, 2017).

3.2.1 EMPIRICAL STUDY 1

In the empirical study presented in article II, we have explored an SVM algorithm using the PSVM software (E. Y. Chang, 2011). PSVM solves the optimization problem resulting from SVM using Interior Point Method (IPM) (E. Y. Chang, 2011) as its inner solver and MPI for the distributed memory parallelism. In the first step, we identified the areas of interest in the algorithm based on their computational time relative to the total calculation time using profiling. The identified areas of interest had potential to benefit from HPC improvements in a way which affect the total calculation time. Minor HPC improvements on heavy computational parts had a large impact on the total computation time for large-scale problems and if the calculation time was not the issue of interest, we improved the classification accuracy for the same calculation time.

Empirical study 1 includes three experiments in order to study the impact of HPC im-

provement on the performance of an SVM algorithm. The objectives of the three experiments are as follows,

- *Experiment 1; Sensitivity of PSVM regarding C and γ in parallel settings.* One of the challenges of SVM implementations using Gaussian and Laplacian kernels is to find appropriate values of parameters, C and γ, that impact the performance in terms of the training time and the classification accuracy (Hsieh, Si, and Dhillon, 2014). Intuitively, γ means how far the influence of training samples reaches the decision boundary (scikit-learn, 2015) (more information in 2.2). One of the common way to find a suitable hyperparameter C and γ is cross-validation (Hsu, C.-C. Chang, C.-J. Lin, et al., 2003), however finding the best value of these parameters are still unclear. Within this context, the aim of this experiment is to study the impact of important parameters, C and γ, on the performance of an SVM algorithm in terms of total training time and at the same time maintaining the target accuracy, i.e., how much the training time varies by changing these parameters. To do this, we chose a range of different values of C and γ and measured the training time and the classification accuracy. Besides we identified the best values of these parameters through an extensive cross-validation via grid search.

- *Experiment 2; The impact of the matrix dimension on the classification accuracy.* An IPM-based SVM can reduce the memory use through approximation of the large matrix resulting from SVM optimization problem (E. Y. Chang, 2011). One example of the approximation methods is the matrix factorization, such as Incomplete Cholesky Factorization (ICF) (E. Y. Chang, 2011). The approximate matrix factorization can lead to reducing the memory use and faster training time, however, it might deteriorate the classification accuracy. In this regard, the size of the approximate matrix is important and has an impact on the accuracy (E. Y. Chang, 2011). This has motivated the design of our second experiment in which we have investigated the impact of the approximate matrix dimension, p, on the performance of the SVM algorithm in terms of the training time and the classification accuracy. We aimed to find an optimal value of p considering a target classification accuracy since finding an appropriate value of p in ICF is non-trivial. To do this, we divided this experiment into sub-experiments as follows, 1) we chose a range of different values of p for the fixed values of C and γ and we measured the class-prediction accuracy, 2) we investigated the performance of PSVM by choosing the best C and γ parameters for each p setting using grid search and we measured the class-prediction accuracy, 3) we studied the impact of p settings on the total training time and the training time on the heavy computational parts of SVM algorithm, and 4) we compared the original PSVM with the improved PSVM and we did a short study of how the changes that we did affected the proportions.

- *Experiment 3; Existence of a threshold.* HPC tools and particularly parallel programming are promising for higher performance, however due to communication overheads choosing the appropriate number of the computational nodes is still non-trivial. Based on the Amdahl's law (Null, Lobur, et al., 2014), the maximum speedup of a program using parallel computing with multiple processor is limited regardless of the number of processors. This has been the motivation for the third experiment in order to study and evaluate the existence of a threshold between the number of computational nodes and the training time. To do this, we have conducted a complexity analysis on the algorithm in question. Besides,

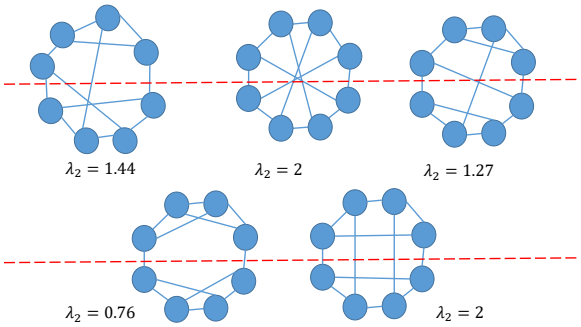


Figure 3.1: Five different 3-regular graphs with 8 nodes that have different connectivity. The dash lines divide the graphs into two subsets with 4 nodes in each subset. The number of edges/links between the two subsets shows the expander property of the graphs.

in this experiment, we studied the relation between the complexities we found in aforementioned experiments and the actual values when running the datasets. We ran the experiment on our improved version of PSVM with 8, 16, 32, 64, 128, and 256 computational nodes and we measured the total training time and the training time regarding the identified computationally expensive parts in the algorithm. For clarity, we measured the proportional training time on the heavy computational parts of SVM algorithm and we also measured the time for communication in those parts.

3.2.2 EMPIRICAL STUDY 2

In the empirical study 2 presented in article III, we have designed experiments to empirically evaluate our hypothesis addressed by research question III. We have studied the impact of network topology on the performance of a distributed consensus ADMM-based SVM in terms of the number of iterations until convergence. To do so, we have implemented several random d-regular graphs with a fixed given number of graph nodes. For instance, we have implemented two groups of random regular expander graphs. The first group, denoted G_1, contains 3-, 5-, 7-, 9-, 11-, 13- and 15-regular graphs with 16 graph nodes and the second group, denoted G_2, contains 10-, 20-, 30-, 40-, 50-, 60-, 70-,80-,90-, and 100-regular graphs with 128 graph nodes. G_1 is designed for training the small datasets and G_2 is designed for training the large datasets.

As mentioned in section 2.4, the connectivity of a graph can be measured by its spectral gap and since several d-regular graphs with the same number of nodes, but different spectral gaps might be generated, we calculated the possible upper and lower bounds of the spectral gap for an arbitrary random d-regular expander graph using the formula given by Joel Friedman (Friedman, 2004). For instance, figure 3.1 shows that one can design several different 3-regular graphs with 8 nodes that have different spectral gaps/connectivity. The formula is given for the second largest eigenvalue of the adjacency matrix $\mu_2 = max\{\acute{\mu_2}, |\acute{\mu_n}|\}$. For simplicity, we adapted the formula for calculating the second smallest eigenvalue of the Lagrangian matrix λ_2. The upper and lower bounds of the spectral gap λ_2 are as follows. For $\epsilon > 0$,

$$d - 2\sqrt{d-1} - \epsilon \leq \lambda_2 \leq d + 2\sqrt{d-1} + \epsilon. \tag{3.1}$$

This holds for every random d-regular graph of size N for sufficiently large Ns.

To take the advantage of expander property, we added an extra condition for implementing the d-regular graphs, i.e., we implemented an efficient type of regular graphs called Ramanujan graphs. A d-regular graph is Ramanujan if and only if $\mu_2 \leq 2\sqrt{d-1}$ holds.

Methods. In this empirical study, we explored the impact of network topology on a consensus ADMM-based SVM using random regular expander graphs. In this algorithm, the classification accuracy is influenced by parameters such as η in ADMM (see chapter 2.3 equations (2.11) and (2.12)) and C and γ in SVM (see chapter 3.2.1). To estimate sufficiently good values of the parameters, we used cross-validation via a grid search. To further improve the classification accuracy, we used normalizing and standardizing scaling techniques. To evaluate the results, we used standard statistics such as true positive/negative rate, positive/negative predictive rate, and accuracy metrics. To measure the impact of expander graph topology, the algorithm trains the datasets for each d-regular expander graph using shared memory parallel programming. Thereafter, the number of iterations and the corresponding elapsed time are measured until convergence. To stabilize our analysis for some of the datasets, we shuffled the training data 10 times. Each shuffled data are trained and the number of iterations and the elapsed time are measured and then we calculated the average value for the final analysis.

In this empirical study, we focused on the binary SVM classifications since a multi-class SVM classification can be transformed into several binary classifications, for example using the one-versus-all technique (see section 2.1).

CHAPTER 4
RESULTS

In this chapter, we briefly describe the results and findings of the research articles contributed to each research question presented in this thesis.

4.1 THE RESULTS OF ARTICLE I

Article I contributes to the objectives addressed in research question 1 which is the foundation of the research in this thesis. The goal of this article was to explore and identify the efficient parallel implementations for improving the performance of SVM algorithms regarding large-scale problems. In this chapter, we briefly summarize the main results of this article.

4.1.1 DOMINANT APPROACH

Here, we briefly describe the dominant and well studied approaches for implementing parallel SVMs.

- The most popular and efficient SVM algorithms used for the parallelization are; decomposition-based algorithms, e.g. Sequential Minimal Optimization (SMO) (Platt, 1998), incremental SVM (Do, Nguyen, and Poulet, 2008), the cascade (H. P. Graf, Cosatto, et al., 2004), IPM (E. Y. Chang, 2011), Kernel-focused SVMs (Díaz-Morales, H. Y. Molina-Bulla, and Navia-Vázquez, 2011), distributed-based SVMs (Shrivastava, Saurabh, and Verma, 2011), and combinational methods (You, Demmel, et al., 2015; Du et al., 2009) (see figure 4.1). The detail of the specification, pros, and cons of each method are described in article I through the online tables.
- Among all aforementioned methods, the most well studied and popular algorithms are as follows, 1) parallel optimizations and particularly Stochastic Gradient Descent (SGD) (Z. A. Zhu et al., 2009), 2) IPM-based, and 3) SMO. Table 4.1 shows the specifications, pros, and cons regarding each method.
- Among all the well studied methods, the decomposition methods and particularly SMO, are the most employed approaches. The reason is that decomposition methods and in the extreme case, the standard SMO use only a small fraction of training data in the working set (see article I, section 5.2). This helps to handle a large amount of training data, however these methods are inherently sequential due to the dependent computation steps. Thus, often they are not the best option for parallelization. In order to decrease the number of dependent steps, parallel decomposition methods use large working sets at the expense of increased cost per step. The cost per iteration can then be reduced by parallelizing each step. The size of the working set has an impact on the training time and accuracy (see

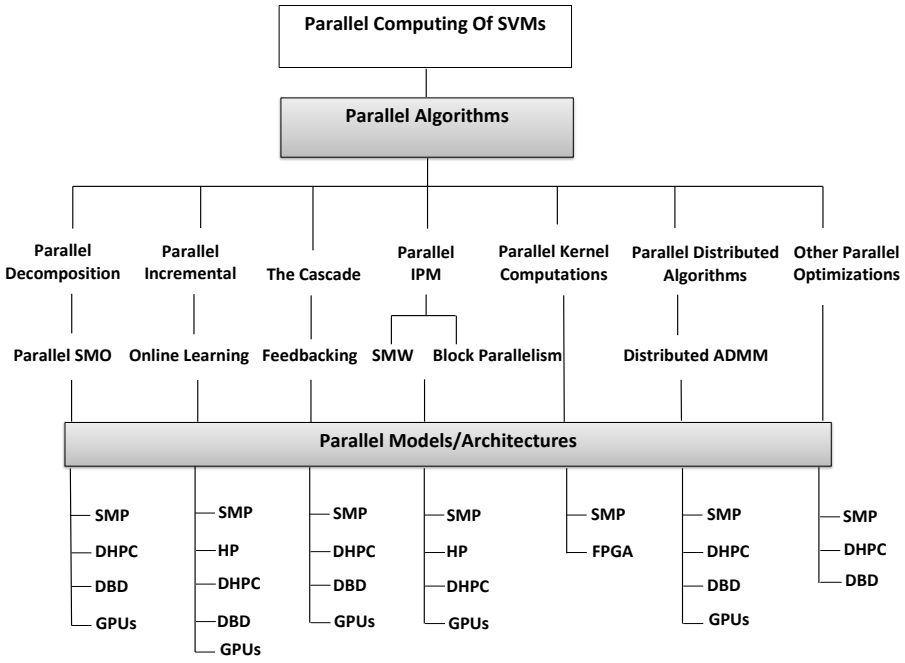


Figure 4.1: Important parallel computing of SVM including algorithms and tools. SMP: Shared Memory Parallelism, DHPC: Distributed HPC Architectures, HP: Hybrid shared-distributed memory Parallelism, GPUs: GPU-based Parallelism, DBD: Distributed Bid Data architectures, and FPGA: Field Programmable Gate Arrays.

Algr	Diff	Type	Pros	Cons
SGD-based SVMs	Focus on primal optimization	Non-linear and multi-class classifications	It is the fastest for linear SVMs on a single machine, easier to be parallelized	A large number of iterations until convergence, accuracy fluctuates
IPM-based SVMs	Focus on dual optimization or dual-primal optimizations	non-linear and binary classifications	Requires few iterations until converges, has the lowest communication cost	The Cholesky factorization lacks theoretical error bound and may be inaccurate for some datasets, slow convergence, difficult to be parallelized
SMO-based SVMs	Focus on dual optimization	Non-linear	Good accuracy, fastest for non-linear SVMs on a single machine	Slow convergence, needs modification to be parallelized

Table 4.1: The comparison of well-studied SVM algorithms.

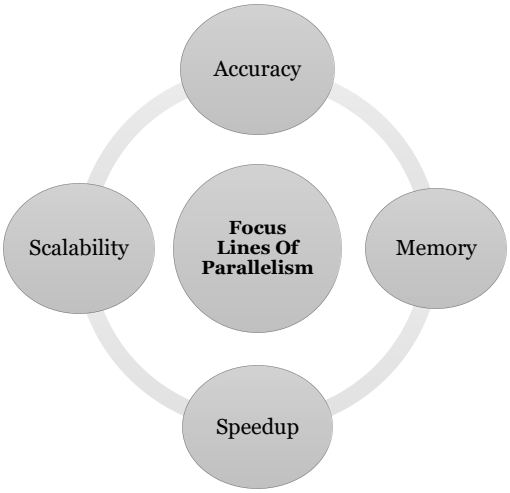


Figure 4.2: Four focus line of parallelism, i.e., parallel implementations have at least one of these focus lines as their goal of parallelism.

section 4.1 in article I). There are different strategies for choosing an appropriate size of the working set, however, there is a lack of agreement on the optimal size. An empirical study of the efficiency of different working set size is needed to better understand the possible improvements in terms of the speed of convergence and the accuracy.

4.1.2 FOUR FOCUS LINES

In article I, memory, speedup, scalability, and accuracy, are identified as the focus lines of parallelism, i.e., all the parallel implementations reviewed in the article had focused on at least one of these elements based on their application scenarios (figure 4.2).

Memory. Parallel approaches, including parallel algorithms and parallel tools, have focused on reducing the memory requirements for solving large-scale problems so that data can fit into the available memory. This is particularly important for the shared memory and GPU-based memory parallelism due to the limited memory or restricted memory access pattern on GPUs. In this regard, the parallel incremental SVMs and parallel IPMs were the most promising approaches. This is because parallel incremental SVMs gradually use a batch of training data at a time and can adjust the increment size as memory allows. Parallel IPMs use approximations to deal with memory restrictions, i.e., they reduce the problem size or dimension using approximation of the kernel matrix that can fit into the available memory.

Speedup. Parallel approaches have focused on improving the performance of SVMs through accelerating the training and/or predicting processes. This has been done through two approaches; One is solving the problem through training a single SVM in which only the computationally expensive tasks are performed in parallel. In the second approach, a large SVM problem is divided into several smaller SVM sub-problems, all of which are performed in parallel. On one hand, solving one single SVM problem leads to higher accuracy since the original optimization problem stays unchanged, however, the speedup deteriorates due to the sequential parts of the algorithm. Another issue is that a single

problem-based algorithm may suffer from memory limitations since the algorithm may require the whole training dataset to proceed. On the other hand, solving multiple SVM sub-problems can address the memory issues, but since the problem is divided into multiple simpler problems, the original problem is changed and this may cause the deterioration of the accuracy. Article I emphasize that the parallel SVM implementations have a tendency towards training multiple SVMs alongside distributed parallelism in which several independent sub-problems are solved in parallel and the computing powers are added on demand.

Scalability. This has been one of the key issues in the implementation of efficient parallel SVMs. Scalability has been considered in terms of the number of computational nodes/cores and the number of samples. Within the first context, the naive increase of the number of computers does not lead to better performance due to communication and synchronization overheads therefore, efficiently handling the overhead is one of the challenges regarding scalability. Within the second context, the increase of the number of samples confronts the memory limitation. While memory issue can be solved by adding more computing power as demanded, this circles back to the overhead problem mentioned earlier. Further to the second context, memory might not be an issue for the increasing number of samples, but the complexity of the problem might grow which leads to larger computationally expensive tasks. This raises the question of how well can a parallel SVM perform for an increasing number of samples? In this regard, there is a lack of empirical studies to supply implementations of large-scale problems considering scalability in both contexts.

Accuracy. Parallel SVMs employ strategies to reduce problem size in order to fit data into the available memory, albeit this comes at the expense of the poor or deterioration of the accuracy. Article I shows a trade-off between the training time and accuracy regarding how much data can be reduced. To improve the accuracy, parallel SVMs often require iteratively updating or re-training the algorithms. The key issue in this regard is that the promising models such as the standard Hadoop implementation of map-reduce has no support for iterative or the sequential natured algorithms. Consequently, implementing a parallel iterative SVM on this efficient framework is still an open question.

4.1.3 PROMISING APPROACHES

Here, we present the parallel approaches that have shown promising performance and efficiency for solving large-scale problems. These approaches are chosen based on the four focus lines described in 4.1.2.

Parallel incremental SVMs. Incremental learning has been one of the efficient and promising approaches to handle limited memory restriction for training large-scale problems on standard workstations. On this subject, the size of increments and the dimension of the input space impact the efficiency of the parallel incremental SVMs. Reviewing literature shows that the high dimensionality of input space is a hurdle for this technique, albeit the incremental SVMs use dimension reduction techniques to overcome the hurdle and harness the power of this technique for training the large-scale problems.

ADMM. It is one of the successful distributed methods since it is robust, distributedly parallelizable, and it has convergence guarantees. In decentralized ADMM through a network, distributed agents/nodes with the knowledge of local data solve local optimization problems and only communicate with their neighboring nodes with the common goal of reaching consensus. Another advantage of ADMM framework is the adaptability to other machine learning algorithms alongside SVM. ADMM has shown a promising

performance considering all the four focus lines of parallelism for solving large-scale problems. However, it may suffer from slow convergence in specific circumstances. Besides, the convergence speed of ADMM concerning graph topology is poorly understood.

Map-reduce. Parallel SVM implementations using map-reduce scale well to a large number of machines. In spite of that, they may not perform well for sequential or iterative nature algorithms. To solve this problem, an extension of the Hadoop (ApacheSoftwareFoundation, 2014) implementation of map-reduce called Twister is created. Notwithstanding the support for iterative methods in Twister (Ekanayake et al., 2010), only a few SVM algorithms use it for the parallelization. One of the key issues in map-reduce-based parallel SVM is the impact of the number of mappers and reducers on the performance of the parallel implementations.

Combinational methods. Parallel incremental learning, map-reduce, the cascade and distributed approaches, particularly ADMM are promising to reduce memory requirements and accelerate the training process for large-scale problems. The results derived from article I confirm that combination of these approaches lead to higher performance since they complement each other and overcome the weaknesses.

Network architecture and topology. These play important roles in the efficiency of network-based parallel implementations. The dominant architecture in the literature is mostly based on centralized networks in which a master node has the duty of distributing data among distributed nodes and also gathering local results to obtain the final one. A drawback regarding a centralized network is that the communication and synchronization can be a hurdle for efficient parallel implementations. In contrast, decentralized computing and peer-to-peer (P2P) computing models exhibit accelerated communication and reduced overhead. Network topology and the connectivity of the underlying graph have also impact on the performance of network-based distributed algorithms in terms of convergence and the number of iterations until convergence. This has not been investigated and been poorly understood in the parallel network-based SVMs.

4.1.4 RECOMMENDATIONS OF POTENTIAL AVENUES FOR FUTURE RESEARCH TOWARDS DESIGNING EFFICIENT FRAMEWORKS

Article I outlines that still, there is space for further improvement regarding computation time, accuracy, scalability and memory issues due to the immense and increasing size of real-world data requiring judicious choice for end users. Having the existing challenges and trade-offs in mind, designers have a great deal of flexibility in designing and implementing SVMs by taking their goals of parallelism into the consideration. For instance, considerable speedups can be achieved by a slight deterioration of the classification accuracy which might be acceptable in some applications. One important point is to identify the efficient parallel tools, heuristics, and strategies that fit the characteristics of SVM algorithms in mind, otherwise one should be prepared for modifications and manipulations of the corresponding algorithms or strategies to take the maximum advantages of parallelism. In the current trend of parallelizing SVMs, it seems that the parallel implementations of SVM solvers are still not sufficient to handle large-scale problems and their challenges open up directions for future work. In this regard, article I provides several recommendations towards implementing efficient parallel SVMs. These recommendations are briefly summarised as follows,

Use of the four identified focus lines. Memory, speedup, accuracy, and scalability has been identified as the four main focus lines of parallel implementations in this thesis, i.e., parallel approaches improve the efficiency of the algorithm in mind in terms of one

or more of these four focus lines. They are not independent and there are trade-offs between them. Based on the line of reasoning in article I, in order to implement efficient parallel SVM, developing algorithms that are capable of addressing all the focus lines leads to a robust implementation.

Reproducibility. Another aspect to consider is improving the transparency of the parallel implementations by focusing on the reproducibility, replicability, and reliability of the results. This gives the possibility of reproducing the results and improving the existing weakness in different application scenarios and problem settings. Besides, it makes it possible to conduct a fair comparison of similar empirical approaches and identify the potential for future research.

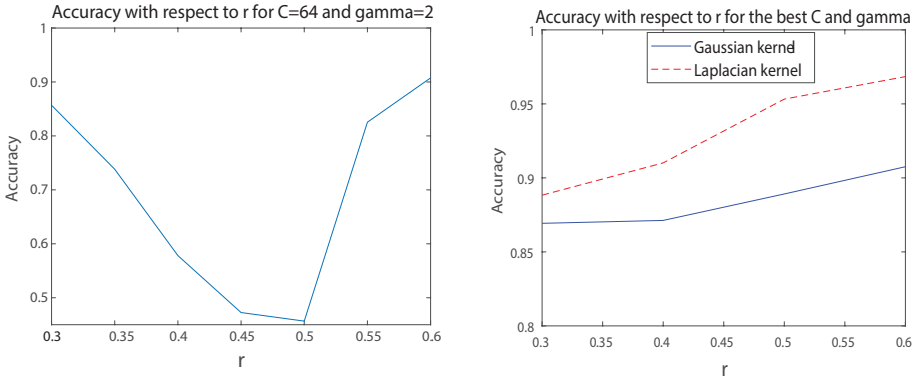
Use of combinational approaches. Combinational approaches have been successful in handling large-scale problems since the matching techniques can complement each other and improve the possible weaknesses. To take the maximum possible advantages of parallelism, a future direction likely to be successful would be the development of an integrated framework that gives the possibility of combining several complementing techniques. Suggestions such as combining incremental SVM with ADMM-based SVMs and combining ADMM-based with map-reduce for solving multi-class and non-linear classification are given in article I.

Use of the available modern technologies. Modern processors already contain technologies that designers of parallel algorithms can take advantage of. These technologies are employed by high-performance and parallel libraries and software, e.g., MKL (2011-2013, 2011-2013). Indirectly, using these tools helps the users to improve the performance of developed algorithms. In addition to that, one can directly use modern processor technologies in SVM algorithms to further improve the performance. However, a restrictive design of algorithms is a hurdle and may not allow using these technologies to the fullest extent. Article II confirms that one should always take the use of the already developed HPC tools into the consideration to avoid developing restrictive parallel SVMs. A future research direction can be to implement non-restrictive parallel SVMs that allow using the available modern processor technologies, e.g., SIMD instructions such as SSE and AVX.

Network architecture and topology. Solving large-scale problems requires a large amount of memory which can be provided by adding more computing resources from different physical locations. The majority of the parallel SVMs has their main focus on a centralized computing in which the distributed computing nodes communicate more or less regularly with a master to obtain the final results. This can cause overheads, thus a hurdle for effective parallel implementations. An interesting future research direction is to develop algorithms that are suitable for decentralized computing that take the advantage of distributed computing resources without major overhead or loss of performance. Another direction to reduce the communication overhead is to design networks with good connectivity.

4.2 THE RESULTS OF ARTICLE II

In order to reach the objectives of this thesis in research question 2, we have conducted the empirical study 1 and the experiments described in 3.2.1. Article II investigates that how simple changes to SVM implementations can have significant improvement in parallel settings. In article II, we improved the PSVM implementation concerning the structure, memory allocation, de-allocation and parallelism point of view. In addition, by finding appropriate values of parameters, we got 20% improvement on the calculation



(a) : Accuracy for the fixed C and Gamma (b) : Accuracy for the best C and Gamma

Figure 4.3: The classification accuracy with respect to different column size for fixed values of C and γ (4.3a) and for the best values of C and γ (4.3b). Here, $r = \frac{log(p)}{log(n)}$, where p is the column size and n is the number of training samples for `webspam` dataset with 300000 samples and 254 features.

of one of the computationally expensive tasks. In continuation of this chapter, we briefly describe the details of the findings in article II.

Experiment 1; Sensitivity of PSVM regarding C and γ. To reach a target classification accuracy, the total training time of the PSVM was not sensitive to changing the value of C and γ. Besides, no interesting trend was found in the training time by changing these parameters, albeit this experiment gives better insight about experiment 2.

Experiment 2; The impact of the approximate matrix dimension, p, on the accuracy. In experiment 2, we show the impact of changing the column size of the approximated matrix on the accuracy using a logarithmic scale, i.e., we use $r = \frac{log(p)}{log(n)}$, where p is the column size and n is the number of training samples. Experiment 2 showed that changing p affected the choice of important parameters, C and γ in SVM, in which choosing the best values of the parameters for special column size p was not necessarily the best value for another p. For instance in Figure 4.3a, the best C and γ ($C = 64$ and $\gamma = 2$) for $r = 0.6$ are not good values for $r = 0.5$. In the original software, a fixed number of columns related to the number of training samples is recommended, i.e., $p = \sqrt{n}$, where n is the number of training samples. In contrast, article II points out that the recommended size was not optimal for different cases. Figure4.3a shows an example of poor accuracy for choosing the recommended size ($p = \sqrt{n}$ or $r = 0.5$) for the fixed values of C and γ. This is also an example of unstable accuracy by increasing the number of columns for the fixed values of C and γ. In Figure 4.3b, we observed an improvement in the classification accuracy while increasing the number of columns of the approximation matrix and choosing the best values of C and γ for each value of r. Article II mentions that replacing C and γ with the best C and γ for each column size led to the stable classification accuracy. As another observation, figure 4.4 shows that the best C and γ stayed close when the value of p was changed. The result of this experiments confirms the performance improvement of the improved PSVM versus the original version. Figure 4.5 shows this comparison for different p size and shows 4 times improvement of performance by our modification on PSVM for a large value of p. Even for the smaller value of p we got 20% improvement on the calculation of one of the computationally expensive task, denoted E.

Experiment 3; Existence of a threshold. In this experiment, we exhibited the existence

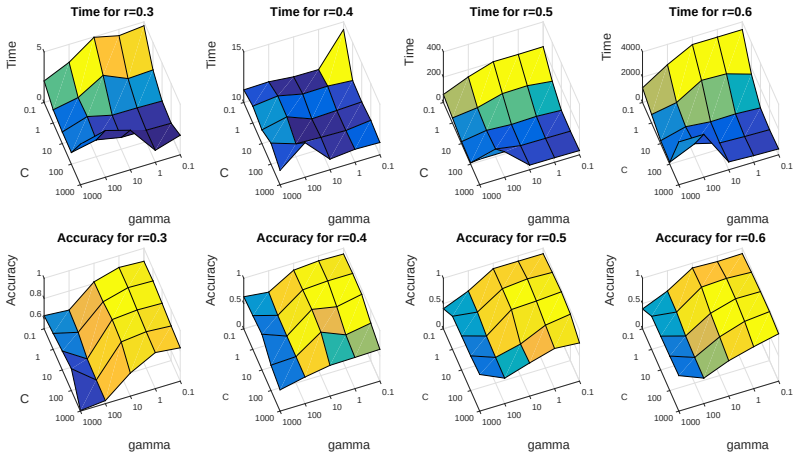


Figure 4.4: The training time and accuracy with respect to different column numbers, p for `cod-rna` dataset considering different C and γ settings for Laplacian kernel. Here, $r = \frac{log(p)}{log(n)}$, where p is the column size and n is the number of training samples.

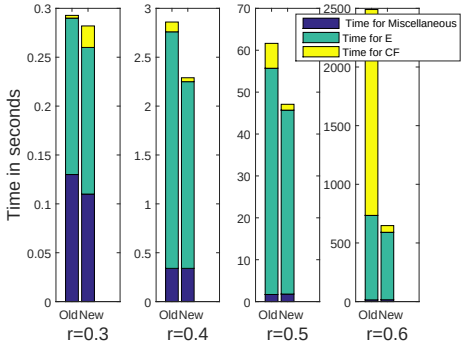


Figure 4.5: The comparison between the original PSVM (old) and the improved PSVM (new) with respect to column size of the approximation matrix, r. Here, $r = \frac{log(p)}{log(n)}$, where p is the column size and n is the number of training samples. E and CF are the computationally expensive tasks in IPM.

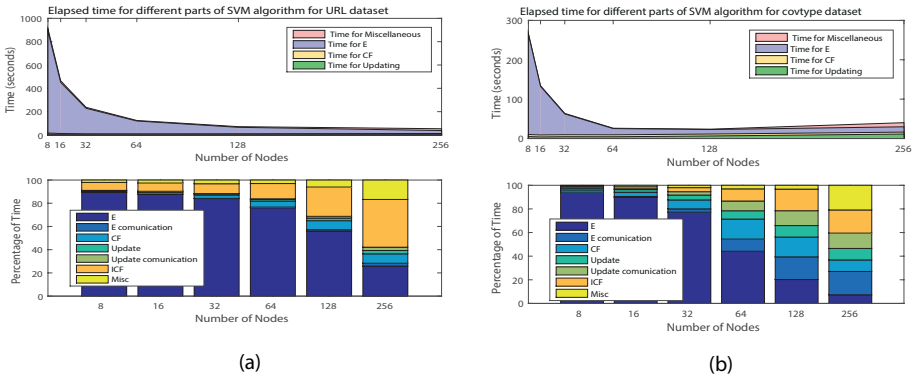
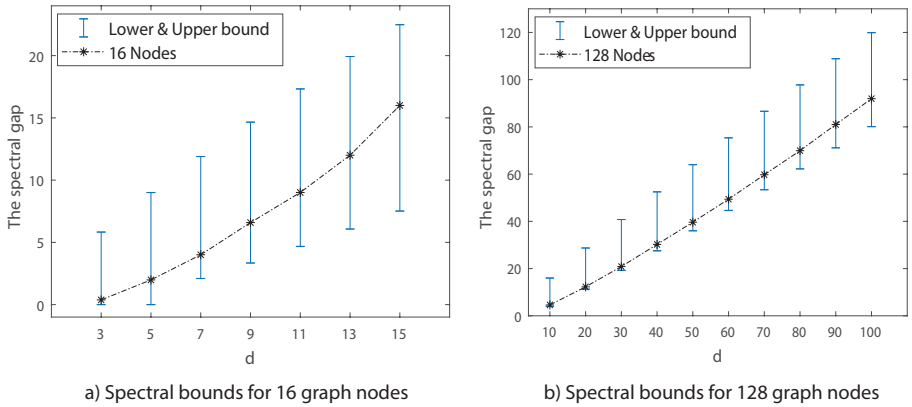


Figure 4.6: The elapsed time for different parts of SVM algorithm with respect to the number of nodes, a) URL dataset with 2150000 samples and 3231961 features, b) covtype dataset with 500000 samples and 54 features. E, CF, $update$ and ICF are the computationally expensive tasks in IPM and $E\ communication$ and $update\ communication$ are the communication time that is required to conduct E calculation and updating the variables in each iteration.

of a threshold between the training time and the number of computing cores using the complexity analysis. Existence of such threshold is important since it can suggest a correct number of computing cores to be used without wasting the computational power due to overhead. In addition, this experiment investigates how training time regarding computationally expensive tasks in PSVM changes while increasing the number of nodes. In Figure 4.6, the upper sub-plots indicate the decrease of the training time while increasing the number of nodes. This trend stops for the number of node more than 128 in figure 4.6 (*a*) and 64 in figure 4.6 (*b*). Thereafter, increasing the number of nodes shows no remarkable improvement in (*a*) and even the increase of the time in (*b*). This is due to the overhead. The lower sub-plots give a better insight about the proportion of the training time for the most computationally expensive tasks in PSVM.

4.3 THE RESULTS OF ARTICLE III

Article III is based on the empirical study 2 and it contributes to answering research question 3 with the aim of investigating the impacts of network topology on the performance of SVM algorithms in terms of convergence. Networks with good connectivity perform well in terms of convergence (H. T. Cao et al., 2016; França and Bento, 2017). In this regard, complete graphs are well-known for their connectivity, however, high connectivity does not come cheap in particular using distributed memory parallelism, i.e., all nodes need to communicate with each other and this may increase the number of iterations until convergence and increase the communication complexity. Random regular expander graphs are good sparse approximations of complete graphs in which good connectivity alongside efficient communication between nodes is inherited. The result of the empirical study in article III confirms that the performance of a network-based ADMM implementation of SVM is improved based on the efficient connection on the neighboring nodes using expander graphs. Furthermore, it confirms that the graphs with good expander property have faster communication between the nodes. We have used Newton-based algorithm from NLopt optimization package (Johnson, 2008) to solve the optimization problem resulting from ADMM in (2.9). The details of the find-



a) Spectral bounds for 16 graph nodes

b) Spectral bounds for 128 graph nodes

Figure 4.7: The upper and lower bounds of the spectral gap for random d-regular expander graphs. a) Graphs with 16 nodes and $d = \{3, 5, 7, 9, 11, 13, 15\}$, b) Graphs with 128 nodes and $d = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

ings of this article are as follows,

- The connectivity of a regular graph increases as the degree of the graph and the corresponding spectral gap become larger. The bound of the spectral gap expands as the degree of the graph and consequently the connectivity increases for a fixed number of nodes. Albeit, increasing the degree of a graph will not necessarily lead to better connectivity if its spectral gap does not become larger. Figure 4.7 shows the upper and lower bounds of the spectral gap for 16 nodes in (*a*) and 128 nodes in (*b*).

  Miller et al. (Miller, Novikoff, and Sabelli, 2008) conjectured that the probability that a random d-regular graph is Ramanujan is 27% as the number of graph nodes grows. Hooray et al. (Hoory, Linial, and Wigderson, 2006) mention that this probability may get larger than 50%. For a fixed number of graph nodes and considering the randomizing, e.g., 16 nodes, we can with probability 27% get a 11-regular graph that is Ramanujan and in the same way we can with probability 73% get a 13-regular graph that is not Ramanujan. Hence, there is a small probability that a lower degree graph may have a higher connectivity.

- Graphs with higher connectivity exhibit accelerated convergence, i.e., the number of iterations decreases as the connectivity of the graph increases. This is shown in figure 4.8 (top). Furthermore, the computing time of the solver decreases as the connectivity of the graph increases. This is shown in the lower subplots of the same figure. Note the decrease in the number of iterations saturates or becomes negligible as the degree of the graphs is close to the number of graph nodes.

- Shuffling increases the classification accuracy and leads to a stable classifier while keeping the trend of decreasing the number of iterations and time for d-regular graphs. This is shown in figure 4.9.

- The findings in article III is consistent with other similar approaches in (H. T. Cao et al., 2016; França and Bento, 2017), whose theoretical analysis suggested

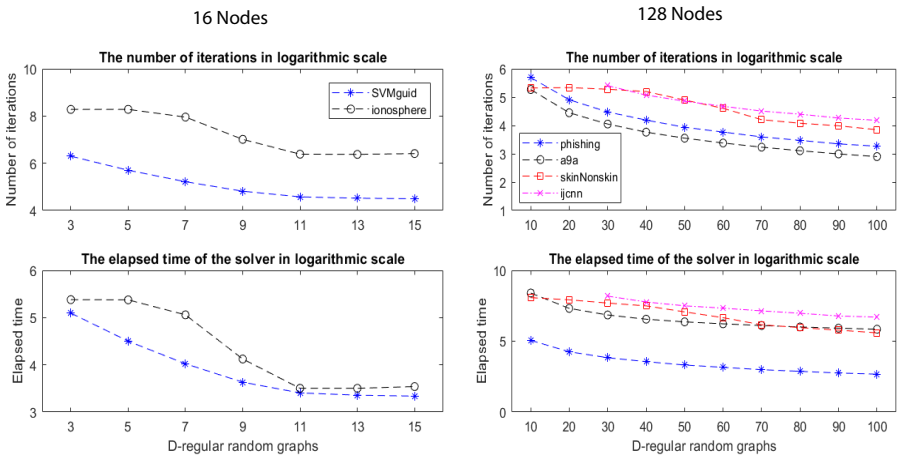


Figure 4.8: Impact of d-regular graphs on the number of iterations and time.

high mean degree for graphs. Despite the fact that those approaches were not tested for a network-based ADMM nor for nonlinear systems.

- The impact of many communicating neighboring nodes is minor in the shared memory parallelism. In contrast, in the distributed memory parallelism, the communication between many nodes reaching the consensus might cause communication overhead. Thereby, increasing the degree of graphs close to complete graphs may not be efficient as in the shared memory parallelism.

In addition to the impact of network topology on the performance of a network-based SVM, several factors showed their importance for the classification accuracy in article III. Some of them are briefly mentioned here.

- *Class imbalance.* To keep a balance between classes while keeping the trend of decreasing the number of iterations and time, we randomly shuffled the data and consequently it exhibited higher classification accuracy versus unshuffled data. Besides the advantages of shuffling, it addresses an issue regarding privacy, i.e., networks that supply their data from distributed sources may not allow combining and shuffling the data due to privacy issues.
- *Data cleansing*. The finding in article III confirms that simply removing the missing and corrupted data exhibits higher accuracy. This shows the importance of preprocessing of data before conducting the training process.

Finally, based on the line of reasoning in article III, we provided several suggestions for future work.

- Find an appropriate degree of regular graphs concerning the number of graph nodes in which the degree should be sufficiently high that leads to effective communication and sufficiently low that leads to good connectivity.
- Investigate the impact of expander graphs on the performance of the algorithm using distributed parallelism.

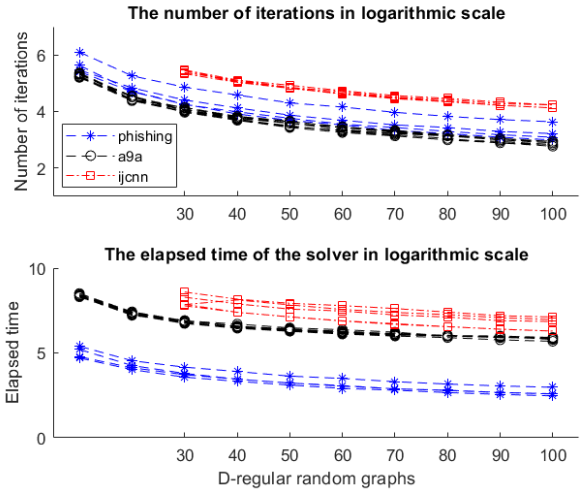


Figure 4.9: Several rounds of shuffling data. Shuffling increases the classification accuracy while keeping the trend of decreasing the number of iterations and time for d-regular graphs.

4.4 RESEARCH QUESTION 4

In this section, we present recommendations as a guideline for the development of an efficient parallel implementation of SVM. These recommendations are based on the findings from all the research articles included in this thesis and they contribute to answering research question 4. Based on the line of reasoning in article I, II, and III, we believe that these recommendations make developing an efficient framework possible and give judicious choice for scientists as well as developers to balance the existing trade-offs.

In article I, we conducted literature review and identified three general phases used for learning large-scale problems in machine learning. These tasks may not be explicitly mentioned in the literature, but we have identified them being used. These phases are pre-processing of data, main processing of training, and post-processing of results.

Pre-processing phase. This phase includes initial tasks that are conducted on data before the training of an algorithm starts. These tasks aim to cleanse, integrate, transform, normalize and/or reduce data (Garcı´a, Luengo, and Herrera, 2015). Feature selection, instance selection, data compression, filtering, shuffling, scaling, dimension reduction, initial clustering of data are examples of pre-processing that are used in the literature regarding SVM. Strategies used in this phase can solve issues regarding class imbalance, missing, noisy, redundant, and inconsistent data.

Main processing phase. This phase includes the actual training of the algorithm and it is the main contribution in most of the research works, i.e., they mainly focus on improving the performance of the algorithm in mind in the training process, nevertheless pre- and post-processing phases alongside main processing play an important role for efficient implementations of SVMs in practice regarding large-scale problems. In the main processing phase, SVM trains data coming from the pre-processing phase. In the main processing phase, we provide the following recommendations to improve the performance of a parallel SVM implementation.

- *Framework foundation.* Distributed optimization frameworks such as ADMM

give a possibility of divining a large training problem into independent smaller sub-problems each of which has its independent objective function and constraints. These frameworks perform independent of their inner optimization solver, i.e., one can use a variety of different optimization solvers to solve the sub-problems. ADMM can be adapted to other machine learning algorithms alongside SVM. To improve the accuracy and reach the global optimal, a consensus-based optimization can be added to the framework (see article I and III).

- *Scalability in terms of the number of computing nodes*. The advantage of a distributed optimization as the foundation of our recommended framework is that it can employ computing power on demand, i.e., computing nodes can be added in order to perform computationally expensive tasks in a distributed manner. As recommended in article II, it is helpful to find a theoretical bound for the scalability since it suggests the optimal number of cores to take the maximum possible advantage of parallelism considering overhead. In article II, we have shown that such theoretical bound exists using the complexity analysis.
- *Scalability in terms of the number of samples*. To handle immense amount of data, as article I suggests, incremental learning and online learning can be added to our recommended framework. This can also be efficient in the case of learning of streaming data.
- *Decentralized learning*. Employing peer-to-peer networks or consensus-based optimizations in which nodes communicating with only neighboring nodes reduce overhead. Furthermore, this assures that the algorithm remains functional in the case of node failure.
- *Parallel computation*. Map-reduce improves the performance of a parallel SVM in terms of time when it is performed in a distributed manner. The framework can employ map-reduce to perform ADMM in parallel. In this regard, the automated and controlled number of mappers and reducers can be added toward further improvement of map-reduce.
- *Network topology*. Expander graphs and particularly regular graphs improve the quality of the communication between the neighboring nodes. As article III outlines, this type of graphs offers an efficient communication pattern for the nodes employed in the framework.

Post-processing. This phase includes the tasks that are conducted after training the algorithm using machine learning techniques. These tasks aim to interpret, explain and evaluate the results acquired from the training. Examples of post-processing are 1) strategies for testing and evaluating the results, 2) strategies for improving the interpretation of the results, 3) methods for making the results understandable for humans, and 4) strategies to improve the quality of the prediction.

CHAPTER 5
CONCLUDING REMARKS AND FUTURE WORK

The objectives of this thesis have been formalized in the form of four research questions which of all are answered through three research articles. In this section, we summarize each research article and the drawn corresponding conclusion.

5.1 SUMMARY AND CONCLUSION

To answer the first research question, article I has explored the important factors on the efficiency of parallel SVM implementations regarding large-scale problems. This led to identifying important steps towards developing efficient parallel implementations such as the choice of 1) underlying SVM algorithm, 2) HPC tools and parallel programming, and 3) additional heuristics to overcome issues regarding big data (figure 1.1). In addition, we have identified four focus lines of parallelism, memory, speedup, scalability, and accuracy, each of which has given a better understanding of the objectives of parallelism in different application scenarios. Our extensive exploration of the state of the art parallel implementations and their pros and cons helped to find the efficient parallel algorithms, HPC tools and heuristics that fit better the parallel settings. As the result of that, we provided clear suggestions for several application scenarios.

Aligned with the goal of this thesis in the second research question, article II contributes to the exploration of simple changes leading to significant improvement in parallel settings. In this article, it is shown that a restrictive design of algorithms is a hurdle and may not allow using the modern technologies to the fullest extent. The article further investigates the impact of the problem dimension on the choice of important parameters. In addition to that, the existence of a threshold between the training time and the number of computing cores is elaborated. A strength of the existence of the aforementioned threshold is to help to employ the correct number of machines without wasting the computational power.

Towards the objective in the third research question, article III contributes to the investigation of the impact of network topology on the performance of a parallel SVM algorithm along with supplying an implementation making theoretical advances practically available. This led to the conclusion that the networks with good connectivity, large spectral gap, and higher degree exhibits accelerated convergence.

All the findings of the research articles presented in this thesis contribute to the objective of research question 4. This led to several recommendations towards developing an efficient parallel framework for SVMs considering three key steps, namely pre-processing of data, main processing of training and post-processing of results.

As a final note, we believe that our contribution in this thesis 1) clarifies the existing challenges, 2) addresses trade-offs, and 3) suggests a roadmap/guideline, all of which facilitate judicious decision for the end users to have a great deal of flexibility in designing

and implementing parallel SVMs by taking their goals of parallelism into the consideration. Besides, our recommendations can 1) make it possible to develop a framework that gathers all the efficient parallel approaches under the same roof, and 2) suggest potential avenues for future research.

5.2 THE CONTRIBUTION OF THIS THESIS IN THE FIELD OF MACHINE LEARNING

The research presented in this thesis is the marriage between the theoretical and practical advances regarding parallel SVMs for large-scale problems. The combination of the reviewed state of the art research works, conducted empirical studies, and designed experiments in this thesis widen the understanding of which and how parallel approaches lead to higher performance. Based on the findings in research question 1, 2, 3, and 4, we briefly summarize the thesis contributions to the field of machine learning.

5.2.1 RESEARCH QUESTION 1

Through the extensive survey of previous research works, this thesis contributes to understanding the current state of knowledge in the field, along with identifying the state of the art research works. Moreover, through a comparative study, it provides new insights into the efficiency of parallel approaches, their challenges, and trade-offs. Having the existing challenges and trade-offs in mind, it provides a great deal of flexibility for designers in implementing efficient SVMs by taking their goal of parallelism into the consideration. The knowledge obtained to answer research question 1, highlights and identifies the key focus line of parallelism in the literature which has not been explicitly done before. In regard to novelty, to our knowledge, it is the first extensive survey that thoroughly reviews the parallel algorithmic approaches along with the parallel tools for the implementations of SVM. It also offers new ideas and clear suggestions of possible avenues for future research. This knowledge contributes to practice since it facilitates judicious decisions for scientists as well as developers to design efficient parallel SVMs and make theoretical advances practically available for different application scenarios.

5.2.2 RESEARCH QUESTION 2

Through the designed empirical study 1, this thesis contributes to the field of practice by presenting the existence of a new threshold between the number of computational cores and the training time. In practice, this is important since it gives a possibility for correctly using the available resources without wasting the computational power, i.e., it gives a new insight and an idea of an optimal number of computational cores that can be used to take the advantage of parallelism. In addition, the knowledge acquired from this empirical study suggests that how simple changes lead to higher performance in practice. Another novelty of this research is that it shows how changing the problem size impact the optimal value of important parameters, whereas the original software suggests a fixed problem size. This contributes to the field of practice, as it relates to adjusting the problem size based on the available memory and the target classification accuracy, all of which are important in the practical implementations.

5.2.3 RESEARCH QUESTION 3

With the help of empirical study 2, the contribution of this thesis is the marriage between the theoretical and practical advances regarding a network-based SVM. From the theoretical point of view, we reviewed the efficiency of the expander graphs and their connectivity. From the practical point of view, we implemented one of the well-known expander graphs in order to study the impact of network topology in practice. In regard to novelty, we implemented a parallel consensus ADMM-based nonlinear SVM in a new setting, i.e., employing expander graphs for the communication between the nodes. This is an important contribution to the field of practice since designing a network with efficient communication pattern, high and good connectivity leads to higher performance, i.e., the same number of computational nodes with efficient communication between the graph nodes exhibits a faster convergence.

5.2.4 RESEARCH QUESTION 4

The fourth contribution of this thesis is the acquired recommendations for developing an efficient framework for the parallel implementations of SVM. This knowledge contributes to the field of practice since it provides a possibility of developing a framework that integrates all the efficient approaches under the same roof in order to take the maximum possible advantage of parallelism. To do so, key steps are introduced and the possible recommendations are provided. This research highlights the importance of the distributed computing for not only SVMs also for other machine learning techniques. Besides, it provides new ideas for combining several efficient methods to improve the performance of SVMs regarding large-scale problems.

5.3 FUTURE WORK

In the era of digitalization, the parallel computing of SVM is becoming a necessity for improving the performance of SVM for big data. In the current trend of parallelization, it seems that the parallel implementations of machine learning solvers are still not sufficient to handle large-scale problems and their challenges open up directions for future work. In this section, we suggest several possible avenues for the future research direction.

- *A distributed framework.* The results from the articles included in this thesis verify a tendency towards training large-scale SVM problems using distributed frameworks. A future direction can be to design an efficient and distributed framework such as ADMM that can integrate many of the efficient approaches under the same roof considering the pre-processing, main processing and post-processing phases. Another future direction is to use ADMM for proximal methods, non-smooth techniques and non-convex optimizations.

- *Decentralized settings.* Many of the existing distributed frameworks are performed in a centralized setting in which one central node has the duty of distributing data among other nodes or accumulating results from the nodes. In order to reduce the concerning overhead caused by the centralized communication setting, a future direction can be to further investigate decentralized settings in which each node only communicates with the neighboring peers using efficient network topology.

- *Efficient network topology, improving the convergence of ADMM and community selection.* The effect of network topology on the performance of distributed network-based SVM algorithms has not sufficiently explored. A future research direction can be to explore the effect of different network topologies on the performance of network-based distributed SVM algorithms. For instance, by adding or removing links between graph nodes, one can create an adaptive network topology that may result in fast convergence of network-based distributed algorithms. Besides the adaptive connection between graph nodes, a future research direction can be further investigation of community selection in which the graph nodes that play important roles for efficient communication are identified.

- *Mapping distributed algorithms on distributed physical computational nodes/-cores.* To perform a distributed optimization algorithm in parallel, the network topology used in the algorithm can be mapped to the distributed physical computational nodes/cores, albeit it will not improve the convergence speed, the computation speed can be significantly improved. As article III suggests, a future direction can be conducting the aforementioned mapping of a distributed optimization to a distributed memory parallelism for solving large-scale problems.

- *Approximation methods.* In distributed parallelism, as article I and II highlight, strategies such as approximation methods are used to further reduce the problem size, memory requirement and the computational complexity. For instance, Cholesky factorization, subsampling, approximated kernel matrix, and incremental training are used to accelerate the computationally expensive tasks. A future research direction can be to investigate the effect of different approximation methods on the performance of a decentralized and consensus-based ADMM implementation of SVM and identify the possible trade-offs.

- *A threshold for the number of computational cores.* In article II, we showed the existence of a threshold for the number of cores using the complexity analysis. A future research direction can be to theoretically prove it. This helps to determine the correct number of computational cores that can be used without wasting the available computational power due to overheads.

- *Pre-processing.* Besides the importance of main processing phase, pre-processing, i.e., initial tasks before training, can also affect the performance of an SVM algorithm. A further investigation can be done to reduce the problem size in the pre-processing. Most of the strategies to reduce problem size are performed in a sequential manner. As article I and II suggest, a future research direction can be exploring strategies such as data cleansing, prototype and feature selection in parallel settings in distributed decentralized-based SVM algorithms.

- *Problem characteristics.* In the articles included in this thesis, we have mentioned several strategies to handle large training samples and improve the classification accuracy. While there are many strategies to handle big data, there is no clear suggestion/roadmap for training problems with different characteristics of training samples such as,

  - The number of samples is significantly larger than the number of features,
  - The number of samples is significantly smaller than the number of features,
  - The number of samples and features are both large.

A future research direction can be to identify strategies that are efficient in every application scenario and investigate how they can be implemented in a parallel manner.

5.3.1 ONGOING RESEARCH

At the current state, we have implemented the base framework and are working on investigating the impact of network topology on the performance of a consensus-based distributed SVM in terms of convergence and the training time using distributed HPC architecture, MPI. We aim to compare impact of expander graphs with other type of networks. The ongoing research will likely lead to a journal article. Our plan is to add more features to the framework such as prototype selection and community selection. Prototype selection helps to clean the data and identify the important training samples that contribute to the separating hyperplane. Community selection helps to identify the graph nodes that are important for an efficient communication in the consensus-based distributed optimization. Additionally, we are working on evaluating the implemented algorithm on real-world data. For instance, there is a possibility of using real-data from Astra Zeneca.

REFERENCES

2011-2013, Evans Data Software Developer surveys (2011-2013). *Performance: Ready to Use*. https://software.intel.com/en-us/intel-mkl. [Online; accessed 11-October-2015].

Alham, Nasullah Khalid, Maozhen Li, and Yang Liu (2014). "Parallelizing multiclass support vector machines for scalable image annotation." In: *Neural Computing and Applications* 24.2, pp. 367–381.

ApacheSoftwareFoundation, The (2014). *What Is Apache Hadoop?* http://hadoop.apache.org/. [Online; accessed 05-August-2018].

Athanasopoulos, Andreas et al. (2011). "GPU acceleration for support vector machines." In: *WIAMIS 2011: 12th International Workshop on Image Analysis for Multimedia Interactive Services, Delft, The Netherlands, April 13-15, 2011*. TU Delft; EWI; MM; PRB.

Biasi, Ian, Andrea Boni, and Alessandro Zorat (2005). "A reconfigurable parallel architecture for SVM classification." In: *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*. Vol. 5. IEEE, pp. 2867–2872.

Bickson, Danny, Elad Yom-Tov, and Danny Dolev (2008). "A gaussian belief propagation solver for large scale support vector machines." In: *arXiv preprint arXiv:0810.1648*.

Bin, Zhao, Liu Yong, and Xia Shao-Wei (2000). "Support vector machine and its application in handwritten numeral recognition." In: *Pattern Recognition, 2000. Proceedings. 15th International Conference on*. Vol. 2, 720–723 vol.2. DOI: 10.1109/ICPR.2000.906176.

Boyd, Stephen et al. (2011). "Distributed optimization and statistical learning via the alternating direction method of multipliers." In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122.

Brugger, Dominik (2006). *Parallel Support Vector Machines*. WSI. Arbeitsbereich Technische Informatik, Universität Tübingen, p. 27. ISBN: 0946-3851.

Byun, Hyeran and Seong-Whan Lee (2002a). "Applications of support vector machines for pattern recognition: A survey." In: *Pattern recognition with support vector machines*. Springer, pp. 213–236.

– (2002b). "Applications of support vector machines for pattern recognition: A survey." In: *Pattern recognition with support vector machines*. Springer, pp. 213–236.

Cao, H. T. et al. (Dec. 2016). "Impacts of network topology on the performance of a Distributed Algorithm Solving Linear Equations." In: *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 1733–1738. DOI: 10.1109/CDC.2016.7798515.

Cao, L.J. et al. (July 2006). "Parallel sequential minimal optimization for the training of support vector machines." In: *Neural Networks, IEEE Transactions on* 17.4, pp. 1039–1049. ISSN: 1045-9227. DOI: 10.1109/TNN.2006.875989.

Carpenter, AUSTIN (2009a). "cuSVM: A CUDA implementation of support vector classification and regression." In: *patternsonscreen. net/cuSVMDesc. pdf*.

– (2009b). "cuSVM: A CUDA implementation of support vector classification and regression." In: *patternsonscreen. net/cuSVMDesc. pdf*.

Caruana, Godwin, Maozhen Li, and Man Qi (2011). "A MapReduce based parallel SVM for large scale spam filtering." In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*. Vol. 4. IEEE, pp. 2659–2662.

Chandra, R. (2001). *Parallel Programming in OpenMP*. High performance computing. Morgan Kaufmann Publishers. ISBN: 9781558606715.

Chang, Chih-Chung and Chih-Jen Lin (2011). “LIBSVM: A library for support vector machines.” In: *ACM Transactions on Intelligent Systems and Technology* 2 (3). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

Chang, Edward Y. (2011). “PSVM: Parallelizing Support Vector Machines on Distributed Computers.” In: *Foundations of Large-Scale Multimedia Information Management and Retrieval: Mathematics of Perception*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 213–230. ISBN: 978-3-642-20429-6. DOI: 10.1007/978-3-642-20429-6_10.

Chang, Pengfei, Zhuo Bi, and Yiyong Feng (July 2014). “Parallel SMO algorithm implementation based on OpenMP.” In: *System Science and Engineering (ICSSE), 2014 IEEE International Conference on*, pp. 236–240. DOI: 10.1109/ICSSE.2014.6887941.

Chatterjee, Anirban, Kelly Fermoyle, and Padma Raghavan (2010). “Characterizing sparse preconditioner performance for the support vector machine kernel.” In: *Procedia Computer Science* 1.1, pp. 367–375.

Chow, Y. T. et al. (Nov. 2016). “Expander graph and communication-efficient decentralized optimization.” In: *2016 50th Asilomar Conference on Signals, Systems and Computers*, pp. 1715–1720. DOI: 10.1109/ACSSC.2016.7869675.

Corporation, NVIDIA (2015). *cuBLAS*. https://developer.nvidia.com/cublas. [Online; accessed 11-October-2015].

– (2018). *NVIDIA TITAN V*. https://www.nvidia.com/en-us/titan/titan-v/. [Online; accessed 23-August-2018].

Cortes, Corinna and Vladimir Vapnik (1995). “Support-vector networks.” In: *Machine learning* 20.3, pp. 273–297.

Cotter, Andrew, Nathan Srebro, and Joseph Keshet (2011). “A GPU-tailored approach for training kernelized SVMs.” In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 805–813.

Dean, Jeffrey and Sanjay Ghemawat (Jan. 2008). “MapReduce: Simplified Data Processing on Large Clusters.” In: *Commun. ACM* 51.1, pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492.

Deng, Wei et al. (May 2017). “Parallel Multi-Block ADMM with o(1 / k) Convergence.” In: *Journal of Scientific Computing* 71.2, pp. 712–736. ISSN: 1573-7691. DOI: 10.1007/s10915-016-0318-2.

Dheeru, Dua and Efi Karra Taniskidou (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Díaz-Morales, R., H. Y. Molina-Bulla, and Á. Navia-Vázquez (July 2011). “Parallel Semiparametric Support Vector Machines.” In: *The 2011 International Joint Conference on Neural Networks*, pp. 475–481. DOI: 10.1109/IJCNN.2011.6033259.

Díaz-Morales, Roberto, Harold Y Molina-Bulla, and Angel Navia-Vázquez (2011). “Parallel Semiparametric Support Vector Machines.” In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, pp. 475–481.

Didiot, Emmanuel and Fabien Lauer (2015). “Efficient Optimization of Multi-class Support Vector Machines with MSVMpack.” In: *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Springer, pp. 23–34.

Do, Thanh-Nghi, Van-Hoa Nguyen, and François Poulet (2008). "Speed up SVM algorithm for massive classification tasks." In: *Advanced Data Mining and Applications*. Springer, pp. 147–157.

Do, Thanh-Nghi and Minh-Thu Tran-Nguyen (2016). "Incremental parallel support vector machines for classifying large-scale multi-class image datasets." In: *International Conference on Future Data and Security Engineering*. Springer, pp. 20–39.

Doan, Thanh-Nghi, Thanh-Nghi Do, and François Poulet (2013). "Large Scale Visual Classification with Parallel, Imbalanced Bagging of Incremental LIBLINEAR SVM." In: *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p. 1.

Domo, Inc. (2018a). *Data Never Sleeps 5.0*. https://www.domo.com/learn/data-never-sleeps-5. [Online; accessed 18-August-2018].

– (2018b). *Data Never Sleeps 6.0*. https://www.domo.com/learn/data-never-sleeps-6. [Online; accessed 18-August-2018].

Donetti, Luca, Franco Neri, and Miguel A Muñoz (2006). "Optimal network topologies: expanders, cages, Ramanujan graphs, entangled networks and all that." In: *Journal of Statistical Mechanics: Theory and Experiment* 2006.08, P08007. URL: http://stacks.iop.org/1742-5468/2006/i=08/a=P08007.

Du, Hongle et al. (2009). "A cooperative intrusion detection system based on improved parallel svm." In: *Pervasive Computing (JCPC), 2009 Joint Conferences on*. IEEE, pp. 515–518.

Duchi, John C, Alekh Agarwal, and Martin J Wainwright (2012). "Dual averaging for distributed optimization: Convergence analysis and network scaling." In: *IEEE Transactions on Automatic control* 57.3, pp. 592–606.

Durdanovic, Igor, Eric Cosatto, and Hans-Peter Graf (2007). "Large scale parallel SVM implementation." In: *Large Scale Kernel Machines*, pp. 105–138.

Eitrich, Tatjana and Bruno Lang (2005). *Efficient implementation of serial and parallel support vector machine training with a multi-parameter kernel for large-scale data mining*. Citeseer.

– (2006). "Data mining with parallel support vector machines for classification." In: *Advances in Information Systems*. Springer, pp. 197–206.

Ekanayake, Jaliya et al. (2010). "Twister: A Runtime for Iterative MapReduce." In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, pp. 810–818. ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851593.

Ferreira, L.V., E. Kaszkurewicz, and Amit Bhaya (2006). "Parallel Implementation of Gradient-Based Neural Networks for SVM Training." In: *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pp. 339–346. DOI: 10.1109/IJCNN.2006.246701.

Forero, Pedro A, Alfonso Cano, and Georgios B Giannakis (2010a). "Consensus-based distributed support vector machines." In: *The Journal of Machine Learning Research* 11, pp. 1663–1707.

– (2010b). "Consensus-based distributed support vector machines." In: *The Journal of Machine Learning Research* 11, pp. 1663–1707.

França, G. and J. Bento (Oct. 2017). “How is Distributed ADMM Affected by Network Topology?” In: *ArXiv e-prints*. arXiv: 1710.00889 [stat.ML].

Friedman, Joel (2004). “A proof of Alon’s second eigenvalue conjecture and related problems.” In: *CoRR* cs.DM/0405020. URL: http://arxiv.org/abs/cs.DM/0405020.

Garcı´a, Salvador, Julián Luengo, and Francisco Herrera (2015). *Data preprocessing in data mining*. Springer.

Gertz, E Michael and Joshua D Griffin (2010). “Using an iterative linear solver in an interior-point method for generating support vector machines.” In: *Computational Optimization and Applications* 47.3, pp. 431–453.

Gonçalves, João, Noel Lopes, and Bernardete Ribeiro (2012). “Multi-threaded support vector machines for pattern recognition.” In: *Neural Information Processing*. Springer, pp. 616–623.

Graf, Hans P, Srihari Cadambi, et al. (2009). “A massively parallel digital learning processor.” In: *Advances in Neural Information Processing Systems*, pp. 529–536.

Graf, Hans P, Eric Cosatto, et al. (2004). “Parallel support vector machines: The cascade SVM.” In: *Advances in neural information processing systems*, pp. 521–528.

Guo, Wenming et al. (2015). “A Resource Aware MapReduce Based Parallel SVM for Large Scale Image Classifications.” In: *Neural Processing Letters*, pp. 1–24.

He, Bingsheng and Xiaoming Yuan (2012). “On the O(1/n) Convergence Rate of the Douglas–Rachford Alternating Direction Method.” In: *SIAM Journal on Numerical Analysis* 50.2, pp. 700–709.

He, Qing et al. (2011). “A parallel incremental extreme SVM classifier.” In: *Neurocomputing* 74.16, pp. 2532–2540.

Hoory, Shlomo, Nathan Linial, and Avi Wigderson (2006). “Expander graphs and their applications.” In: *Bulletin of the American Mathematical Society* 43.4, pp. 439–561.

Hsieh, Cho-Jui, Si Si, and Inderjit S. Dhillon (2014). “A Divide-and-conquer Solver for Kernel Support Vector Machines.” In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China: JMLR.org, pp. I-566–I-574. URL: http://dl.acm.org/citation.cfm?id=3044805.3044870.

Hsu, Chih-Wei, Chih-Chung Chang, Chih-Jen Lin, et al. (2003). “A practical guide to support vector classification.” In:

Hwu, W.W. (2011). *GPU Computing Gems Emerald Edition*. Applications of GPU Computing Series. Elsevier Science. ISBN: 9780123849892.

Ivanciuc, Ovidiu (2007a). “Applications of support vector machines in chemistry.” In: *Reviews in computational chemistry* 23, p. 291.

– (2007b). “Applications of support vector machines in chemistry.” In: *Reviews in computational chemistry* 23, p. 291.

Jin, Jing, Xianggao Cai, and Xiaola Lin (2013). “Efficient SVM Training Using Parallel Primal-Dual Interior Point Method on GPU.” In: *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference on*. IEEE, pp. 12–17.

Joachims, T. (1998). *Making large-Scale SVM Learning Practical*. LS8-Report 24. Universität Dortmund, LS VIII-Report.

Johnson, Steven G. (2008). *The NLopt nonlinear-optimization package*. URL: http://ab-initio.mit.edu/nlopt.

Li, Qunwei et al. (2017). "Robust Federated Learning Using ADMM in the Presence of Data Falsifying Byzantines." In: *CoRR* abs/1710.05241. arXiv: 1710.05241. URL: http://arxiv.org/abs/1710.05241.

Li, Tao et al. (2013). "GPU Acceleration of Interior Point Methods in Large Scale SVM Training." In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, pp. 863–870.

Liao, Quan et al. (2009). "GPU accelerated support vector machines for mining high-throughput screening data." In: *Journal of chemical information and modeling* 49.12, pp. 2718–2725.

Lu, Yunmei et al. (2014). "A Survey of GPU Accelerated SVM." In: *Proceedings of the 2014 ACM Southeast Regional Conference*. ACM SE '14. Kennesaw, Georgia: ACM, 15:1–15:7. ISBN: 978-1-4503-2923-1. DOI: 10.1145/2638404.2638474.

Malliaros, Fragkiskos D and Vasileios Megalooikonomou (2011). "Expansion properties of large social graphs." In: *International Conference on Database Systems for Advanced Applications*. Springer, pp. 311–322.

Marzolla, Moreno (2011). "Fast training of support vector machines on the Cell processor." In: *Neurocomputing* 74.17, pp. 3700–3707.

Masih, Shraddha and Sanjay Tanwani (2014). *Data Mining Techniques in Parallel and Distributed Environment-A Comprehensive Survey*.

Matloff, Norm (2011). "Programming on parallel machines." In: *University of California, Davis*.

Matsushima, Shin, SVN Vishwanathan, and Alexander J Smola (2012). "Linear support vector machines via dual cached loops." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 177–185.

Maxfield, Clive (2011). *FPGAs: instant access*. Newnes.

Miller, Steven J, Tim Novikoff, and Anthony Sabelli (2008). "The distribution of the largest nontrivial eigenvalues in families of random regular graphs." In: *Experimental Mathematics* 17.2, pp. 231–244.

Narasimhan, Jeyanthi et al. (2014). "Fast Support Vector Machines Using Parallel Adaptive Shrinking on Distributed Systems." In: *arXiv preprint arXiv:1406.5161*.

Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Null, Linda, Julia Lobur, et al. (2014). *The essentials of computer organization and architecture*. Jones & Bartlett Publishers.

Papadonikolakis, Markos, Christos-Savvas Bouganis, and George Constantinides (2009). "Performance comparison of GPU and FPGA architectures for the SVM training problem." In: *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, pp. 388–391.

Peng, Nanbo, Yanxia Zhang, and Yongheng Zhao (2011). "CUDA-accelerated SVM for celestial object classification." In: *Astronomical Data Analysis Software and Systems Xx*. Vol. 442, p. 119.

Platt, John (Apr. 1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Tech. rep. MSR-TR-98-14. Microsoft Research, p. 21. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=69644.

Platt, John et al. (1998). “Sequential minimal optimization: A fast algorithm for training support vector machines.” In:

Reyna-Rojas, RA et al. (2003). “Implementation of the SVM generalization function on FPGA.” In: *International Signal Processing Conference (ISPC), Dallas (US)*, pp. 147–153.

scikit-learn (2015). *RBF SVM parameters*.

Shalev-Shwartz, Shai et al. (2011). “Pegasos: Primal estimated sub-gradient solver for svm.” In: *Mathematical programming* 127.1, pp. 3–30.

Shrivastava, Naveeen Kumar, Praneet Saurabh, and Bhupendra Verma (2011). “An efficient approach parallel support vector machine for classification of diabetes dataset.” In: *J. Computer Applications* 36.6, pp. 19–24.

Takác, Martin et al. (2013). “Mini-Batch Primal and Dual Methods for SVMs.” In: *ICML (3)*, pp. 1022–1030.

Tao, H., B. Wu, and X. Lin (Dec. 2014). “Budgeted mini-batch parallel gradient descent for support vector machines on Spark.” In: *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 945–950. DOI: 10.1109/PADSW.2014.7097914.

Tavara, S, H Sundell, and A Dahlbom (2015). “Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM.” In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p. 177.

Tavara, Shirin (2018). “Parallel Computing of Support Vector Machines: A Survey.” In: *ACM Computing Surveys (CSUR)*.

Tavara, Shirin and Alexander Schliep (2018). “Effect Of Network Topology On The Performance Of ADMM-based SVMs.” In: *High-Performance Machine Learning Workshop (HPML)*.

Tyree, Stephen et al. (2014). “Parallel Support Vector Machines in Practice.” In: *arXiv preprint arXiv:1404.1066*.

Vapnik, Vladimir (2013). *The nature of statistical learning theory*. Springer Science & Business Media, p. 314. ISBN: 9781475732641.

Vapnik, Vladimir N (1999). “An overview of statistical learning theory.” In: *Neural Networks, IEEE Transactions on* 10.5, pp. 988–999.

Venkateshan, Sriram, Alap Patel, and Kuruvilla Varghese (2015). “Hybrid Working Set Algorithm for SVM Learning With a Kernel Coprocessor on FPGA.” In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.10, pp. 2221–2232.

Vishnu, Abhinav et al. (2015). “Fast and Accurate Support Vector Machines on Large Scale Systems.” In: *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, pp. 110–119.

Vural, Volkan and Jennifer G Dy (2004). “A hierarchical method for multi-class support vector machines.” In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 105.

Woodsend, Kristian and Jacek Gondzio (2009a). “High-performance parallel support vector machine training.” In: *Parallel Scientific Computing and Optimization*. Springer, pp. 83–92.

– (Dec. 2009b). “Hybrid MPI/OpenMP Parallel Linear Support Vector Machine Training.” In: *J. Mach. Learn. Res.* 10, pp. 1937–1953. ISSN: 1532-4435.

Yang, L.T. and M. Guo (2005). *High-Performance Computing: Paradigm and Infrastructure*. Wiley Series on Parallel and Distributed Computing. Wiley. ISBN: 9780471732709.

You, Yang, J. Demmel, et al. (May 2015). “CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems.” In: *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pp. 847–859. DOI: 10.1109/IPDPS.2015.117.

You, Yang, S.L. Song, et al. (May 2014). “MIC-SVM: Designing a Highly Efficient Support Vector Machine for Advanced Modern Multi-core and Many-Core Architectures.” In: *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 809–818. DOI: 10.1109/IPDPS.2014.88.

Yuan, Guo-Xun, Chia-Hua Ho, and Chih-Jen Lin (2012). “Recent advances of large-scale linear classification.” In: *Proceedings of the IEEE* 100.9, pp. 2584–2603.

Zanghirati, Gaetano and Luca Zanni (2003). “A parallel solver for large quadratic programs in training support vector machines.” In: *Parallel computing* 29.4, pp. 535–551.

Zhao, HX and Frédéric Magoules (2011). “Parallel support vector machines on multi-core and multiprocessor systems.” In: *11th International Conference on Artificial Intelligence and Applications (AIA 2011)*. IASTED.

Zhu, Kaihua et al. (2008). “Parallelizing Support Vector Machines on Distributed Computers.” In: *Advances in Neural Information Processing Systems 20*. Ed. by J.C. Platt et al. Curran Associates, Inc., pp. 257–264. URL: http://papers.nips.cc/paper/3202-parallelizing-support-vector-machines-on-distributed-computers.pdf.

Zhu, Zeyuan Allen et al. (2009). “P-packSVM: Parallel primal gradient descent kernel SVM.” In: *Data Mining, 2009. ICDM’09. Ninth IEEE International Conference on*. IEEE, pp. 677–686.

PART I
ARTICLE I

Parallel Computing of Support Vector Machines: A Survey

SHIRIN TAVARA, University of Borås & University of Skövde

The immense amount of data created by digitalization requires parallel computing for machine learning methods. While there are many parallel implementations for support vector machines, there is no clear suggestion for every application scenario. Many factors including optimization algorithm, problem size and dimension, kernel function, parallel programming stack, hardware architecture impact the efficiency of implementations. It is up to the user to balance trade-offs particularly between computation time and the classification accuracy. In this survey, we review the state of the art implementations of SVM, their pros and cons, and suggest possible avenues for future research.

CCS Concepts: •**Computing methodologies** →**Support vector machines;** *Parallel algorithms; Ensemble methods; Feature selection; Cross-validation;*

Additional Key Words and Phrases: Dual Optimization, Primal Optimization, Decomposition, CPU Parallelism, GPU Parallelism, Speedup, Data Movement

ACM Reference format:
Shirin Tavara. 2016. Parallel Computing of Support Vector Machines: A Survey. *ACM Comput. Surv.* 1, 1, Article 1 (January 2016), 35 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

High-Performance Computing (HPC) [1] and parallel computing are promising tools for improving the performance of machine learning algorithms in terms of time, especially for large-scale problems. Support Vector Machines (SVM) [2] is a supervised machine learning technique that can take the advantage of HPC. SVM is a popular technique because of the good generalization performance on many real-life data [3, 4]. Generalization performance shows how accurately unseen data can be classified by a pattern classifier [3, 5]. Despite the advantages of SVM, it suffers from a long training process and limited memory for solving large-scale problems [3, 6–10]. The reason is that the Quadratic Programming (QP) [11] problem addressed by SVM contains computationally expensive tasks [11, 12] including matrix related operations [11] for kernel computations, checking optimality conditions [13] and gradient updating tasks [14, 15].

In the digital era, the size of data has been growing exponentially, thus the computation time and memory requirements regarding very large problems have been increasing and using HPC tools has become even more important [16]. The published results show that parallel SVMs can achieve considerable speedups compared to sequential SVM algorithms that use only a single CPU [10]. Parallelism has potential to improve the performance of SVM in terms of time and memory, however, parallel implementations of SVM are far beyond easy tasks [17] and may become inefficient for solving very large problems or using a large number of processors, i.e., they

This work is supported by Universities of Borås and Skövde in Sweden.
Author's addresses: SH. Tavara, Faculty of Librarianship, Information, Education and IT, University of Borås and School of Informatics, University of Skövde.

DOI: 10.1145/nnnnnnn.nnnnnnn

may not scale well to a large number of samples or processors. Problems such as communication overheads, computationally dependent steps, and memory limitations degrade the efficiency of parallelism [6]. Besides, coding part of a parallel SVMs algorithm is difficult and requires considerable skills [3]. Due to the high space and time complexity of SVM algorithms, it is important to use the right and appropriate algorithms and efficient heuristics for the given problem. Thereby, to identify the appropriate parallel approaches that have potentials to obtain the peak performance for the given large-scale problem, it is helpful to thoroughly study the efficiency of existing parallel approaches.

Tyree et al. [18] categorize the approaches used for the parallelization of SVM into the implicit and explicit parallelization. In the implicit parallelization [18], an algorithm is written as a series of operations that can be computed using highly optimized parallel libraries, e.g., Intel Math Kernel Library (MKL) [19] and cuBLAS [20]. In the explicit parallelization [18], programmers parallelize the computationally expensive tasks using parallel programmings, e.g., shared memory, distributed memory, and GPUs. The review conducted by Tyree et al. covers most of the existing parallel implementations of SVM and it shows the importance of the implicit parallelism. Despite many advantages of their review, it lacks to consider the important parallel algorithmic approaches and paradigms, e.g., the cascade and map-reduce, and it only considers SMO-type optimization methods. Lu et al. [21] review the mathematical optimization approaches used for accelerating the training process of SVM. The survey only considers the GPU-based parallel implementations of SVM. To our knowledge, there is no survey that thoroughly reviews the parallel algorithmic approaches along with parallel tools for implementations of SVM.

The objective of this paper is to provide a summary of the parallel algorithmic approaches and parallel tools that have been used for implementations of SVM in order to provide insights into efficient and potential approaches for solving large-scale problems. Besides, the paper provides a brief summary of promising heuristics, their advantages, and challenges in the parallelism. In this paper, the dominant and promising parallel approaches that can be the target of future studies for further improvements are identified. We review parallel implementations of SVM with respect to four focus lines that we have identified as the important goals of parallelism.

Two of the focus lines are speedup and memory, i.e., the parallel implementations of SVM may focus on improving the performance of SVM algorithms in terms of time and memory for large-scale problems [18, 21]. Parallel approaches that reduce the problem size and handle the memory issues, may face deterioration in the classification accuracy. Thereby, another focus line is the accuracy, i.e., parallel approaches may focus on improving accuracy or maintaining accuracy while reducing the computation time. The fourth focus line is the scalability, i.e., parallel algorithms may focus on scaling well for a large number of training samples and a large number of processors or more focus on minimizing overheads.

The outline of the paper is as follows. An overview of SVM is described in section 2. The methodology for choosing and reviewing the publications regarding parallel SVM and the corresponding taxonomy are described in section 3. A review of the parallel algorithmic techniques and models that have been used for SVM is briefly described in section 4. A brief summary and some of the remarkable works along with the discussion are mentioned in section 5. The conclusion and potential future works are mentioned in section 6.

2 WHAT ARE SUPPORT VECTOR MACHINES

SVM is a supervised machine learning technique developed by Vapnik et al. [2] from statistical learning theory to solve classification and regression problems [2, 11, 22]. The basic idea of SVM in a simple binary classification problem is to search for the hyperplane that is the farthest to the closest training data points from both sides of the hyperplane [23, 24]. This process has two phases, training and testing. In the training phase, the machine is trained to find a hyperplane that separates the given data samples into two classes with known labels, negative or -1 and positive or $+1$. After the machine is trained, the training model is extracted and then the testing phase is carried out. In the testing phase, the SVM model predicts which class label a new unseen test sample should have [25]. As mentioned in section 1, SVM gives a good generalization performance [25] and minimizes

Table 1. Examples of well-known kernel functions

Kernel Function	Inner Product	Kernel Type
Linear kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	linear
Gaussian/Radial-Basis Function (RBF)	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$	non-linear
Polynimial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + const)^d$	non-linear
Laplacian	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|)$	non-linear

the upper bound of the generalization error [3]. SVM has special characteristics that are used to implement efficient parallel algorithms in terms of time and memory. One characteristic is that the solution to a classification problem is obtained by only a few samples called Support Vectors (SV) [2] that determine the maximum margin separating hyperplane [26]. Another characteristic of SVM is to perform the nonlinear mapping without knowing the mapping function using predefined functions called kernels for calculating the inner product of mapping functions [26]. Other characteristics of SVM are the simple structure of constraints and the especial definition of the kernel function in a linear case, i.e., the inner product is a simple dot product [27]. The sparsity of solutions is the next characteristic of SVM [28]. The primal optimization problem addressed by SVM is as follows,

$$\begin{aligned} min \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ s.t. \quad \forall i: \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad , i = 1, 2, ..., N \\ \forall i: \quad & \xi_i \geq 0 \quad , i = 1, 2, ..., N \end{aligned} \tag{1}$$

Here, $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}$ is a vector of observations, y_i is the class labels and $y_i \in \{+1, -1\}$, b is the bias parameter and $\Phi(\mathbf{x})$ is the map function. In cases that data can be classified by a linear classifier, thus $\Phi(\mathbf{x}) = \mathbf{x}$, but real-life data cannot always be classified by a linear classifier [3]. In non-linear cases, one can map the data from the input space into a high dimensional feature space using a non-linear transformation, i.e. $\Phi(\mathbf{x})$ maps the input vector $\mathbf{x}$ to the feature space [3]. In the feature space, the data can be linearly separable. Consequently, the dual form of equation (1) is represented in equation (2).

$$\begin{aligned} min \quad & D(\alpha) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ s.t. \quad & \sum_{i=1}^{N} y_i\alpha_i = 0 \quad , i = 1, 2, ..., N \\ \forall i: \quad & 0 \leq \alpha_i \leq C \quad , i = 1, 2, ..., N \end{aligned} \tag{2}$$

Here, $\boldsymbol{\alpha}$ is the vector of Lagrangian multipliers ($\alpha_i \in \boldsymbol{\alpha}$), $\mathbf{1}^T$ is a vector of ones, Q is a matrix of size $N \times N$ and $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$.

Kernel functions. In order to compute matrix Q, it is sufficient to compute the inner product of $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$ without knowing the $\Phi(\mathbf{x})$ function. This is done through a pre-defined kernel function, $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. The kernel matrix measures the similarity or the distance between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ [25]. Examples of well-known kernel functions are presented in table 1. The kernel function defines the feature space where the training samples are classified, therefore the selection of an appropriate kernel function is important [29].

Multi-class classification. SVM can solve multi-class classifications either by considering the multi-class classification in the optimization problem or though decomposing the multi-class classification into a series

of binary class classification [30]. The latter group is popular and includes One-Versus-One (OVO) [31] (also know as One-against-One), One-Versus-All (OVA) [31] (also known as One-against-All), and Directed Acyclic Graphs (DAGs) [30, 32]. In OVO, all binary combinations of N classes are created, thus $N(N-1)/2$ classifiers are built, where N is the number of classes [30]. DAGs is a directed graph that combines the results of OVO classifiers [32]. In OVA, the samples of a specific class (class i) will be considered as the positive class and all the remaining samples will be considered as the negative class. Thereby, N different classifiers are built [30]. Multi-class classifications are computationally expensive and they suffer from long training time due to involving many classes in the classification and passing through the training data many times [33]. For a large number of classes, OVO is more computationally expensive than the OVA [30].

2.1 Challenges

The training phase of SVM involves a dense QP and solving such QP is computationally expensive since it involves computations of a Hessian/kernel matrix [34, 35]. Kernel evaluations include computationally expensive tasks [14, 36], i.e., matrix-vector and matrix-matrix multiplications, gradient function updates [14] and optimality condition updates [33, 37]. The old and general purpose QP solvers are no longer suitable for solving the QP addressed by SVMs [3, 11, 25, 34] since many of QP solvers require computing and storing the kernel matrix in the memory which is not always possible due to the memory limitations [38]. One efficient approach to improve the performance of SVM algorithms is parallelism, however, the parallel approach used in the general purpose QP solvers may not take the advantage of the special characteristics of SVMs mentioned in section 2 [14, 27, 39, 40] or they may not be easily parallelizable [27]. Besides, the immense size of real-life data can cause problems regarding computationally expensive tasks as follows.

- **Memory.** The whole data may not fit into the memory [35] or inefficient memory access slows down the training and testing phases [41].
- **Speedup.** The matrix operations might take too long time to be performed due to the computationally expensive tasks, e.g., matrix-vector, matrix-matrix multiplications [14], and overheads [42].
- **Scalability.** Algorithms may not scale to a large number of processors or a large number of samples [27].
- **Accuracy.** Approximation methods [43] for reducing the size of the problem may lead to poor classification accuracy [44].

Some efficient heuristics along with parallel approaches have been used to speed up the optimization addressed by SVM and to further handle the issues mentioned regarding memory, speedup, scalability, and accuracy. A brief overview of the common heuristics and the corresponding challenges are mentioned as follows.

- **Grid Search And Cross-Validation.** One can improve the accuracy of an SVM model by selecting appropriate values for the model's parameters. This is done through grid search and cross-validation [45] in which a grid of different value sets of parameters is generated and in each value set, cross-validation is conducted. In n-fold cross-validation, the training set is randomly divided into n subsets with almost equal sizes. In each fold, one subset is used as the validation set for testing the model and $(n-1)$ subsets are used as the training set [46]. The cross-validation process is computationally expensive due to recomputing kernel matrix values at each iteration [45]. The process gets more computationally expensive for multi-class classifications since recomputations of the kernel elements may be repeated for each fold.
- **Caching.** Kernel evaluations are computationally expensive and every evaluation of $K(x_i, x_j)$ requires at least $O(d)$ flops, where d is the number of features [15]. Consequently, computing a submatrix of size $s \times m$ requires at least $O(smd)$ flops at each training step, where s is the number of rows and m is the number of columns [15]. Many computations in kernel evaluations are repeated or unused, therefore one can reduce the memory requirements and the computation time by avoiding recomputations of

the kernel evaluations and only storing the previously computed values in the memory using caching [15]. To do so, caching stores a number of rows of the kernel matrix as the memory allows. One of the common updating strategies for caching is Least Recently Used (LRU) [47] technique [47, 48]. Caching has been well studied and been a popular heuristic in parallel implementations of SVMs [27, 28, 48–54]. Beside the advantages, caching is difficult and system dependent [3, 37] and it might be inefficient for a large number of training samples [55, 56]. The reason is that the number of the cached rows of the kernel matrix is small due to limited memory, thereby, the size of the active sets will not be large enough to achieve fast optimization [55, 57] or if an algorithm is memory bound, the memory access cost will be high and multi-threading might cause memory contention due to limited bandwidth [56].

– **Shrinking.** Based on empirical studies and in practice, the number of SVs is much less than the total number of training samples. In order to solve the QP faster in each iteration, one can find SVs in advance and perform the training only on the SVs and discard the non-SVs in the optimization process. This makes the QP smaller and faster than the QP for all the training samples in each iteration and it obtains the same optimal result [58, 59]. Joachims et al. [58] proposed a strategy called shrinking that temporarily eliminates points that are unlikely to be selected in the working set in each iteration. To do this, Joachims et al. [58] temporarily discard samples that the corresponding Lagrangian multipliers reach a pre-defined upper or lower bound. Shrinking reduces the computations regarding kernel evaluations since only parts of the Hessian matrix that correspond to SVs are calculated [49, 60]. Although shrinking has been used in parallel implementations of SVMs [13, 17, 47, 49, 56, 61, 62], except in a very few number of articles, the process of shrinking has not been parallelized [63]. Despite the advantages, shrinking requires rearranging index, marshaling, and reconstruction, which may cause significant overheads and therefore it may lead to the performance loss. Therefore, one should get enough information from datasets and the problem parameters to decide how to carefully use shrinking [56].

3 METHODOLOGY

This survey is based on reviewing the publications and information in technical books, journals, conference proceedings, technical reports, authentic websites and libraries used for parallel implementations of SVMs. There has been no special focus on applications of SVMs in a specific field such as medicine, biomedical or finance. The selection of journal and conference articles were conducted using well-known databases, e.g. IEEE, Elsevier, ACM digital library, and the Google Scholar search engine. Parallel SVM implementations that use parallel algorithmic approaches, parallel models and frameworks are chosen regardless of their applications. In addition to the parallel approaches, the heuristics and strategies that improve the performance of the SVM algorithms with respect to the four focus lines are mentioned in this survey. Besides, the parallel implementations of SVMs using FPGA are briefly mentioned, although the customized and dedicated hardware for computational purposes is not the main focus of this survey. A review of publications regarding the sequential implementations of SVMs is excluded from this survey. The body of research regarding parallel computing of SVMs can be studied considering two aspects, i.e., parallel algorithms and parallel models related to parallel architectures. In this survey, the parallel SVM algorithms are identified and categorized and within each category, parallel models used for the parallel implementations of SVMs are reviewed. The resulting taxonomy of parallel SVMs is illustrated in figure 1.

4 PARALLEL ALGORITHMIC APPROACHES AND PARALLEL MODELS

Parallelization of SVM algorithms is far beyond an easy task due to problems such as dependencies between the computation steps [17], high latency in memory access [41, 56], and limited memory [56]. In this section, the most common techniques and heuristics used for parallel implementations of SVM are briefly described, each of which improves the efficiency of SVM algorithms in terms of memory, speedup, scalability and accuracy.

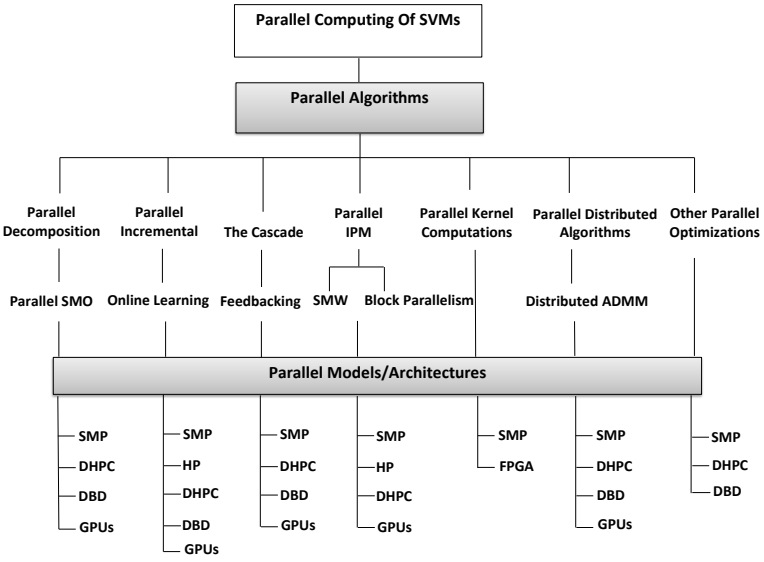


Fig. 1. The parallel computing of SVMs, including algorithms and tools. SMP: Shared Memory Parallelism, DHPC: Distributed HPC Architectures, HP: Hybrid shared-distributed memory Parallelism, GPUs: GPU-based Parallelism, DBD: Distributed Bid Data architectures, and FPGA: Field Programmable Gate Arrays.

4.1 Parallel Decomposition Techniques

Parallel decomposition implementations are popular for large-scale SVM problems since at each iteration, it uses only a few samples in the working sets and skips the rest. However, decomposition techniques are inherently and essentially sequential. This is because the selection of working sets at each iteration depends on the results of the previous iteration, albeit some of the computationally expensive tasks/operations can be performed in parallel. In this section, the parallel decomposition-based SVMs are briefly described.

4.1.1 Parallel Decompositions Using Shared Memory Parallelism (SMP). SMP has been used for performing decomposition techniques in parallel [13, 15, 37, 61, 64–66]. To do so, computationally expensive tasks such as kernel evaluations, gradient updates and working set selections [15, 66] are performed in parallel.

Memory and speedup. The advantage of parallel implementations of decomposition algorithms using SMP is that one copy of data is stored and shared between different processors/threads, avoiding multiple copies of data or communication overheads for transferring data [13]. Eitrich and Lang [66] use a simple loop-based parallelism for efficiently performing kernel evaluations, gradient updates, and selection of the working set in parallel. They employ BLAS routine libraries to accelerate matrix-vector and matrix-matrix multiplications. The advantage of parallel decomposition techniques is that depending on the size of the working set, only a small subset of the large kernel matrix is computed in parallel [15]. However, if the appropriate size of the working set is large, the data may not fit in the limited shared memory. Going beyond a binary classification, Didiot et al. [15] reduce the memory requirements for multi-class classifications using kernel caching in which kernel elements are shared across all the processors/threads. The proper data alligenment in [15] allows to vectorize the kernel functions in which Single Instruction Multiple Data (SIMD) [67] instructions are used to apply the same operation on multiple components of large vectors. Unlike previous work, You et al. [56] remove the caching and reduce the

shrinking frequency in SMO and focus on memory coverage using a two-level parallel mechanism in which tasks with independent computations and memory requirements, e.g., kernel evaluations, are computed through task parallelism and tasks with dependency in the computations, e.g., partial kernel evaluations, are computed through data parallelism. Unlike the previous works, this two-level parallelism supports both dense and sparse data formats. They were the first to introduce the use of modern processor technologies for the vectorization.

Scalability and accuracy. Eitrich et al. [37] introduce a data transformation in which an expensive operation in the kernel function is transformed into a less expensive operation, i.e., a division is transformed to a multiplication. The data transformation along with the largest possible working set that fit in the memory leads to a speedup of 5.5 times using 7 threads for a binary classification. The data transformation allows using a Gaussian kernel without additional cost. However, the algorithm does not scale well to a large number of processors. In contrast, Gonccalves et al. [64] use Universal Kernel Function (UKF) [64] that has less computationally expensive operations than a Gaussian kernel. It yields a better speedup and a better or equal accuracy compared to a Gaussian kernel [64]. However, the impacts of using the UKF kernel on different applications are unknown. Chang et al. [65] take a step further and parallelize a decomposition-based algorithm for a multi-class classification, but, they only report experiments using a linear kernel. Unlike previous work, Marzolla [61] parallelizes a decomposition-based algorithm using a specific multi-core processor called the cell processor. The algorithm obtains low speedups on datasets with few attributes due to transferring many small data blocks.

4.1.2 Parallel Decompositions Using Distributed HPC Architectures. Parallel implementations of SVMs using SMP can achieve considerable speedups. Still, due to the limited capacity of the shared memory, data may not fit in the memory. To overcome the limited shared memory, distributed HPC architectures have been employed in which local memory is used without a single global memory space shared between processors. To do so, the computationally expensive tasks in a decomposition technique are distributed across multiple processors to be performed in parallel [14, 33, 45, 63, 68, 69]. In this regard, a particular decomposition algorithm called Sequential Minimal Optimization (SMO) [22] has been popular [13, 26, 28, 33, 48, 65, 70–72] since it can analytically solve the QP addressed by SVM. The basic idea in SMO is that a large QP problem is divided into the smallest possible QP subproblems each of which is solved using only two variables in their working set. SMO has been a target of distributed memory parallelism since around 90% of the total computation time is spent on updating the gradient function [33]. The most common approach for parallelization of SMO is to divide the training dataset into smaller subsets and then distribute the subsets into multiple processors where the local gradient functions can be calculated and updated in parallel [33, 73]. Shrinking [63, 63, 68, 68], caching [14] and two-level parallelism [56] are the most common strategies used for reducing the problem size and memory requirements.

Scalability and accuracy. The parallelization of SMO-based algorithms might achieve satisfactory speedups [33], however, they may not show good scalability in terms of the number of processors and training samples. This is because that those algorithms often require to go through the entire samples to select appropriate working sets [63] or processors may often need to communicate with each other to obtain the final result. This causes overheads. Narasimhan et al. [63] overcome scalability issues by adding an adaptive shrinking in which non-contributing training samples are eliminated, thus the algorithm deals with only a part of the samples. Shrinking reduces the memory requirements along with accelerating the training process. The time complexity of the shrinking process is further reduced by performing the shrinking in parallel. In [63], the adaptive shrinking allows fast convergence. However, incorrect eliminations of samples in shrinking result in an inaccurate classification. Vishnu et al. [68] overcome this problem using several shrinking strategies ranging from early to late elimination of non-contributing samples, in which important data structures are synchronized using distributed memory parallelism to avoid false elimination. Their proposed algorithm shows good scalability, i.e., it scales up to 4096 cores (256 nodes), for a problem with 2.3 million samples. Besides the advantages of decomposition techniques, they may not converge if SVs do not fit in the memory due to a phenomenon called thrashing in which a correction

on a working set is canceled by the correction of another working set [74]. Yom-Tov [74] overcomes this problem using batch mode in which every Lagrangian multiplier is updated at each iteration. Scalability of the proposed algorithm is unclear and the experiments report the performance for only 3 to 4 processors.

Memory and speedup. Shrinking might cause overheads if it is not handled carefully (section 2.1). One may skip inefficient shrinking and use caching instead (section 2.1). For instance, Brugger [14] uses parallel and distributed caching along with a sparse data presentation and large working sets in an SMO-based algorithm. The results of experiments show the linear and in some cases superlinear speedups for large problems. The approach in [14] supports different formulations of SVMs for classification and regression problems.

Speedup. While SMO is one of the most common decomposition methods used in SVMs, it may suffer from the existing dependencies between the computation steps. To overcome this problem, Hazan et al. [75] proposed a different decomposition technique using Fenchel duality that is more suitable for the distributed memory parallelism and for computing clusters of independent nodes with independent memory. The proposed algorithm uses a parallel block-update approach that can update and send back the results of all processors in an iterative manner with low communication overheads [75]. This leads to a speedup of $\frac{k}{2}$ using k processors.

4.1.3 Parallel Decompositions Using Distributed Big Data Architectures. These architectures have been used for parallel implementations of SMO [13, 44, 76] .

Memory and scalability. The parallel algorithm proposed by Zhao et al. [13] reduces the memory requirements using a map-reduce implementation of SMO along with caching and a sparse matrix representation. Caching reduces memory requirements since a unique copy of the cache is shared between all mappers, and the sparse data representation compresses the sample vectors to occupy as less memory as possible. Nevertheless, the sample vectors need to be reformed back into the dense format to compute the corresponding dot product. The combination of caching, sparse data representation and parallel implementation using map-reduce leads to 4 fold speedup compared to Libsvm. Concerning the memory usage, the algorithm could use a larger size of cache and concerning the scalability, the algorithm shows only slight time improvement for increasing number of cores. The reason is that the memory is not fully and efficiently used.

Accuracy. Distributed training of SVM may cause accuracy degradation. One can improve the accuracy using an ontology-based enhancement in which an end-user corrects and modifies the training data [44]. To do so, the user adds some optimized instances (intelligence) as feedbacks to the weighted training data chunks and then all the local SVMs are re-trained in parallel. For instance, Caruana et al. [44] design a feedback-based approach for training and classification processes in which the optimized instances are added into the feedback loops leading to an average of 5% accuracy improvement. The classification accuracy of this approach depends on the quality of provided intelligence by the end-user which requires expert knowledge in the domain.

The previous approaches mostly take homogeneous computing environments into the consideration in which machines load equal sizes of data. The reason can be that the most of the approaches perform the parallelism via Message Passing Interface (MPI) [77], and MPI was primarily designed for homogeneous computing environments [76]. In contrast, Hadoop implementation of map-reduce could use a cluster of distributed heterogeneous computers. However, it does not handle load balancing. One can balance loads for distributed heterogeneous computers using resource aware algorithms. Alham et al. [76] balance the loads using a genetic algorithm scheme which distributes data chunks of different sizes into heterogeneous computers using map-reduce in which the processing times for all data chunks are equalized and the communication overheads are reduced.

4.1.4 Parallel Decompositions Using GPUs. The computational power of GPUs has been used to speed up the computationally expensive kernel evaluations in decomposition techniques [10, 12, 28, 45, 50, 52, 78–84]. In GPU-based parallelism, using a sparse data format [79, 80, 82], selecting appropriate working sets [12, 48, 79] and adjusting data precision [28, 45, 48] are the most common strategies used for overcoming the limited memory bandwidth in GPUs and accelerating training/predicting processes for SVM.

Memory and accuracy. The first GPU-based parallel SVM, gpuSVM, is implemented by Catanzaro et al. [48]. They reduce the memory requirements in gpuSVM and consequently accelerate the training/testing process in SVM using single precision floating point arithmetic. Note that the 32-bit arithmetic deteriorates the accuracy [28, 45]. Carpenter [28] propose a solution, cuSVM, to improve the accuracy using the mixture of single and double precision arithmetic. Although this mixture boosts the performance on dense samples, it does not duplicate the accuracy achieved by Libsvm and does not support the cross-validation for improving the accuracy [45]. Besides, cuSVM in many cases performs slower than gpuSVM.

Memory reduction through reducing precision may not lead to the desired results. Therefore, one may reduce memory requirements using a sparse format for storing large samples [79, 80, 82]. Sopylaya et al. [80] show that simply using a compressed sparse row format leads to a speedup of 35 times, however, the algorithm in [80] only supports cases with sparse input data and not dense ones [29]. Cotter et al. [41] mention that the sparse format of data can reduce memory requirements, but it may not follow the certain restrictive memory access patterns on GPUs and therefore they use coalesced memory access for sparse data that fits the GPU architecture [41]. Different from the previous works that focus on the parallelization of matrix-vector multiplication for updating the gradients, Vanek et al. [83] introduce an Optimized Hierarchical Decomposition SVM (OHD-SVM) [83] in which kernel values are computed via matrix-matrix multiplications that fits better the GPU architecture. OHD-SVM achieves significantly better performance than in [18, 41, 85] and it achieves the speedup up to 12 times over the fastest published GPU-based SVM for binary classifications, gpuSVM [48]. It is the only implementation that supports both sparse and dense data formats with the efficient loading of data from the memory.

Cross-validation. It is used to enhance the accuracy by finding appropriate values of corresponding parameters (section 2.1). It is computationally expensive and therefore only a few parallel implementations support this process. Cross-validation contains independent tasks that can be computed in parallel using GPUs [45, 52, 54]. Kernel caching is used to accelerate the cross-validation albeit it needs to be performed carefully to avoid recomputation of the kernel at every iteration of the cross-validation. One can avoid the recomputing of the kernel values by calculating the kernel matrix only once for every fold of the cross-validation [45, 52, 54, 78, 81]. This speeds up the process n times, where n is the number of folds in n-fold cross-validation [45]. Athanasopoulos et al. [45] parallelize the cross-validation process using the combination of CPU and GPU parallelism which results in one order of magnitude faster processing time than using only CPU parallelism. One drawback of the proposed approach is that the algorithm loads the input data on the GPU memory and due to the limited onboard memory on the GPU, a large dataset may not fit in the memory [45, 81].

Speedup. Size of working sets impact training/testing time in decomposition techniques [12, 48, 79]. The standard SMO chooses only two points for the working set. It is not clear that this is the best choice for the GPU-based parallelism. One of the major differences of the work proposed by Liao et al. [79] with the standard SMO-based algorithms is that Liao et al. empirically choose around 16-32 points in the working set. The reason is that the proposed size is large enough to fit in the memory and small enough to accelerate the training process, and it gives a good balance between the accuracy and solution time. Another difference is that a different kernel function called Tanimoto is used which shows a higher classification performance over widely used kernels for applications in molecular fingerprints. The algorithm can solve both classification and regression problems. The sparse format of storing input data, the data movement between GPU and CPU, caching and mainly the change of working set size may describe why [79] achieves better speedups than in [48], even though the latter uses a highly parallel map-reduce model for binary classifications. Going beyond binary classifications, Herrero-Lopez et al. [78] implement the first OVA-based parallel SVM that improves the work proposed by Catanzaro et al. [48] for multi-class classifications and supports dense data formats.

Data storage format. One of the issues in parallel SVMs is that they mostly support one type of data format. For instance, the state-of-the-art SVM implementations in [82] and [48] support only compressed sparse row and dense data formats, respectively. You and Demmel [84] show that employing a unified data format for all

datasets may significantly deteriorate the performance of the corresponding parallel algorithm in terms of storage, computation, and memory bandwidth. In [84], they use auto-tuning techniques to employ different formats for the same dataset. Their approach leads to the speedup of 4 (worse case scenario) to 14 times (best case scenario).

Multi-class classification. A multi-class classification is computationally expensive (section 2) and has been a target of GPU parallelism in decomposition techniques [50, 78, 82]. Herrero-Lopez et al. [78] use OVA to solve the N-class classification through N binary classifications in which a single SMO is solved similar to [48] and N(N× P) classifiers are constructed, where N is the number of class and P is the subsets. In [78], different tasks evoke only a single kernel and this causes task-controlling overheads due to many memory transactions between the global memory. In contrast, Lin and Chien [82] use OVO (section 2) in which N(N-1)/2 classifiers are constructed. To overcome the task-controlling overheads in [78], Lin and Chien use a different caching strategy in which every different task evokes a kernel at each iteration. Although this caching approach along with sparse matrix format gives a speedup of around 134 times over Libsvm, the sequential computation of the operations in each task is less efficient than those if computed in parallel. Unlike the previous works, Zhang et al. [50] reduce data transaction for a binary classification by simply moving data from the global memory to the shared memory since data access from the shared memory on GPUs is faster than the global memory. To further reduce the computational time, they use scatter-parallel-reduce-gather reduction method. This simple modification significantly improves the access speed on GPUs, however, the limited shared memory can still be an issue for large working sets.

Prediction. Unlike previous works that focus on the training process, Peng et al. [12] mention that an appropriate size for working sets accelerates the predicting process more than the training process. This is because that the training process is more complicated and its time depends on many factors other than the working set size. Besides, some parts of the training process may not be parallelizable, whereas the predicting process is computationally intensive in which almost all parts are easily parallelizable. Peng et al. [12] improve the work initiated by Liao et al. [79] for parallelization of both training and predicting processes in SVMLight. We highlight the findings, pros, and cons of the parallel decomposition-based SVMs mentioned in this subsection in online table Decomposition.

4.2 Parallel Incremental SVMs

Incremental learning is another approach that has been used for parallel implementations of SVMs [10, 30, 86–95]. The basic idea in an incremental learning is that a partition of data is processed and corresponding SVs are identified. At each step, the SVs of the previous step together with a new data partition are added as inputs of the current step. The process is terminated when no partitions of data are left. Incremental learning can be used for on-line SVM training when new data are added [9]. Note, one may consider online learning as a solution for incremental learning [92] and in a few cases, they have been used interchangeably [86]. Incremental learning has potential to solve problems regarding scalability and memory for large-scale problems since it divides large samples into several subsets, each of which fits into the main memory [9, 94]. The main focus line considered in incremental learning is memory handling for large-scale problems.

4.2.1 Parallel Incremental SVMs Using SMP. An incremental learning approach has been used for data that arrives in a streaming fashion. To our knowledge, the algorithm proposed by Matsushima et al. [89] is the only SMP-based incremental learning of SVM for a linear classification. They use POSIX multi-threaded programming for streaming data. The algorithm identifies insignificant data, i.e., non-SVs, as data is added. The memory requirements are further reduced by processing data in blocks and by trading off the file I/O using data expand on the fly; i.e., features are generated on demand. Furthermore, data is cached using software that provides a thread-safe in-memory hash table called Kyoto Cabinet [96]. One drawback is that the proposed algorithm requires multiple passes through the data to achieve the optimal performance. This may cause considerable overhead, although after one or two passes a close to the optimal performance is achieved.

4.2.2 Hybrid Parallelism. SMP may be unable to handle multi-class classifications due to the limited memory, although incremental learning to some extent handles the limited memory issues by processing a partition of data at a time. To overcome the limited memory issues, Doan et al. [30] use a hybrid of shared and distributed memory parallelism for implementing a multi-class classification problem. This is because using only distributed memory via MPI requires loading the whole subset into the memory for the learning process, and this may not be possible for very large subsets. They add SMP to avoid loading the whole subset for each MPI process. The algorithm handles the class imbalance problem that happens when the number of examples in classes is very unbalanced. They overcome the problem using a balanced bagging incremental approach in which the number of samples in the majority class is reduced, known as undersampling.

4.2.3 Parallel Incremental SVMs using Distributed HPC Architectures. Incremental learning of SVM has been performed in parallel using distributed HPC architectures [87, 90, 91, 95].

Scalability. One of the effective strategies for training large SVM problems is using row- or column-based distributed algorithms in which data is split into blocks of rows or columns and then the corresponding matrix multiplications are computed incrementally in parallel. Do and Poulet [90] study both row- and column-based block distribution of data. The row-based approach has linear dependencies on the number of examples, the number of machines, and a second order depending on the number of dimensions. Therefore, the row-based approach is suitable for a large number of samples with a small enough number of dimensions. This is opposite for the column-based approach. To adapt the row-based algorithm in [90] to solve problems with high dimensional input space, Sherman-Morrison-Woodbury (SMW) [97] formula has been applied in which the inverse of a large matrix is computed in an efficient and inexpensive way that reduces memory requirements and the runtime. To our knowledge, [90] is one of a few algorithms that can handle the biggest samples reported in the experiments.

Incremental learning can be further accelerated by avoiding re-training old data when a new data is added [86, 87], i.e., the classifier is incrementally updated only when a new point is added. Unlike in [90], Tveit and Engum [87] use a tree-based structure to allow the distributed nodes access their input data efficiently [87, 91]. They use a heap-based tree topology for the efficient data access for linear classifications. The heap-numbering of the nodes creates efficient communications between the nodes in which each node of the tree efficiently calculates the addresses of the corresponding child and parent nodes. The results of experiments regarding the proposed algorithm show that the reading on all nodes in the tree outperforms the reading on only leaf nodes. The algorithm scales linearly with the number of cores. This is because that increasing the increment size leads to increasing speedups and each node deals with heavy processing with minor communications [87].

Scalability for different network topologies. Incremental learning is an efficient technique to reduce the communications between processors by avoiding re-training of old data. In this regard, choosing an appropriate network topology with efficient communications of processors can reduce the overheads. Incremental learning of SVM has been implemented using various networks, namely, a centralized network [94], a fully decentralized network [91], and a strongly connected network [92].

Centralized networks. In this type of networks, distributed nodes solve their associated local SVMs independently and then the resulting local SVs are aggregated in a central processing center. The parallel algorithm proposed by Syed et al. [94] uses this approach, but it does not succeed to obtain the global optimum. Caragea et al. [95] improve upon the work in [94] through sending SVs back from the center node to the distributed nodes and repeat the process until the global optimal solution is achieved. The centralized network may suffer from communication and synchronization overheads if the slave nodes regularly need to communicate with the master.

Decentralized networks. In this type of networks, data are distributed across several nodes, each of which only communicates with its neighboring nodes without needing to communicate with the master or central node. The advantage of a decentralized network is that, if a node, including the master, fails for any reason, the network remains connected, thus the algorithm remains operational [91]. To further reduce overheads,

Alternating Direction Method of Multipliers (ADMM) [91] has been designed to handle exchanging messages only between neighboring nodes (section 4.6.2). To do this, Forero et al. [91] cast a centralized SVM as a set of decentralized optimization sub-problems using consensus constraints. One of the advantages of this approach is that the proposed SVM is fully distributed in which nodes do not exchange training data. This leads to minor communication overhead and good scalability in terms of the number of computing nodes. Note that the topology of the network can affect the efficiency of the algorithm. Although the convergence of this method is guaranteed, poor choices of values for the corresponding parameters have negative impacts on the convergence and the algorithm may require many iterations to obtain the desired result. The algorithm only supports binary classifications and has not been evaluated for large samples.

Strongly connected networks. In this type of networks, each node receives its input data from its ancestor. The received data together with the new data are used to train the local sub-problems and then the resulting SVs are sent to the descendent nodes. For instance, Lu et al. [92] show that their incremental learning on a strongly connected network converges in a few number of iterations and the computing time is independent of the number of SVs at each iteration, thus the algorithm scales to a large size of training samples. One should take into the consideration that increasing the size of the network causes high communication overheads [92].

4.2.4 Parallel Incremental SVMs Using Distributed Big Data Architectures. Map-reduce has been used for the parallelization of kernel computations and it has been combined with incremental learning to reduce memory requirements and accelerate the corresponding processes. For instance, He et al. [86] use incremental learning along with a map-reduce technique to handle large-scale problems in which the computations regarding new data are performed without re-doing the computations regarding the old data. The computational complexity of the proposed algorithm increases linearly in terms of the number of new data, where the size of the new data is much smaller than the size of the training samples. The proposed algorithm does not take the multi-class classifications into consideration and it does not scale to a large number of cores due to the overhead.

4.2.5 Parallel Incremental SVMs Using GPUs. GPUs have been used for the parallelization of incremental learning of SVM [10, 93]. In this regard, one may need to minimize the memory requirements due to the limited memory on GPUs. To do so, one can avoid loading the entire training samples into the main memory and update the solution with a growing set. In each incremental step, a block of rows/columns is loaded into the CPU memory and then the block is copied from the CPU memory into the GPU memory. After all the results are computed in parallel using GPU, they are uploaded from the GPU memory back to the CPU memory. Do et al. [93] use Newton SVM formulation which requires only the solution of a system of linear equations instead of QP. They incrementally compute the corresponding gradient function and the Hessian matrix for each iteration. The algorithm in [93] linearly classifies billions of data points on a standard workstation, if the dimension of the input space is small enough, i.e., less than 10^3. For problems with large training samples and high dimensional input space, one may first use dimension reduction strategies such as Principle Components Analysis (PCA) [98]. However, reducing the dimension of large-scale problems is difficult and computationally expensive. Do et al. in [10] improve the algorithm in [93] by exchanging the Newton SVMs optimization into a least square optimization formulation (LS-SVM) since the proposed LS-SVM converges faster than Newton method, thus leads to better performance. The proposed LS-SVM is 1000 times faster than Libsvm for linear classifying 5 million data points. We show brief comparisons of the parallel incremental SVMs in online table Incremental.

4.3 The Cascade

The cascade [17, 99] is a parallel scheme that has been used for the parallelization of SVMs [99–111] in which SVM sub-problems are trained in different layers. In the first layer, training samples are split into smaller subsets, each of which is individually and independently trained by an SVM sub-problem to find the corresponding SVs. The results of sub-problems are combined pairwise and sent as the input for the next layer of the cascade. This

process continues until only one set of training samples remains. In order to check the global convergence, SVs together with non-SVs in the last layer of the cascade are fed back into the first layer. The cascade finds the global solution if SVs that are fed to the first layer are the same as the SVs coming out of the first layer [17].

4.3.1 Parallel Cascade Using SMP. SMP has been used for the parallelization of the cascade of SVMs [99, 101].

Memory and accuracy. Three points impact the training time and accuracy of the cascade; 1) how a problem can be efficiently divided into smaller sub-problems, 2) how SVs are identified in each layer [99], and 3) how and which feedbacks can be given to get the maximum improvement in accuracy and time [99, 101]. Regarding the second point, Zhang et al. [99] and Hu and Hao [101] filter out insignificant data using the distance mean of samples so that if the distance of a data point from the hyperplane is less than the mean, the point is chosen as an SV [99, 101]. Zhang et al. [99] improve the accuracy of the cascade using various feedbacks. They use alternating feedback strategy in which the feedback is not added to each subset of the first layer and instead, it is added in a crossed way (for more information refer to [99]). Hu and Hao [101] follow the same approach, but using .NET for the parallelization of the algorithm which achieves a superlinear speedup compared to the standard cascade, however, it is unclear that how many passes through the cascade is needed to achieve the desired accuracy.

Scalability. The parallel cascade-based SVMs scale to large numbers of training samples [17] and the required communication between processors is minor since processors only need to exchange the SVs, thus latency is low [51]. Low latency makes the cascade suitable for parallelization using a loosely coupled network of processors [112]. One drawback of the cascade is the poor scalability in terms of the number of processors, i.e., speedup saturates for a small number of processors, around 16 processors. The reason is that one machine should combine all SVs in the last layer, thus overheads are increased [112]. Another drawback is the poor performance of the cascade for highly imbalanced samples [113]. These problems still exist in [99] and [101].

4.3.2 Parallel Cascade Using Distributed HPC Architectures. Distributed HPC architectures have been used for performing the cascade in parallel [102–105, 114].

Accuracy and the number of SVs. The classification accuracy of the cascade can be improved using feedbacks (section 4.3.1). One strategy that has been used is feedbacking in a crossed manner [99, 101, 102]. For instance, Yang et al. [102] add the resulting SVs of each sub-problem of the first layer as feedbacks to the rest of the samples in the first layer. New sub-problems are trained until the last layer of the cascade. This leads to improved accuracy and faster training process compared to the standard cascade. Meyer et al. [100] go a step further and improve the accuracy and minimize communication overheads in [102]. In the proposed cascade, instead of generating one single SVM in the last layer of the cascade, they use a bagging SVM in which SVs regarding one of the sub-problems are chosen as the final results using vote aggregation between the sub-problems. This combination approach is promising for accelerating the cascade process, however, it does not achieve optimal results for all the evaluated datasets and it misses an efficient parameter tuning. Unlike the previous works, Wen et al. [114] improve the accuracy and reduce the number of SVs in the cascade for multi-class classifications. They use a hierarchical approach similar to a 3-layer cascade in which the samples of each class are divided into k partitions and instead of the pairwise combining of resulting SVs, they combine SVs regarding k subproblems. The number of SVs obtained by the algorithm is less than that in the traditional cascade. The training time in [114] is reduced when the value of k increases. But, it is unclear that how large the value of k can be, i.e., how many partitions a dataset can be divided into without degrading the accuracy. This work is one of the few ones that study the impact of the number of partitions in the performance of the cascade.

Scalability. The previously mentioned cascade SVMs may deal with a large number of SVs that may not fit in the memory for large-scale problems. To overcome this problem, the cascade has been combined with approaches such as incremental learning [105] and a divide and conquer scheme [104]. The combination approaches can handle large problems without requiring a large amount of memory. For instance, Du et al. [105] combine a cascade SVM with incremental learning in which the new training data together with the resulting SVs in the last

layer are fed back to the first layer. Their combination approach achieves considerable speedups compared to the standard cascade since the number of SVs in the proposed cascade is fewer than those in the standard cascade. You et al. [104] combine the cascade with a divide and conquer scheme to make the balance between the accuracy and training time. Different from the previous works, they minimize the communication between distributed computing nodes using initial clustering of data in which large samples are divided into subsets using k-means clustering. The algorithm in [104] removes the inter-node communication and this overcomes the scalability issue of the cascade. Consequently, it scales well to large numbers of processors with a good load balance.

Peer-To-Peer environment (P2P). The cascade has a potential for further improvements in terms of communication, computations and memory costs. For instance, Ang et al. [103] introduce the first P2P network for training a cascade SVM in which an upper bound on the communication overhead is identified. They further improve the communication between peers and reduce the overhead of sending SVs from a peer to another peer using a reduced-SVM strategy. The basic idea of the reduced-SVM strategy is that the number of SVs is reduced as much as possible. To do so, Ang et al. [103] generate a small random subset of the training samples as the representative of the entire samples [115]. In the proposed algorithm, the accuracy is not degraded as the number of peers increases and the algorithm scales well with the size of the network.

4.3.3 Parallel Cascade Using Distributed Big Data Architectures. Hadoop implementation of map-reduce has been used for the parallelization of the cascade SVMs [108–110, 116, 117] in which each map function trains a sub-problem and each reduce function combines the resulting SVs from two sub-problems. A map-reduce implementation of the cascade reduces the overhead as the number of mappers increases [108], but it may not balance loads of different computers. Guo et al. [116] propose a solution for load balancing issues using a genetic-based algorithm in a heterogeneous environment. Despite the advantages, the proposed solution is complicated to implement and it may lose performance due to the fact that the computation time in the genetic algorithm may get longer than the time spent on training the sub-problems [118, 119].

Accuracy. One may further reduce the memory requirements and accelerate the training process in the cascade using feature selection/extraction strategies in which important features of samples are selected while the others are discarded [109, 110], however, these strategies may lead to the deterioration of the accuracy [110]. One can improve the accuracy of the cascade by combining it with other strategies (section 4.3.2). For instance, Pouladzadeh et al. [117] combine the cascade with incremental learning in which database is periodically updated to improve the accuracy in each step. They further improve the accuracy using ontology-based enhancement in which an end-user confirms the classification results. This approach has been suitable for small-scale problems.

Multi-class classifications. Map-reduce has been used for parallelization of the cascade for multi-class classifications [108, 109]. The algorithms used for multi-class classifications may require iterative map/reduce tasks, however, Hadoop-based map-reduce may not support iterative algorithms and iterative map/reduce tasks. To overcome this problem, Sun et al. [109] use Twister implementation of map-reduce that supports iterative tasks.

4.3.4 Parallel Cascade Using GPUs. The computational power of GPUs has been used for accelerating the cascade of SVMs for large-scale problems [106, 107, 111]. Data reduction and data chunking are employed to overcome the limited memory on GPUs [107]. The parallel cascade algorithm proposed by Li et al. [107] along with data chunking and data reduction, shows superior performance in terms of training time. The basic idea of data reduction is that insignificant samples are eliminated in order to fit significant ones in the memory. The basic idea of data chunking is that the number of training samples is reduced by grouping them into chunks/groups based on their similarities. In the proposed algorithm in [107], data reduction with a single GPU outperforms, in terms of training time, data chunking with multi-GPUs when the accuracy is not an issue. The proposed algorithm reduces the memory requirements for large-scale problems with slight deterioration of the training accuracy. Unlike the previous works, Tarapore et al. [111] study the impact of different numbers of partitions on the training time, speedups and the accuracy. The results of experiments in [111] show that the number of SVs

may decrease as the number of partitions increases, thus the communication cost between processors gets more dominant than the training time. The speedup increases as the number of partitions increases while maintaining a high accuracy. This trend deteriorates as the number of partitions increases and the speedup is reduced. The reason again can be described as the increase of communication overheads.

Different from the above approaches that use public datasets from well-known repositories, Wimer et al. [106] use parallel cascade SVMs for a specific application, i.e., video-based pedestrian recognition and tracking at intersections. The detection rate of the corresponding algorithm is the same as other similar published articles for the pedestrian detection, but the computation time of their approach is much less than that of others. Online table Cascade summarizes the findings, pros, and cons of parallel cascade SVMs mentioned in this subsection.

4.4 Parallel Interior Point Methods

One of the popular solvers used in parallel SVM implementations is the Interior Point Method (IPM) [11, 29, 62, 120, 121] in which an interior point traverses on the feasible region until reaching the optimal solution of the optimization problem. IPM has been an attractive method for solving SVM problems for reasons as follows. It fits the special formulation of the kernel function in the linear case and it uses the simple structure of the optimization constraint addressed by an SVM problem [27]. The number of iterations in IPM is constant or increases very slowly as the problem dimension grows [29]. IPM has polynomial time complexity and the major computations in IPM involve solving one or two systems of linear equations with constant and predictable structures [29]. The primal-dual IPM is the most effective IPM algorithm in which the inequality constraints are removed using a barrier function and the iterative Newton method is used to solve the linear system corresponding to a Hessian matrix [29]. The computational cost and memory requirements of the standard dual-primal IPM algorithms are $O(n^3)$ and $O(n^2)$, respectively. High computational costs and large memory requirements of the algorithm limits the use of IPM for training large-scale problems. To overcome these problems, approximate matrix factorizations along with parallelization approaches have been used [11, 29, 34, 42, 62, 88, 121–123]. Incomplete Cholesky factorization (ICF) in which the kernel matrix is approximated by low-rank approximations and Kronecker are two factorization schemes that have been used in IPM [88]. ICF and Kronecker have the computational cost of $O(p^2n)$ and $O(2n^2)$, respectively, where p is the reduced matrix dimension after factorization and n is the number of samples [88]. The efficiency of other approximate matrix factorizations that can reduce the memory requirements without significant trading off the accuracy is still an open question.

4.4.1 Parallel IPM Using SMP. Parallelization of IPM using SMP can improve training time, however, one may need to employ strategies to further reduce computational costs for solving large-scale problems. These strategies often confront trade-offs between speedups and accuracy [88].

Accuracy and speedup. Coarse-grained and fine-grained approximate algorithms are two examples of strategies for reducing computational costs. Coarse-grained approximations achieve good speedups at the expense of poor training accuracy, whereas fine-grained approximations achieve good training accuracy at the expense of long training time. In order to find a balance between the accuracy and speedups, Wu et al. [88] combine two approximations in a parallel incremental approximate matrix factorization in which the approximate IPM solution of a coarse-grained factorization initiates the IPM of a fine-grained factorization. The warm start for an IPM algorithm leads to fast convergence, thus high speedups. One can further reduce computational costs in IPMs by only one-time computing of approximate matrix factorizations before IPM iterations are started. After the factorization, only a reduced matrix needs to be stored at each iteration. Wu et al. [88] parallelize the ICF matrix factorization scheme using SMP in which the memory requirement is reduced from $O(n^2)$ to $O(np)$, where p is the reduced matrix dimension after factorization, n is the number of training points and $p \ll n$.

4.4.2 Parallel IPM Using Hybrid Parallelism. The computational capabilities of shared and distributed memory parallelism have been combined for the parallelization of IPM algorithms in which the distributed HPC architecture

handles the communication between processors and SMP handles the computations inside processors. For instance, Woodsend et al. [34] use BLAS routines for computing matrix-vector and matrix-matrix multiplications in the shared memory and they handle communications between the processors using the distributed HPC architecture. The proposed algorithm achieves high classification accuracy compared to LibLinear for linear classifications (LibLinear is an open source library for linear SVMs [124]). A drawback of the proposed algorithm is that it requires loading all the samples into the memory. Therefore the algorithms cannot tackle large-scale problems.

4.4.3 Parallel IPM Using Distributed HPC Architectures. IPM-based SVMs have been performed in parallel using distributed HPC architectures [11, 42, 121–123] in which training samples are distributed across multiple processors. Approximate matrix factorizations have been used to speed up the process [121, 123].

Memory and speedup. One of the computationally expensive tasks in IPM-based algorithms is the calculation of the inverse of a large matrix. SMW formula has been used to compute the inverse of a large matrix in an inexpensive way that reduces memory requirements and the runtime of IPM [42, 121–123]. For instance, Gerts et al. [121] employ SMW for an iterative technique called preconditioned linear conjugate gradient method. This method does not explicitly require the computation of the kernel matrix, but it requires the computation of matrix-vector products. The results of experiments in [121] show that the proposed parallel IPM outperforms the standard IPM algorithms. However, it only supports linear kernels for problems with a limited number of features. In contrast, Woodsend et al. [11] mention that SMW can cause numerical instabilities and to overcome this problem, they use the Cholesky decomposition. The proposed decomposition is applied to all the features at once and this made the memory cache of processors to be used efficiently.

To further reduce memory requirements for large-scale problems, linear algebra operations have been used to exploit the block structures of the matrix addressed by SVMs [11, 34]. In this regard, the order of distributing the matrix across machines has an impact on memory usage, i.e., one should take into the consideration that which of row- or column- wise distributing of the matrix fits the model chosen for the parallelization [42, 123, 125].

Column- and row-based data partitioning. A column-based approach is more suitable for SMP than the distributed memory. The reason is that in the distributed memory parallelism, each machine needs to reach all the training data to perform its corresponding computations and therefore it should store a local copy of all data that is inefficient and memory intensive. Moreover, in the column-based approach, only calculations regarding the inner product can be parallelized [123]. Chang et al. in [123] and [42] propose a parallel IPM-based algorithm using parallel row-based ICF in which only essential data are loaded into multiple machines and each machine performs the corresponding computations in parallel. Thereby, the memory requirements are reduced from $O(n^2)$ to $O(np/m)$ and the computational time is reduced from $O(n^3)$ to $O(np^2/m)$, where p is the reduced matrix dimension after factorization, n is the number of training samples, m is the number of machines and p is much smaller than n [42, 122, 123]. The results show that the computation time of parallel ICF is reduced as the problems size increases since the communication overheads are low. The computation speedup is sublinear due to the unparallelizable step in ICF that has the computation time of $O(p^2)$ [123] and according to Amdahl's law, even a small sequential part can deteriorate the speedup. One should consider that communication overheads are minor for problems that use only a few machines for the parallelization [42, 122, 123].

4.4.4 Parallel IPM Using GPUs. Despite the fact that the computational powers of GPUs make this hardware attractive for solving computationally expensive tasks, GPU-based parallelism is used only in a few IPM-based SVMs [29, 62, 126]. In those algorithms, low-rank matrix approximations and SMW have been the target of GPU parallelism. For instance, Li et al. [29] use GPUs to perform the computationally expensive tasks such as ICF matrix factorization, matrix-matrix and matrix-vector multiplications in parallel. They perform the Cholesky factorization on CPU using the master process in a serial code and optimize the data transfer between the host and the device by allocating a contiguous memory space for transferring the matrix in the host memory. The matrix is copied from the host to the global memory of the device by only one function call, thus communication

and synchronization overheads are reduced as the size of samples increases. The algorithm outperforms the CPU-based parallel IPM, however, it does not take the maximum advantage of new GPU features, e.g., the pinned memory that provides transfer speed and enlarges memory space of the GPU and unified virtual address that provides one address space for CPU and GPU memory. In another work, Li et al. [126] improve the data transfer and memory access speed in [29] by exploiting the heterogeneous hierarchical memory on a CPU-GPU cluster using the dual buffer 3-stage pipeline mechanism and the pinned memory. In the dual 3-stage pipeline stream scheduling, data send, calculation, and receive operations are performed concurrently. We briefly mention the comparisons of the parallel IPM-based algorithms in online table IPM.

4.5 Parallel Kernel Computations

Kernel matrix calculations contain calculating dot products regarding large vectors in linear classifications and a Hessian matrix in non-linear classifications [34, 35]. These computations dominate the total computation time in training and testing/predicting processes [112]. One straightforward approach to accelerate these processes for large-scale problems is to parallelize the kernel computations and kernel matrix factorizations [43, 47, 55, 112, 127, 128].

4.5.1 Parallel Kernel Matrix Using SMP. Kernel matrix calculations contain computationally expensive tasks.

Memory. One can reduce the memory requirements for the kernel matrix calculations using matrix approximations via block diagonal matrices [47, 55] and reduced-set SVMs [43]. In the matrix approximation strategy, the kernel matrix is approximated with block diagonal matrices that can be computed in parallel. Dong et al. [55] propose parallel and sequential optimization steps in a parallel algorithm in which non-SVs are eliminated in the parallel optimization step, thus with only remaining SVs, the training time for the sequential optimization step is reduced. Regarding the scalability, the analysis of the algorithm proposed by Dong et al. shows that the runtime complexity linearly scales with the size of the datasets and the number of classes. Unlike the previous strategy, Diaz et al. [43] use the reduced-SVM strategy to eliminate non-significant samples, i.e., non-SVs. In this method, the kernel matrix is approximated and the training samples are reduced based on the sparse greedy rule in which only candidates with the highest possible error descent are chosen at each iteration. Diaz et al. propose a parallel reduced-set SVMs along with a matrix decomposition and a block matrix scheme for computing the kernel matrix inversion in multiprocessors [43]. The results show that almost twice speedup is achieved by doubling the number of cores, note that the efficiency deteriorates around 10%. This is because launching parallel processes on the increasing number of cores may take longer time than the actual computation time. One should take into the consideration that a parallel reduced-set SVMs using SMP may not perform well for small-scale problems or problems with fewer SVs since the time for launching the tasks for/from different cores may take longer time than computing parallel tasks [43].

Accuracy. One needs to consider that reducing the size of datasets in the previous works may cause class imbalance problems, i.e., the number of samples in one class may get much larger (the majority class) or smaller (the minority class) than those in the other class. To overcome the class imbalance, Severyn and Moschitti [127] use a re-sampling strategy in which examples corresponding to the importance-weights are chosen iteratively, thus no important information is lost as it may in [43, 55]. They define weights using the cost-proportionate rejection re-sampling strategy in which examples from both the majority and minority classes are chosen in the desired proportion. The SVM algorithm proposed by Severyn and Moschitti [127] is a modular algorithm that is easy to be performed in parallel and it uses tree-based kernels. They use a Cutting Plane Algorithm (CPA) [127] in which the constraints of the original optimization problem are replaced with linear combinations of the constraints from the original optimization problem. Although the algorithm is parallel friendly, the kernel evaluations at each iteration are computationally expensive for nonlinear SVMs regarding large problems. To reduce the kernel evaluations in a tree-based kernel, they use an approximate CPA along with DAGs (mentioned

in multi-class classification in section 2) in which a fraction of training examples is chosen and many examples share the common sub-structures in the tree. This reduces the number of kernel evaluations at each iteration from $O(tn^2)$ to $O(tr^2/p)$, where p is the number of processors, r is the number of chosen examples and n is the number of total training examples.

4.5.2 Parallel Kernel Matrix Using Field Programmable Gate Arrays (FPGAs). Kernel computations can be implemented in hardware using FPGAs [57, 129–132]. FPGAs are digital integrated circuits that contain programmable blocks of logic and programmable interconnects between the blocks [133]. The disadvantage is that FPGA may suffer from limited RAM blocks [132].

Low power dissipation. One of the reasons for a growing interest in employing dedicated architectures for computing intensive operations is that those architectures can be designed with the aim of reducing power dissipation. For instance, Graf et al. [131] build more compact circuits for the fixed-point arithmetic. With this choice of arithmetic, faster computations trade off the accuracy, albeit a slight deterioration of the accuracy might be acceptable.

Algorithm adjustments. A dedicated coprocessor consisting of a grid of cores can compute several columns of the kernel matrix in parallel [57]. In this regard, one may need to perform adjustments and modifications to fit the corresponding algorithm for the hardware characteristics. For instance, for the parallelization of a decomposition technique, one may use strategies to reduce the number of iterations at the expense of more cost per iteration and the cost per iteration can be reduced using FPGA at each iteration. In order to take the advantage of FPGAs, one should think about the availability of the cached kernel values and a fast convergence criterion [57].

FPGA versus GPU. Papadonikolakis [132] compares the dedicated high-performance architecture using FPGA to the GPU-based parallelism for training SVMs. The results show that FPGA outperforms GPUs only for datasets that fit in the FPGA RAM blocks and not otherwise. The reason is that GPU needs to transfer data from the device global memory to the CPU's shared memory which may cause high overheads [132].

The parallel implementations of kernel computations using distributed memory parallelism (section 4.4.3), GPU-based parallelism (sections 4.1.4 and 4.4.4) and map-reduce (section 4.1.3 and 4.2.4) have been excluded from this subsection since their main contributions fit better the other sections. We show the brief comparisons, pros and cons of the parallel kernel algorithms in online table ParallelKernel.

4.6 Parallel Distributed Algorithms

A large SVM problem can be performed in parallel by dividing the problem into multiple smaller problems using approaches as follows. One approach is training of the combined results of individually trained sub-problems [27, 39, 69, 118, 134–138]. Other approaches are combining the results of ensembles SVMs [119, 139, 140] and initial clustering of data before the training [41, 139, 141].

4.6.1 Parallel Distributed Algorithms Using SMP. One can use initial clustering of data to fit large-scale problems into the limited shared memory before the training phase starts. To do so, large training samples can be clustered into smaller chunks of similar data, all of which are trained in parallel [141].

Memory. The advantage of initial clustering is that the memory requirements are reduced by reducing the number of data into clusters of data and then each data cluster becomes a representative of a group of samples that have pre-defined similarities. In this regard, k-means clustering has been used to train many local SVMs instead of one global SVM [139, 141]. One may expect that the training time of n clusters using n machines is dropped by the factor of n times, but this may not be achieved due to the overhead. For instance, Shrivastava et al. [141] use parallel k-means clustering in order to use the minimum number of training examples as abstracts of a large dataset. The proposed algorithm achieves a speedup of 3 times using SMP compared to the sequential SVMs, but detailed information is needed to analyze the experiments more clearly, e.g., it is unclear that the algorithm can support non-linear or multi-class classifications.

Accuracy. The results of experimenst in [141] show that there is a trade-off between the number of clusters and the training accuracy. One may control the accuracy by selecting an appropriate starting point for k-means clustering. Do et al. [139] follow the similar approach for parallelizing Libsvm for non-linear and multi-class classifications. Another approach for solving SVMs regarding large-scale problems is to use ensemble SVMs. For instance, Claesen et al. [119] employ ensemble learning to train several SVMs. Unlike in [139, 141], the ensemble SVM [119] gives higher accuracy competitive with Libsvm using ensembles of SVMs with a bagging strategy in which local SVMs are trained on bootstrap subsamples and the results of all local SVMs are aggregated based on majority voting. This approach reduces the training complexity.

4.6.2 Parallel Distributed Algorithms Using Distributed HPC Architectures. Two key points play a significant role in the efficiency of parallel distributed SVMs using distributed HPC architectures. One point is to find an efficient strategy to minimize the overheads with regard to combining the results of sub-problems [142]. The second point is to find an efficient strategy to divide the data into subsets or to distribute data across processors.

Scalability. One strategy to reduce overheads is that the number of computations that need to be sent to the processors is reduced at each optimization iteration. For instance, Bickson et al. [135] reduce the communication overheads for problems with high dimensions by reducing the number of messages sent between processors from $O(n^2)$ to $O(n)$, where n is the number of samples. To do this, they shift from an algebraic to a probabilistic domain using Gaussian distribution. At each iteration, a message which contains only two real numbers is sent to neighboring nodes through mutual edges, if available. The algorithm is the largest parallel implementation of the belief propagation algorithm that scales up to 1024 computing nodes for 150000 data points with comparable accuracy to a well-known SVMs software called SVM^{light}. One drawback is that they use an implementation of MPI, i.e., MPICH2, that lacks the support of asynchronous communications for heterogeneous systems. Besides, the scalability in terms of the number of samples is questionable and it is unclear that whether the algorithm can solve large-scale problems due to the fact that the full kernel matrix needs to be computed.

Divide and conquer. Note that the way of dividing a large dataset into subsets and the way of aggregating results have impacts on accuracy. One can use a divide and conquer strategy to break down a large problem into smaller problems. The basic idea is that multiple local SVMs are trained and the results are combined based on an aggregation scheme [69, 118, 119, 134–137, 139]. A parallel mixture of SVMs [134], parallel modular SVMs [118] and parallel ensemble SVMs [119] are examples of divide and conquer. Collobert et al. [134] use a mixture of SVMs in which each SVM is trained on a part of samples and the results of sub-problems are aggregated using a neural network. Although it leads to high prediction accuracy, the corresponding computation time is longer than the time spent on training the sub-problems [118, 119]. Huang et al. [118] improve the work in [134] using a region-computing modular network to train several SVMs, each of which is only trained on a small subregion of the sample space. This approach is easy to be performed in parallel because of its modular nature and it does not have the overheads and complications of [134] since the neural network is replaced with neural quantizer modules. These modules allow the local SVMs to be fired if their inputs belong to their specialized sub-region, otherwise, the neural quantizer modules inhibit the output of the local SVMs [118]. In contrast, Claesen et al. [119] use a much simpler aggregation model than in [134] and [118] with competitive performance (section 4.6.1).

Accuracy. Another approach to partition data is to randomly partition and distribute the data in order to solve class imbalance that happens when the local data contain mostly the same label [143]. Qiu and Lane [142] distribute training samples across processors using a dimension-wise data partition in a distributed memory parallelism. This strategy is simpler than the previous ones and it reduces the communication overheads between the nodes since it follows the memory access pattern in the distributed memory parallelism, thus reduces the data transfer. They further reduce the communication overheads using an approximation of the kernel matrix.

ADMM. It is a popular distributed scheme for solving SVM problems in networks of interconnected nodes [144] in which each node has a private cost function and private constraints [91, 113, 143–146]. The goal is to minimize

the sum of all the cost functions with respect to the intersection of all the constraints [144]. Despite the fact that ADMM is a promising scheme for reducing memory requirements, it may suffer from slow convergence and high time complexity [146, 147]. In ADMM, one can reduce the communication between local processors using a coloring scheme of networks [144] and hash table [143]. A coloring scheme of networks is to assign colors to the nodes of networks so that no adjacent nodes have the same color. Coloring controls the order of communications between nodes and the nodes with the same color can asynchronously perform their computations in parallel. For instance, Mota et al. [144] use a generalization of distributed ADMM with a coloring scheme that leads to less communication between nodes than that in the state-of-the-art algorithms. The theoretical explanation of why this algorithm is more efficient than the similar algorithms is still unclear. Different from the work in [144], Zhang et al. [143] use the combination of efficient strategies in which the algorithm integrates random sampling, a warm start of the sub-problems, inexact minimization, normalizing test data, over-relaxation, and cross-validation of multi-class classifications. One requirement of the algorithm is that data should fit in the distributed memory, otherwise one should use batching strategies to do so. This algorithm outperforms similar approaches in [148] and [149] in terms of training time and convergence rate, respectively. In a similar fashion, Deist et al. [146] introduce a systematic multi-centric data sharing framework based on ADMM with the application in personalized medicine in which the patient data privacy is preserved. To do this, SVM models are learned on data from separated and distributed databases using a customized IT infrastructure in which different sites communicate via file-based and asynchronous messaging [146]. Although the ADMM approaches are promising for solving large-scale distributed SVMs, they may suffer from slow convergence, weak global consensus, supporting only linear classifications, and high time cost. To overcome the slow convergence, Wang et al. [150] divide the slave nodes into groups. The local results of the salves are gathered in the corresponding group to update the global result. The group-based ADMM converges faster and saves up to 30% of the total time compared to the non-grouped ADMM [150].

Unlike the previous algorithms that only focus on the linear classifications, Chen et al. [113] combine ADMM with multiple kernel learning [113] for accelerating the parallel implementation of SVM using multiprocessors. In multiple kernel learning, the global SVM problem with multiple kernels is divided into multiple local problems, each of which is optimized with a single kernel in a local processor. The approach in [113] is the first hybrid ADMM and multiple kernel learning implementation that coordinates the communications between processors to obtain the global solution. The algorithm outperforms the standard cascade [17] and SVM ensembles [108].

4.6.3 Parallel Distributed Algorithms Using Distributed Big Data Architectures. Considering memory, these architectures reduce memory requirements by dividing a large training dataset into many smaller subsets. The strategy that divides the dataset into subsets can have an impact on the training/testing accuracy [44, 140]. For instance, if the distribution of samples in each sub-set is very different, the accuracy is degraded [140].

Accuracy. Parallel balanced bootstrapping has been used to improve the classification accuracy in which training samples are re-sampled into sub-sets so that each sample appears the same number of times in all bootstrap sets. For instance, Alham et al. [140] use this strategy to re-sample data for ensembles of SVMs in which multiple weak learners are combined to create a strong learner. The proposed approach only supports binary classifications and it does not balance loads for heterogeneous computing environments.

Memory and speedup. Different from [140] which supports non-linear classifications, Pechyony et al. [151] use block minimization to reduce memory requirements and accelerate map-reduce based parallel distributed algorithms for linear classifications in which a kernel matrix is divided into blocks and the optimization regarding each block is performed in parallel by map functions. At each iteration, all the distributed blocks are computed in parallel by slave nodes and the results coming from slaves are combined in a master node (centralized computing) using averaging and line search strategies. The results of the experiments concerning the scalability focus line regarding a line search strategy show that the algorithm can solve large problems containing training and testing samples around 80 million each, in only 11 minutes. The communication complexity of the proposed algorithm is

independent of the number of training data points. Besides the advantages, the algorithm only conducts linear classifications and the global convergence is not proven.

4.6.4 Parallel Distributed Algorithms Using GPUs. One can take the advantage of GPU parallelism if the algorithm in use is designed for GPU architectures. Two drawbacks of the previous GPU-based SVMs are as follows; 1) they perform the matrix multiplications addressed by the standard SVM algorithms using GPU-based libraries without paying extra attention for modifying the algorithms to better fit the GPU architectures. 2) they often do not support sparse datasets [41]. The problem regarding sparse datasets is that they may not follow the certain pattern of memory access on GPUs and this is a hurdle to get the maximum performance from GPUs. One needs to employ techniques such as coalescing for efficiently accessing data from the memory [36, 41]. In this regard, Cotter et al. [41] propose a novel sparsity clustering that takes the advantages of the sparsity of datasets and GPU parallelism. The sparsity clustering groups the training examples of the similar sparsity pattern thus overcomes the limited memory in GPUs. The algorithm in [41] is specially designed for GPUs and it supports both dense and sparse datasets. At the time of publication, it was the only GPU-based algorithm that supported both binary and multi-class classifications. The algorithm improves the runtime from $O(n^2)$ to $O(nk)$ for a small k, where n is the number of examples and k is the number of active clusters. Active clusters are candidate clusters that are considered for each training example. The results show that further decreasing k can improve the runtime with only minor deterioration of accuracy. In a similar fashion, Codreanu et al. [36] show that changing the uncoalesced memory access to coalesced memory access results in 3 times speedup. Furthermore, they show that the algorithms similar to [41] can achieve around 10-fold speedups by changing the input data structure from arrays of structures to structures of arrays. This is because that structures of arrays fit better the memory access pattern on GPU architectures and it leads to higher memory throughput [36]. The algorithm is the fastest GPU-based SVM for problems with high dimensional input spaces. Note, the scalability has not been discussed in any GPU-based implementations of SVMs. The comparisons of the parallel distributed SVMs are shown in online table DistributedSVMs.

4.7 Other Parallel Optimizations

Gradient-based [69, 137], gradient projection-based [27, 39, 53], Gaussian belief propagation [135], Iterative Re-Weighted Least Squares (IRWLS) [152, 153] and semiparametric [154] algorithms are another types of algorithms used for solving SVMs that have been parallelized to accelerate the training and testing/predicting processes.

4.7.1 Parallel Optimizations Using SMP. SMP has been used for performing IRWLS methods for training SVMs in parallel [152, 153]. IRWLS reformulates the primal optimization addressed by SVMs in the form of weighted least squares that can be independent of slack variables which provide tolerance for missclassifications. IRWLS solves one linear system in every iteration in which the inverse of a large matrix containing kernel evaluations is calculated. This is computationally expensive for large-scale problems. Therefore, Morales and Vázquez [152] use the Cholesky factorization to reduce the kernel evaluations for solving full SVMs in IRWLS. Solving full SVMs for large-scale problems is computationally expensive. To solve this problem, Morales and Vázquez use semiparametric IRWLS in which the complexity of the resulting model is under control. This results in the speedup of the classifications of new examples. In another work, Morales and Vázquez [153] improve their work in [152] using the budgeted IRWLS to control the number of SVs in which a set of basis elements is chosen and the approximation of the weight vector is calculated using sparse greedy matrix approximation and random sampling. Comparing the results of the two papers shows that the first algorithm PSIRWLS outperforms the second algorithm LIBIRWLS for the same datasets. One possible explanation is the efficiency of the Cholesky factorization in PSIRWLS for reducing the kernel evaluations. PSIRWLS outperforms the similar parallel semiparametric SVM called PS-SVM [154] in which quadtrees, a parallel block matrix inversion, instead of Cholesky factorization is used. We show the detailed comparisons in online table IRWLS.

4.7.2 Parallel Optimizations Using Distributed HPC Architectures. These architectures have been the most common parallel architectures used for performing the optimization algorithms in parallel.

Gradient-based algorithms. These algorithms are the standard SVM algorithms that have been parallelized using distributed HPC architectures [69, 137]. Gradient-based algorithms often compute the gradient using the whole training samples and this is a hurdle for solving large-scale problems. To overcome this problem, Zhu et al. [137] propose P-packSVM, a Stochastic Gradient Descent (SGD) algorithm in which the gradient is stochastically approximated using only a single training example. The advantage of P-packSVM is that it handles an arbitrary kernel and develops the parallelism using a distributed hash table. The table stores the key-value pairs and then the computationally heavy tasks are performed in parallel via distributed storage of inputs. Gradient-based SVMs often require a larger number of iterations than those for IPM-based SVMs before convergence happens and this causes higher communication cost for gradient methods. To reduce the communication cost, Zhu et al. [137] use a packing strategy in which the communication overheads are minimized by packing a number of iterations into a single iteration. This reduces the number of times that processors need to communicate by the factor of $O(r)$, where r is the number of iterations packed into a single iteration. Considering memory, packing along with hash table makes the algorithm highly parallelizable with the memory requirements of $O(m/p)$ for each processor, while the memory requirements for PSVM [42] that is IPM-based is $O(m^{3/2}/p)$, where m is the number of input space dimension and p is the number of machines. Considering speedup, Zhu et al. [137] have shown that stochastic gradient-based SVM algorithms outperform the IPM and SMO based SVMs in which the training time for a dataset with 800k samples reduces to only 13 minutes with 95% accuracy, while the parallel IPM-based PSVM proposed by Chang et al. [42] trains the same dataset in 5 hours with 92% accuracy. This shows that packing may compensate a large number of iterations required by gradient-based methods to converge. Other advantages of the algorithm proposed by Zhu et al. is that it scales to a large number of examples, i.e., 8 million data points for multi-class classifications, while PSVM focuses mainly on binary classifications with fewer samples. As mentioned in section 2, the multi-class classification procedure is computationally expensive and to reach considerable speedups, one may need to avoid training on a full dataset. To do so, a parallel bagging SVMs with a sampling strategy has been used. For instance, Nghi et al. [155] propose an under-sampling strategy in which the majority class is re-sampled to get the equal size the same as the other class(s). This approach achieves more than 1000 times speedup compared to Libsvm, but it requires to load the whole training data into memory.

In a different manner, Ferreira et al. [69] ease the parallelization of linear and non-linear SVMs with gradient-based neural networks that only computes the lower triangular matrix addressed by QP in which each processor only computes a part of the lower triangular matrix needed for its local computations. Although the parallel algorithm proposed by Ferreira et al. outperforms Libsvm and SVMlight for datasets with around 50k samples in non-linear and binary classifications, the work by Nghi et al. [155] performs better for large-scale problems. The article lacks the details of implementations for the fairer comparisons with previous algorithms.

Gradient projection-based algorithms. These algorithms are the standard SVM algorithms that have been performed in parallel using distributed HPC architecture [27, 39, 53] in which sub-problems are optimized based on an iterative projection of the gradient. Caching, shrinking, a sparse format for storing data, and block-wise distribution of kernel matrix are some of the efficient approaches used in these algorithms [27, 39]. Although the existing gradient projection-based algorithms are efficient, they suffer from some issues. For instance, the algorithm proposed by Zanghirati et al. [27] is effective only for a specific kernel function, i.e., the Gaussian kernels. In the algorithm proposed by Zanni et al. [39] the speedup deteriorates for the increasing number of processors. This happens due to the increased communication overhead. Gradient projection-based SVMs algorithms are iterative methods and they are based on chunking techniques in which strong data points, i.e., SVs, are retained from chunk to chunk in an iterative manner [138]. At each iteration, the chunks can be computed by multiple processors. If the number of SVs is large, the communication overheads will be increased since SVs from different processors need to be combined to obtain the final result [138, 156]. To overcome the overheads,

Winters et al. [156] reduce the number of SVs for each chunk by forcing some of the corresponding low-value multipliers to zero, i.e., cutting down the SV vector and force some of SVs to be a non-SV. The proposed algorithm handles both the regression and the classification problems, however, it only considers binary classifications.

4.7.3 Parallel Optimizations Using Distributed Big Data Architectures. The process of multi-class classification is computationally expensive (section 2). In this regard, one can extract important features to reduce the data size. Extracting features for large problems, such as large-scale image classifications with 1000-class SVM classifiers, is computationally expensive. To reduce the computational costs, Lin et al. [157] propose a parallel feature extraction using map-reduce in which the Hadoop distributed file system (HDFS) [158] distributes the images through all machines, all of which can perform the extraction tasks independently on the local images. The feature extraction process is easily parallelizable since the extraction task on each machine is independent. After the feature extraction, they use Averaging Stochastic Gradient Descent (ASGD) [157] to train SVMs. Concerning the speedup focus line, adding the averaging scheme to SGD leads to fast computation for large redundant samples. The reason is that the averaging is much simpler to compute than the second-order SGD which requires computing the inverse of a computationally expensive matrix. Lin et al. [157] reduce the traffic for loading a large training dataset by minimizing the file I/O. To do so, the memory is shared on each multi-core machine. The memory sharing enables the algorithm for loading a non-sparse training dataset containing 1.37 terabytes of data and it reduces memory requirements since multiple programs, that train the same data chunk, only once need to load data for a multi-class classification. Besides the advantages, the algorithm in [157] may suffer from communication overheads and the update of the corresponding parameters one by one. To reduce the communications between computing units, Chu et al. [159] modify the optimization problem addressed by SVMs in order to obtain computations that can be performed independently and in parallel. Chu et al. re-write the summation formula addressed by the Batch Gradient Descent (BGD) [159] in a certain summation form in which each piece of the summation can be easily and independently performed in parallel and the results are averaged to form the final results. The advantage of the BGD in the proposed algorithm is that the data are combined in batches, thus the communications between the processors are minimized since the result corresponding each batch needs to be communicated instead of communicating the result of a single data. Concerning the scalability focus line, the proposed algorithm by Chu et al. [159] does not scale to a large number of cores, i.e., it only gets around 13% speedup using 16 cores. Another disadvantage of a BGD is that it uses the whole examples in each iteration and this is a hurdle for solving large-scale problems. Thereby, Tao et al. [160] use a Mini-Batch Gradient Descent (MBGD) [160] in which a part of examples is used in each iteration. Unlike the earlier algorithms, while MBGD is suitable for solving large-scale problems, it may suffer from the curse of dimensionality that causes unbounded linear growth for model size and the update time with data size [160]. To overcome this problem, Tao et al. use a budget maintenance strategy for MBGD to keep the number of SVs under the control by removing some of SVs, this results in constant space and time complexity in each update. Another advantage of the algorithm proposed by Tao et al. is that they use the Spark data processing engine for solving the iterative algorithm in which the working set is saved and cached in memory. This overcomes the problem of reading and writing data repeatedly from the Hadoop distributed file system. Table 2 shows a brief comparison of gradient-based SVMs. We highlight the findings, pros and cons of the parallel optimization algorithms in online table OtherParallelAlg.

5 DISCUSSION

One of the challenges we have faced for comparing parallel SVM implementations is the difficulty of reproducing and replicating the results. Except for a few cases, the source codes are not publicly available or the settings of the experiments are not clarified. Some of the pressing issues are the size or the dimension of samples, user-defined values of parameters, the number of processors, the type of classifications, and the efficiency of algorithms for

Table 2. The comparison of gradient-based approach

Algorithms	Differences	Pros	Cons
SGD-based SVMs	Uses one example in each iteration	Trains only one example	Updates parameters one by one, cannot be applied for large-scale problems
BGD-based SVMs	Uses the whole examples in each iteration	Cumulates the update of the parameters in a batch	unsuitable for large-scale problems
BMBGD-based SVMs	Uses a part of examples in each iteration, Parallelized on Spark	Suitable for large-scale problems, fewer calculation and higher accuracy, solves iterative algorithms, faster than Hadoop, constant space and time complexity per update, higher accuracy compared to SGD	

non-linear classifications, or the type of parallelism. In the following subsections, we summarize and further highlight some of the other challenges in parallel SVM implementations.

5.1 Summary

We survey parallel SVM implementations, including parallel algorithms and architectures that have been used for solving large-scale problems. Our discussion generated from this survey is with respect to the goals of parallelism, including the four mentioned focus lines. Figure 1 shows our categorization of parallel SVM implementations that we have reviewed. We have also briefly surveyed efficient heuristics, including grid search, cross-validation, caching, shrinking, sparse formats, feature extraction/selection, working set size/selection, data movement, data reordering, and memory access pattern in this paper. Table 3 shows a brief comparison of the well-studied SVM algorithms. Table 4 summarizes the characteristics of parallel SVMs with top 10 largest training examples and table 5 summarizes the characteristics of parallel SVMs with top 5 largest speedups reported in the experiments.

5.2 The Dominant Approach

Among the parallel SVMs reviewed in section 4, the parallel decomposition is the dominant approach for parallel implementations of SVM. The reason is that the decomposition methods use only a fraction of input data in the working set. For instance, the standard SMO as the most common decomposition technique has the working set size of 2, i.e., SMO only calculates two rows/columns of the kernel matrix. One drawback of the decomposition methods is that they are inherently sequential due to the dependent computation steps, thus they are not the best option for parallelization. In order to decrease the number of dependent steps, parallel decomposition-based SVMs use large working sets at the expense of increased cost per step. The cost per iteration can then be reduced by parallelizing each step. The size of working sets has an impact on the training time and accuracy (section 4.1). In this regard, parallel decompositions have had different strategies for choosing an appropriate size of the working set, however, there is a lack of agreement on the optimal size and partitioning data on available memory. An empirical study of the efficiency of different working set sizes is needed to better understand possible improvements of decomposition methods in terms of the speed of convergence and the accuracy.

Table 3. The comparison of well-studied SVM algorithms

Algorithms	Differences	Problem Type	Pros	Cons
SGD-based SVMs	Focus on primal optimization	Non-linear and multi-class classifications	It is the fastest for linear SVMs on a single machine, easier to be parallelized	A large number of iterations until convergence, accuracy fluctuates
IPM-based SVMs	Focus on dual optimization	non-linear and binary classifications	Requires few iterations until converges, has the lowest communication cost	The Cholesky factorization lacks theoretical error bound and may be inaccurate for some datasets, slow convergence, difficult to be parallelized
SMO-based SVMs	Focus on dual optimization	Non-linear	Good accuracy, fastest for non-linear SVMs on a single machine	Slow convergence, needs modification to be parallelized

Table 4. Characteristics of parallel SVMs with top 10 largest training examples that are evaluated in the experiments.

Settings	Approach	Tool	#Samples	Reference
Block partitioning, LS-SVM, linear and non-linear	Parallel incremental	DMP[a]	1 billion	Do et al. [90]
Block minimization	Map-Reduce	Hadoop	80 million	Pechyony et al. [151]
Random sampling, inexact minimization, normalization, cross-validation, multi-class	ADMM	DMP	~20 million & 30 million features	Zhang et al. [143]
Caching, dual coordinate descent	Parallel incremental	SMP[b]	20 million	Matsushima et al. [89]
Newton SVM	Parallel Incremental	GPU	10 million	Do et al. [93]
Multi-class, gradient-based, packing, arbitrary kernel, hash table	Parallel Optimization	DMP	8 million	Zhu et al. [137]
LS-SVM	Parallel Incremental	GPU	5 million	Do et al. [10]
Data reordering, SMO	Parallel Kernel Computation	DMP	4 million	Durdanovic et al. [112]
Various shrinking	Parallel Decomposition	DMP	2.3 million	Vishnu et al. [68]
Gradient Projection-based	Parallel Optimization	DMP	2 million	Zanni et al. [39]

[a]DMP denotes Distributed Memory Parallelism
[b]SMP denotes Shared Memory Parallelism

5.3 Four Focus Lines

We have identified four main focus lines that have been investigated in the parallel approaches reviewed in this paper. In this section, we briefly discuss aspects of these focus lines.

Table 5. Top 5 Speedups. Characteristics of parallel algorithms with top 5 largest speedups in the experiments.

Settings	Parallel Approach	Parallel Tool	Speedups	Reference
Multi-class, balanced class, bagging SVMs	Parallel Incremental	Hybrid SMP[a]-DMP[b]	1193× LibLinear, and 732× original algorithm, 160 cores	Doan et al. [30]
Block partitioning, LS-SVM, linear and non-linear classifications	Parallel incremental	GPU	1000×	Do et al. [10]
LS-SVM , column incremental	Parallel incremental	SMP	190× over LibSVM	Do et al. [161]
Data reordering	Parallel Kernel Computation	DMP	100× with 48 machines	Durdanovic et al. [112]
Random sampling, inexact minimization, normalization, cross-validation, multi-class	ADMM	DMP	60× with 8 machines over LibLinear	Zahng et al. [143]

[a]SMP denotes Shared Memory Parallelism
[b]DMP denotes Distributed Memory Parallelism

5.3.1 Memory. One focus line of parallel approaches is reducing the memory requirements for large-scale problems so that data can fit into the available memory. This is particularly important for shared memory and GPU-based memory parallelism due to limited memory or restricted memory access pattern on GPUs. While there are effective approaches to reduce memory requirements (section 2.1 and 4), there is still no clear suggestion for every SVM implementation. Besides, the resulting speedup magnitude and parallelizability of these approaches need further investigation.

Parallel incremental SVMs. Incremental learning among parallel SVM implementations is one of the efficient and promising approaches to handle limited memory restriction on standard workstations. Factors including the size of increments and the dimension of the input spaces impact the efficiency of the implementations (section 4.2.5). The efficiency and scalability of parallel incremental SVMs are still open questions for large-scale problems with high dimensional input spaces. For such problems, although one can use dimension reduction techniques, these techniques are difficult to perform [98].

Parallel IPMs. The large size of input data requires employing approximate matrix factorizations for IPM-based algorithms in order to reduce memory requirements (section 4.4). The efficiency of approximate matrix factorization schemes is studied on sequential IPM-based algorithms in [120]. However, to our knowledge, there is no benchmarking and empirical study of the schemes for parallel settings. A computational comparison of these schemes would be beneficial to identify possibly simple, computationally inexpensive and easily parallelizable schemes for improving the performance of IPM-based algorithms. One suggestion could be a Jacobi factorization since it is easy to compute and parallelize.

The parallel IPM-based SVMs in section 4.4 agree that there is a trade-off between the training/testing time and training/testing accuracy due to approximations. One challenge is the number of samples that can, through the approximation, be reduced without the deterioration of accuracy. Although some suggestions for an optimal size of reduced ICF have been given, they are only optimal for a few scenarios [125]. To our knowledge, there is no clear suggestion for an optimal size of reduced data without trading off the accuracy. It seems that an optimal size depends on the problem size and dimension of input spaces, opening the opportunity for future research.

5.3.2 Speedup. Another focus line of parallel approaches is accelerating the training and testing processes using available parallel architectures. This can be done through two approaches; One is solving the problem through training a single SVM in which only the computationally expensive tasks are performed in parallel. The second approach is to divide the large SVM problem into several smaller SVM sub-problems, all of which are performed in parallel (section 1). On one hand, solving one single SVM problem leads to higher accuracy since the main optimization problem stays unchanged, however, due to the sequential parts of the algorithm, the speedup deteriorates. Another issue is that a single problem-based algorithm may suffer from memory limitations since the algorithm may require computing the sequential parts using the whole training samples (section 4.1 and 4.4). On the other hand, solving multiple SVM sub-problems can solve the memory issues, but since the problem is divided into multiple simpler problems, the original problem is changed and this may cause the deterioration of the accuracy (section 4.3). To our knowledge except for the cascade approach, the efficiency and the possible trade-offs of training a single SVM versus training multiple SVMs have not been explicitly investigated for parallel settings. Albeit, distributed approaches of parallel SVM implementations show a tendency towards training multiple SVMs in which several independent sub-problems are solved in parallel and the computing powers are added on demand.

5.3.3 Scalability. One can consider the scalability both in terms of the number of machines employed for the parallelization and/or the size of samples.

Scalability in terms of the number of machines. Increasing the number of computing machines does not always lead to better performance due to communication and synchronization overheads. For SMP, while there are many strategies for reducing the overheads (section 4), there are very few implementations that investigate synchronization between threads and measure the scalability. For the distributed memory parallelism, although more works take the scalability into the consideration, only a very few parallel SVM approaches have evaluated the scalability for a large number of machines (e.g. [68, 104, 135]). In addition to measuring scalability, it would be beneficial to find a theoretical bound for the scalability in order to find the optimal number of machines usable in parallel. For instance, [125] shows that such theoretical bound exists, albeit, it lacks the proof. To our knowledge, [125] is the only parallel SVM that investigates the theoretical bound for the scalability.

The parallel implementations of SVMs using P2P networks have shown good scalability in terms of the number of peers (section 4.3.2). P2P networks are promising for solving large-scale problems since more peers can be added when a large amount of memory and computing powers are demanded. However, a few parallel SVMs have explored the impact of P2P networks on the efficiency of SVM implementations. A further investigation of efficiency and minimizing corresponding overheads would be beneficial to exploit the computing power of distributed networks.

Scalability in terms of the number of samples. Parallel SVM implementations have considered the scalability (section 4), but only a few parallel implementations have been evaluated for large training or testing samples with/without high dimensional input spaces. Besides, most of the parallel SVMs focus on large training samples rather than large testing samples. Except in incremental learning, very few parallel SVMs use samples that do not fit into the memory. It would be useful to investigate the scalability of parallel implementations regarding both training and testing samples and the dimension of the input spaces for large-scale problems. This would allow us to handle a wide range of datasets without limited size range.

5.3.4 Accuracy. There are many strategies to reduce data in order to fit them into the available memory (section 4.6.2). The benefit comes at the expense of the poor or a slight deterioration of the accuracy. The parallel SVMs reviewed in this paper show a trade-off between the training time and accuracy regarding how much data can be reduced (section 4.4). Besides, parallel SVMs often require updating or re-training the algorithms to iteratively improve the accuracy. The promising models such as a Hadoop implementation of map-reduce may not support algorithms with the iterative manner or the sequential nature (section 4.3.3). The sequential nature

algorithms require updating at each iteration and updating may lead to high communication overheads. Iterative algorithms may require moderate numbers of iterations to reach the global solution. Each iteration is solved using a map-reduce job. Thus many iterations require many map-reduce jobs which are expensive to launch [162]. Therefore, starting up the parallel routines at each iteration is inefficient and if the problem is not large enough, it leads to high overheads. In this regard, it would be useful to implement a map-reduce framework for iterative algorithms and investigate the possible trade-offs between the accuracy and training time.

5.4 Promising parallel approaches

In this section, we mention the parallel approaches that have shown promising performance and efficiency for solving large-scale problems (for detailed information refer to section 4).

5.4.1 Map-reduce. As we discussed in 5.3.1, map-reduce based algorithms have good scalability in terms of the number of machines and samples. Nevertheless, they may not perform well for sequential or iterative nature algorithms. There is an extension of the Hadoop implementation of map-reduce called Twister that supports iterative algorithms [162]. To our knowledge, only few parallel map-reduce-based SVMs including [109] use Twister. Further exploration of map-reduce framework in Twister can be useful for SVM algorithms that solve large-scale problems in an iterative manner. Another point that has not been explored sufficiently is the impacts of the number of mappers and reducers on the performance of parallel SVMs since, after some numbers, the execution time for an increasing number of mappers/reducers is saturated [116]. A further investigation of optimal numbers of mappers and reducers is required to improve the efficiency of a map-reduce framework.

5.4.2 Incremental learning. It can overcome memory limitations for large-scale problems. Incremental learning shows good scalability in terms of the number of samples and the number of machines because of low overheads. In spite of that, good scalability happens to samples with small enough dimensional input spaces [93] and reducing dimensions for large-scale problems is difficult (section 4.2.5). Therefore, it seems that the size of samples is not an issue for the scalability of incremental learning, but the dimension is. Incremental learning can be further explored in order to find possible avenues for solving problems with high dimensional input spaces.

5.4.3 Combination approaches. Parallel incremental learning, map-reduce, the cascade and distributed approaches, particularly ADMM are promising to reduce memory requirements and accelerate the training process for large-scale problems. A combination of these approaches may lead to further improvements since they can complement each other (section 4.1.3, 4.2, 4.3 and 4.6.2). It would be interesting to find out which combinations match and give the optimal results and performance.

One suggestion can be a combination of ADMM with map-reduce for multi-class and non-linear classifications of large-scale problems. ADMM is one of the promising distributed schemes for reducing memory requirements. In spite of that, it suffers from slow convergence and high time complexity [147]. A combination of ADMM with methods that reduce the time complexity and accelerate the convergence rate is beneficial. Map-reduce has this potential and it matches the distributed structure of ADMM.

5.4.4 Network Architecture. The dominant architectures used for parallel SVMs are centralized networks. On these networks, a master node often has the duty of distributing data among slaves, each of which often needs to communicate or to be synced by the master to obtain the final result. One drawback is that communication and synchronization overheads in centralized networks reduce the efficiency of parallel implementations. In contrast, decentralized computing or P2P computing models can minimize the overheads and reduce memory requirements (section 4.2.3). In these models, each node receives a part of data only from neighbors, thus skips communicating with the master. The parallel algorithms proposed by Bickson et al. [135], Fei et al. [136] and Ang et al. [103] show the attractiveness and potentials of P2P models in which large-scale problems can be solved

with a classification accuracy comparable to the centralized model. A further research is required to solve issues such as handling asynchronous communications for decentralized networks.

6 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Parallel computing of SVMs is becoming a necessity for improving the performance of SVMs for big data and already has demonstrated promising results for improving large-scale problems. This survey presents a summary of state-of-the-art techniques and tools used to solve SVMs in parallel. As we outline, still, there is space for further improvement regarding computation time, accuracy, scalability and memory issues due to the immense and increasing size of real-life data requiring judicious choice for end users. Having the existing challenges and trade-offs in mind, designers have a great deal of flexibility in designing and implementing SVMs by taking their goals of parallelism into the consideration. For instance, considerable speedups can be achieved by a slight deterioration of the classification accuracy which might be acceptable in some applications. One important point is to identify the efficient parallel tools, heuristics, and strategies that fit the characteristics of SVM algorithms in mind, otherwise one should be prepared for modifications and manipulations of the corresponding algorithms or strategies to take the maximum advantages of parallelism. In the current trend of parallelizing SVMs, it seems that the parallel implementations of SVM solvers are still not sufficient to handle large-scale problems and their challenges open up directions for future work. Here, we mention some of the potential open questions that result from our review of parallel approaches of SVMs on large-scale problems.

Use of the four focus lines. We have identified the four focus lines of parallelism that have been targeted in the parallel SVM implementations, i.e., memory, speedup, accuracy, and scalability. These four focus lines of parallelism are not independent and there are trade-offs between them. Consequently, there has not been much work addressing all four focus lines at the same time. One direction of the future work is to develop robust algorithms that are capable of addressing all four focus lines of parallelism and their possible impacts. Another aspect is improving the transparency of the field by focussing on the reproducibility and reliability of the experiments. These give the possibility of comparing the performance of the developed algorithms with the existing ones independent of the specific size of data and dimension of the input spaces.

Use of combinational approaches. As shown in table 4, combinational approaches have been successful in handling large samples since the matching techniques improve the possible weaknesses. To take the maximum advantages of parallelism, a future direction likely to be successful would be the development of an integrated framework that combines the complementing and matching techniques and gives the possibility of combining several techniques. A thorough investigation of combining the state-of-the-art techniques can open up new frontiers for solving large-scale problems.

Use of the available modern processor technologies. Modern processors already contain technologies that designers of parallel algorithms can take advantage of. These technologies are employed by high performance and parallel libraries and software, e.g., MKL. Indirectly, using these libraries and software helps the users to improve the performance of developed algorithms. In addition to that, one can directly use modern processor technologies in SVM algorithms to further improve the performance, e.g., in [56, 125, 163]. However, the restrictive design of algorithms is a hurdle and may not allow using these technologies to the full extend [125]. A future research direction can be to implement non-restrictive parallel SVMs that allow using the available modern processor technologies, e.g., SIMD instructions, SSE and AVX.

Use of decentralized computing. Solving large-scale problems requires a large amount of memory which can be provided by adding more computing resources from different physical locations. But, the majority of the parallel SVMs has their main focus on a centralized computing in which the slaves communicate more or less regularly with a master to obtain the final results. This can cause overheads, thus a hurdle for effective parallel implementations. An interesting future research direction is to develop algorithms that are suitable for P2P and

decentralized computing to take the maximum advantages of distributed computing resources without major overheads and loss of performance.

REFERENCES

[1] L.T. Yang and M. Guo. *High-Performance Computing: Paradigm and Infrastructure.* Wiley Series on Parallel and Distributed Computing. Wiley, 2005.

[2] Vladimir Vapnik. *The nature of statistical learning theory.* Springer Science & Business Media, 2013.

[3] Hyeran Byun and Seong-Whan Lee. Applications of support vector machines for pattern recognition: A survey. In *Pattern recognition with support vector machines*, pages 213–236. Springer, 2002.

[4] Zhao Bin, Liu Yong, and Xia Shao-Wei. Support vector machine and its application in handwritten numeral recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 720–723 vol.2, 2000.

[5] Peter Bartlett and John Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. *Advances in Kernel methodsfisupport vector learning*, pages 43–54, 1999.

[6] 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.

[7] Janmenjoy Nayak, Bighnaraj Naik, and HS Behera. A comprehensive survey on support vector machine in data mining tasks: Applications & challenges. *International Journal of Database Theory and Application*, 8(1):169–186, 2015.

[8] R Ravinder Reddy, B Kavya, and Y Ramadevi. A survey on svm classifiers for intrusion detection. *International Journal of Computer Applications*, 98(19), 2014.

[9] G. Wang. A survey on training algorithms for support vector machine classifiers. In *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, volume 1, pages 123–128, Sept 2008.

[10] Thanh-Nghi Do, Van-Hoa Nguyen, and François Poulet. Speed up svm algorithm for massive classification tasks. In *Advanced Data Mining and Applications*, pages 147–157. Springer, 2008.

[11] Kristian Woodsend and Jacek Gondzio. High-performance parallel support vector machine training. In *Parallel Scientific Computing and Optimization*, pages 83–92. Springer, 2009.

[12] Nanbo Peng, Yanxia Zhang, and Yongheng Zhao. Cuda-accelerated svm for celestial object classification. In *Astronomical Data Analysis Software and Systems Xx*, volume 442, page 119, 2011.

[13] HX Zhao and Frédéric Magoules. Parallel support vector machines on multi-core and multiprocessor systems. In *11th International Conference on Artificial Intelligence and Applications (AIA 2011)*. IASTED, 2011.

[14] Dominik Brugger. *Parallel Support Vector Machines.* WSI. Arbeitsbereich Technische Informatik, Universität Tübingen, 2006.

[15] Emmanuel Didiot and Fabien Lauer. Efficient optimization of multi-class support vector machines with msvmpack. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 23–34. Springer, 2015.

[16] Ana Gainaru, Emil Slusanschi, and Stefan Trausan-Matu. Mapping data mining algorithms on a gpu architecture: a study. In *Foundations of Intelligent Systems*, pages 102–112. Springer, 2011.

[17] Hans P Graf, Eric Cosatto, Leon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *Advances in neural information processing systems*, pages 521–528, 2004.

[18] Stephen Tyree, Jacob R Gardner, Kilian Q Weinberger, Kunal Agrawal, and John Tran. Parallel support vector machines in practice. *arXiv preprint arXiv:1404.1066*, 2014.

[19] Evans Data Software Developer surveys 2011-2013. Performance: Ready to Use. https://software.intel.com/en-us/intel-mkl, 2011-2013. [Online; accessed 11-October-2015].

[20] NVIDIA Corporation. cuBLAS. https://developer.nvidia.com/cublas, 2015. [Online; accessed 11-October-2015].

[21] Yunmei Lu, Yun Zhu, Meng Han, Jing (Selena) He, and Yanqing Zhang. A survey of gpu accelerated svm. In *Proceedings of the 2014 ACM Southeast Regional Conference*, ACM SE '14, pages 15:1–15:7, New York, NY, USA, 2014. ACM.

[22] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[23] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[26] Ovidiu Ivanciuc. Applications of support vector machines in chemistry. *Reviews in computational chemistry*, 23:291, 2007.

[27] Gaetano Zanghirati and Luca Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel computing*, 29(4):535–551, 2003.

[28] AUSTIN Carpenter. cusvm: A cuda implementation of support vector classification and regression. *patternsonscreen. net/cuSVMDesc. pdf*, 2009.

[29] Tao Li, Hua Li, Xuechen Liu, Shuai Zhang, Kai Wang, and Yulu Yang. Gpu acceleration of interior point methods in large scale svm training. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 863–870. IEEE, 2013.
[30] Thanh-Nghi Doan, Thanh-Nghi Do, and François Poulet. Large scale visual classification with parallel, imbalanced bagging of incremental liblinear svm. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.
[31] W.W. Hwu. *GPU Computing Gems Emerald Edition*. Applications of GPU Computing Series. Elsevier Science, 2011.
[32] Volkan Vural and Jennifer G Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the twenty-first international conference on Machine learning*, page 105. ACM, 2004.
[33] L.J. Cao, S.S. Keerthi, Chong-Jin Ong, J.Q. Zhang, and H.P. Lee. Parallel sequential minimal optimization for the training of support vector machines. *Neural Networks, IEEE Transactions on*, 17(4):1039–1049, July 2006.
[34] Kristian Woodsend and Jacek Gondzio. Hybrid mpi/openmp parallel linear support vector machine training. *J. Mach. Learn. Res.*, 10:1937–1953, December 2009.
[35] John C Platt et al. Using analytic qp and sparseness to speed training of support vector machines. *Advances in neural information processing systems*, pages 557–563, 1999.
[36] Valeriu Codreanu, Bob Dröge, David Williams, Burhan Yasar, Po Yang, Baoquan Liu, Feng Dong, Olarik Surinta, Lambert RB Schomaker, Jos BTM Roerdink, et al. Evaluating automatically parallelized versions of the support vector machine. *Concurrency and Computation: Practice and Experience*, 2014.
[37] Tatjana Eitrich and Bruno Lang. *Efficient implementation of serial and parallel support vector machine training with a multi-parameter kernel for large-scale data mining*. Citeseer, 2005.
[38] Fernando Pérez-Cruz, Anıbal R Figueiras-Vidal, and Antonio Artés-Rodríguez. Double chunking for solving svms for very large datasets. *Proceedings of learning*, 2004.
[39] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *The Journal of Machine Learning Research*, 7:1467–1492, 2006.
[40] Thomas Serafini, Gaetano Zanghirati, and Luca Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2-3):353–378, 2005.
[41] Andrew Cotter, Nathan Srebro, and Joseph Keshet. A gpu-tailored approach for training kernelized svms. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 805–813. ACM, 2011.
[42] Edward Y. Chang. *PSVM: Parallelizing Support Vector Machines on Distributed Computers*, pages 213–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
[43] Roberto Díaz-Morales, Harold Y Molina-Bulla, and Angel Navia-Vázquez. Parallel semiparametric support vector machines. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 475–481. IEEE, 2011.
[44] Godwin Caruana, Maozhen Li, and Man Qi. A mapreduce based parallel svm for large scale spam filtering. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 4, pages 2659–2662. IEEE, 2011.
[45] Andreas Athanasopoulos, Anastasios Dimou, Vasileios Mezaris, and Ioannis Kompatsiaris. Gpu acceleration for support vector machines. In *WIAMIS 2011: 12th International Workshop on Image Analysis for Multimedia Interactive Services, Delft, The Netherlands, April 13-15, 2011*. TU Delft; EWI; MM; PRB, 2011.
[46] W.C.C. Chu, H.C. Chao, and S.J.H. Yang. *Intelligent Systems and Applications: Proceedings of the International Computer Symposium (ICS) Held at Taichung, Taiwan, December 12 – 14, 2014*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2015.
[47] Jian-xiong Dong, Adam Krzyżak, and Ching_Y Suen. A fast parallel optimization for training support vector machine. In *Machine Learning and Data Mining in Pattern Recognition*, pages 96–105. Springer, 2003.
[48] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on Machine learning*, pages 104–111. ACM, 2008.
[49] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, pages 301–320, 2007.
[50] Xueqin Zhang, Yifeng Zhang, and Chunhua Gu. Gpu implementation of parallel support vector machine algorithm with applications to detection intruder. *Journal of Computers*, 9(5):1117–1124, 2014.
[51] Léon Bottou. *Large-scale kernel machines*. MIT press, 2007.
[52] Qi Li, Raied Salman, Erik Test, Robert Strack, and Vojislav Kecman. Gpusvm: a comprehensive cuda based support vector machine package. *Open Computer Science*, 1(4):387–405, 2011.
[53] Lingfeng Niu and Ya-xiang Yuan. A parallel decomposition algorithm for training multiclass kernel-based vector machines. *Optimization Methods and Software*, 26(3):431–454, 2011.
[54] Qi Li, Raied Salman, Erik Test, Robert Strack, and Vojislav Kecman. Parallel multitask cross validation for support vector machine using gpu. *Journal of Parallel and Distributed Computing*, 73(3):293–302, 2013.
[55] Jian-xiong Dong, Adam Krzyżak, and Ching Y Suen. Fast svm training algorithm with decomposition on very large data sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(4):603–618, 2005.

[56] Yang You, S.L. Song, Haohuan Fu, A. Marquez, M.M. Dehnavi, K. Barker, K.W. Cameron, A.P. Randles, and Guangwen Yang. Mic-svm: Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 809–818, May 2014.

[57] Sriram Venkateshan, Alap Patel, and Kuruvilla Varghese. Hybrid working set algorithm for svm learning with a kernel coprocessor on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2221–2232, 2015.

[58] T. Joachims. Making large-scale svm learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[59] Yunmei Lu, Yun Zhu, Meng Han, Jing Selena He, and Yanqing Zhang. A survey of gpu accelerated svm. In *Proceedings of the 2014 ACM Southeast Regional Conference*, page 15. ACM, 2014.

[60] Thorsten Joachims. Making large scale svm learning practical. Technical report, Universität Dortmund, 1999.

[61] Moreno Marzolla. Fast training of support vector machines on the cell processor. *Neurocomputing*, 74(17):3700–3707, 2011.

[62] Jing Jin, Xianggao Cai, and Xiaola Lin. Efficient svm training using parallel primal-dual interior point method on gpu. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference on*, pages 12–17. IEEE, 2013.

[63] Jeyanthi Narasimhan, Abhinav Vishnu, Lawrence Holder, and Adolfy Hoisie. Fast support vector machines using parallel adaptive shrinking on distributed systems. *arXiv preprint arXiv:1406.5161*, 2014.

[64] João Gonçalves, Noel Lopes, and Bernardete Ribeiro. Multi-threaded support vector machines for pattern recognition. In *Neural Information Processing*, pages 616–623. Springer, 2012.

[65] Pengfei Chang, Zhuo Bi, and Yiyong Feng. Parallel smo algorithm implementation based on openmp. In *System Science and Engineering (ICSSE), 2014 IEEE International Conference on*, pages 236–240, July 2014.

[66] Tatjana Eitrich and Bruno Lang. Data mining with parallel support vector machines for classification. In *Advances in Information Systems*, pages 197–206. Springer, 2006.

[67] Hwancheol Jeong, Sunghoon Kim, Weonjong Lee, and Seok-Ho Myung. Performance of sse and avx instruction sets. *arXiv preprint arXiv:1211.0820*, 2012.

[68] Abhinav Vishnu, Jeyanthi Narasimhan, Lawrence Holder, Darren Kerbyson, and Adolfy Hoisie. Fast and accurate support vector machines on large scale systems. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 110–119. IEEE, 2015.

[69] L.V. Ferreira, E. Kaszkurewicz, and Amit Bhaya. Parallel implementation of gradient-based neural networks for svm training. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pages 339–346, 2006.

[70] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, and K.R.K. Murthy. Improvements to the smo algorithm for svm regression. *Neural Networks, IEEE Transactions on*, 11(5):1188–1193, Sep 2000.

[71] F. Aiolli and A. Sperduti. An efficient smo-like algorithm for multiclass svm. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 297–306, 2002.

[72] Peng Peng, Qian-Li Ma, and Lei-Ming Hong. The research of the parallel smo algorithm for solving svm. In *Machine Learning and Cybernetics, 2009 International Conference on*, volume 3, pages 1271–1274, July 2009.

[73] Peng Peng, Qian-Li Ma, and Lei-Ming Hong. The research of the parallel smo algorithm for solving svm. In *Machine Learning and Cybernetics, 2009 International Conference on*, volume 3, pages 1271–1274. IEEE, 2009.

[74] Elad Yom-Tov. A parallel training algorithm for large scale support vector machines. In *Neural Information Processing Systems (NIPS), Workshop on Large Scale Kernel Machines 2005*. Whistler, Canada, 2005.

[75] T. Hazan, A. Man, and A. Shashua. A parallel decomposition solver for svm: Distributed dual ascend using fenchel duality. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[76] Nasullah Khalid Alham, Maozhen Li, and Yang Liu. Parallelizing multiclass support vector machines for scalable image annotation. *Neural Computing and Applications*, 24(2):367–381, 2014.

[77] MPI Forum. Message passing interface forum. http://www.mpi-forum.org/, 2015. [Online; accessed 18-October-2015].

[78] Sergio Herrero-Lopez, John R. Williams, and Abel Sanchez. Parallel multiclass classification using svms on gpus. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 2–11, New York, NY, USA, 2010. ACM.

[79] Quan Liao, Jibo Wang, Yue Webster, and Ian A Watson. Gpu accelerated support vector machines for mining high-throughput screening data. *Journal of chemical information and modeling*, 49(12):2718–2725, 2009.

[80] Krzysztof Sopyła, Paweł Drozda, and Przemysław Górecki. Svm with cuda accelerated kernels for big sparse problems. In *Artificial Intelligence and Soft Computing*, pages 439–447. Springer, 2012.

[81] Qi Li. Fast parallel machine learning algorithms for large datasets using graphic processing unit. 2011.

[82] Tsung-Kai Lin and Shao-Yi Chien. Support vector machines on gpu with sparse matrix format. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 313–318. IEEE, 2010.

[83] J. Vank, J. Michlek, and J. Psutka. A gpu-architecture optimized hierarchical decomposition algorithm for support vector machine training. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3330–3343, Dec 2017.

[84] Yang You and James Demmel. Runtime data layout scheduling for machine learning dataset. In *Parallel Processing (ICPP), 2017 46th International Conference on*, pages 452–461. IEEE, 2017.

[85] Noel Lopes and Bernardete Ribeiro. Gpumlib: An efficient open-source gpu machine learning library. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:355–362, 2011.

[86] Qing He, Changying Du, Qun Wang, Fuzhen Zhuang, and Zhongzhi Shi. A parallel incremental extreme svm classifier. *Neurocomputing*, 74(16):2532–2540, 2011.

[87] Amund Tveit and Havard Engum. Parallelization of the incremental proximal support vector machine classifier using a heap-based tree topology. In *Parallel and Distributed Computing for Machine Learning. The Proceedings of the 14th European Conference on Machine Learning and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2003), Cavtat-Dubrovnik, Cratia*. Citeseer, 2003.

[88] Gang Wu, Edward Chang, Yen Kuang Chen, and Christoper Hughes. Incremental approximate matrix factorization for speeding up support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 760–766. ACM, 2006.

[89] Shin Matsushima, SVN Vishwanathan, and Alexander J Smola. Linear support vector machines via dual cached loops. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 177–185. ACM, 2012.

[90] Thanh-Nghi Do and François Poulet. Classifying one billion data with a new distributed svm algorithm. In *RIVF*, pages 59–66, 2006.

[91] Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. Consensus-based distributed support vector machines. *The Journal of Machine Learning Research*, 11:1663–1707, 2010.

[92] Yumao Lu, Vwani Roychowdhury, and Lieven Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *Neural Networks, IEEE Transactions on*, 19(7):1167–1178, 2008.

[93] Thanh-Nghi Do, Van-Hoa Nguyen, and François Poulet. A fast parallel svm algorithm for massive classification tasks. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 419–428. Springer, 2008.

[94] Nadeem Ahmed Syed, Syed Huan, Liu Kah, and Kay Sung. Incremental learning with support vector machines. 1999.

[95] Cornelia Caragea, Doina Caragea, and Vasant Honavar. Learning support vector machines from distributed data sources. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, page 1602. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[96] FAL Labs. Bukkoji cabinet: a straightforward implementation of dbm. http://fallabs.com/kyotocabinet/, March, 04, 2011. [Online; accessed 02-October-2016].

[97] R. Bekkerman, M. Bilenko, and J. Langford. *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2012.

[98] Xue-Qiang Zeng and Guo-Zheng Li. Incremental partial least squares analysis of big streaming data. *Pattern Recognition*, 47(11):3726–3735, 2014.

[99] Jian-Pei Zhang, Zhong-Wei Li, and Jing Yang. A parallel svm training algorithm on large-scale classification problems. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 3, pages 1637–1641. IEEE, 2005.

[100] Oliver Meyer, Bernd Bischl, and Claus Weihs. Support vector machines on large data sets: Simple parallel approaches. In *Data Analysis, Machine Learning and Knowledge Discovery*, pages 87–95. Springer, 2014.

[101] Mingsheng Hu and Weixu Hao. A parallel approach for svm with multi-core cpu. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, volume 15, pages V15–373–V15–377, Oct 2010.

[102] Jing Yang. An improved cascade svm training algorithm with crossed feedbacks. In *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*, volume 2, pages 735–738. IEEE, 2006.

[103] Hock Hee Ang, Vivekanand Gopalkrishnan, Steven CH Hoi, and Wee Keong Ng. Cascade rsvm in peer-to-peer networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 55–70. Springer, 2008.

[104] Yang You, J. Demmel, K. Czechowski, Le Song, and R. Vuduc. Ca-svm: Communication-avoiding support vector machines on distributed systems. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 847–859, May 2015.

[105] Hongle Du, Shaohua Teng, Xiufen Fu, Wei Zhang, and Yuanfang Pu. A cooperative intrusion detection system based on improved parallel svm. In *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pages 515–518. IEEE, 2009.

[106] Daniel Weimer, Sebastian Köhler, Christian Hellert, Konrad Doll, Ulrich Brunsmann, and Roland Krzikalla. Gpu architecture for stationary multisensor pedestrian detection at smart intersections. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 89–94. IEEE, 2011.

[107] Qi Li, Raied Salman, and Vojislav Kecman. An intelligent system for accelerating parallel svm classification problems on large datasets using gpu. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 1131–1135. IEEE, 2010.

[108] Nasullah Khalid Alham, Maozhen Li, Yang Liu, Suhel Hammoud, and Mahesh Ponraj. A distributed svm for scalable image annotation. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 4, pages 2655–2658. IEEE, 2011.

[109] Zhanquan Sun and Geoffrey Fox. Study on parallel svm based on mapreduce. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 16–19, 2012.

[110] Ke Xu, Cui Wen, Qiong Yuan, Xiangzhu He, and Jun Tie. A mapreduce based parallel svm for email classification. *Journal of Networks*, 9(6):1640–1647, 2014.

[111] N. Z. Tarapore, D. B. Kulkarni, and V. K. Prasad. Implementation of parallel algorithm for support vector machine applied to intrusion detection systems. In *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, pages 179–184, Dec 2016.
[112] Igor Durdanovic, Eric Cosatto, and Hans-Peter Graf. Large scale parallel svm implementation. *Large Scale Kernel Machines*, pages 105–138, 2007.
[113] Zhen-Yu Chen and Zhi-Ping Fan. Parallel multiple kernel learning: a hybrid alternating direction method of multipliers. *Knowledge and Information Systems*, 40(3):673–696, 2014.
[114] Yimin Wen and Baoliang Lu. A hierarchical and parallel method for training support vector machines. In *Proceedings of the Second International Conference on Advances in Neural Networks - Volume Part I*, ISNN'05, pages 881–886, Berlin, Heidelberg, 2005. Springer-Verlag.
[115] Yuh-Jye Lee and Olvi L Mangasarian. Rsvm: Reduced support vector machines. In *SDM*, volume 1, pages 325–361, 2001.
[116] Wenming Guo, Nasullah Khalid Alham, Yang Liu, Maozhen Li, and Man Qi. A resource aware mapreduce based parallel svm for large scale image classifications. *Neural Processing Letters*, pages 1–24, 2015.
[117] Parisa Pouladzadeh, Shervin Shirmohammadi, Aslan Bakirov, Ahmet Bulut, and Abdulsalam Yassine. Cloud-based svm for food categorization. *Multimedia Tools and Applications*, pages 1–18, 2014.
[118] Guang-Bin Huang, KZ Mao, Chee-Kheong Siew, and De-Shuang Huang. Fast modular network implementation for support vector machines. *Neural Networks, IEEE Transactions on*, 16(6):1651–1663, 2005.
[119] Marc Claesen, Frank De Smet, Johan AK Suykens, and Bart De Moor. Ensemblesvm: a library for ensemble learning using support vector machines. *Journal of Machine Learning Research*, 15(1):141–145, 2014.
[120] Anirban Chatterjee, Kelly Fermoyle, and Padma Raghavan. Characterizing sparse preconditioner performance for the support vector machine kernel. *Procedia Computer Science*, 1(1):367–375, 2010.
[121] E Michael Gertz and Joshua D Griffin. Using an iterative linear solver in an interior-point method for generating support vector machines. *Computational Optimization and Applications*, 47(3):431–453, 2010.
[122] Kaihua Zhu, Hang Cui, Hongjie Bai, Jian Li, Zhihuan Qiu, Hao Wang, Hui Xu, and Edward Y Chang. Parallel approximate matrix factorization for kernel methods. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1275–1278. IEEE, 2007.
[123] Edward Y Chang, Hongjie Bai, and Kaihua Zhu. Parallel algorithms for mining large-scale rich-media data. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 917–918. ACM, 2009.
[124] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.
[125] S Tavara, H Sundell, and A Dahlbom. Empirical study of time efficiency and accuracy of support vector machines using an improved version of psvm. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 177. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
[126] T. Li, X. Liu, Q. Dong, W. Ma, and K. Wang. Hpsvm: Heterogeneous parallel svm with factorization based ipm algorithm on cpu-gpu cluster. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 74–81, Feb 2016.
[127] Aliaksei Severyn and Alessandro Moschitti. Fast support vector machines for structural kernels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 175–190. Springer, Springer, 2011.
[128] D. Steinkraus, I. Buck, and P.Y. Simard. Using gpus for machine learning algorithms. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1115–1120 Vol. 2, Aug 2005.
[129] RA Reyna-Rojas, D Dragomirescu, D Houzet, and D Esteve. Implementation of the svm generalization function on fpga. In *International Signal Processing Conference (ISPC), Dallas (US)*, pages 147–153, 2003.
[130] Ian Biasi, Andrea Boni, and Alessandro Zorat. A reconfigurable parallel architecture for svm classification. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 5, pages 2867–2872. IEEE, 2005.
[131] Hans P Graf, Srihari Cadambi, Venkata Jakkula, Murugan Sankaradass, Eric Cosatto, Srimat Chakradhar, and Igor Dourdanovic. A massively parallel digital learning processor. In *Advances in Neural Information Processing Systems*, pages 529–536, 2009.
[132] Markos Papadonikolakis, Christos-Savvas Bouganis, and George Constantinides. Performance comparison of gpu and fpga architectures for the svm training problem. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 388–391. IEEE, 2009.
[133] Clive Maxfield. *FPGAs: instant access*. Newnes, 2011.
[134] Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of svms for very large scale problems. *Neural computation*, 14(5):1105–1114, 2002.
[135] Danny Bickson, Elad Yom-Tov, and Danny Dolev. A gaussian belief propagation solver for large scale support vector machines. *arXiv preprint arXiv:0810.1648*, 2008.
[136] Liu Fei, Zhang Wen-Ju, Yu Shui, Ma Fan-Yuan, and Li Ming-Lu. A peer-to-peer hypertext categorization using directed acyclic graph support vector machines. In *Parallel and Distributed Computing: Applications and Technologies*, pages 54–57. Springer, 2004.

[137] Zeyuan Allen Zhu, Weizhu Chen, Gang Wang, Chenguang Zhu, and Zheng Chen. P-packsvm: Parallel primal gradient descent kernel svm. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 677–686. IEEE, 2009.
[138] Angel Navia-Vázquez and Emilio Parrado-Hernandez. Distributed support vector machines. *Neural Networks, IEEE Transactions on*, 17(4):1091–1097, 2006.
[139] Thanh-Nghi Do. Non-linear classification of massive datasets with a parallel algorithm of local support vector machines. In *Advanced Computational Methods for Knowledge Engineering*, pages 231–241. Springer, 2015.
[140] Nasullah Khalid Alham, Maozhen Li, Yang Liu, and Man Qi. A mapreduce-based distributed svm ensemble for scalable image classification and annotation. *Computers & Mathematics with Applications*, 66(10):1920–1934, 2013.
[141] Naveeen Kumar Shrivastava, Praneet Saurabh, and Bhupendra Verma. An efficient approach parallel support vector machine for classification of diabetes dataset. *J. Computer Applications*, 36(6):19–24, 2011.
[142] Shibin Qiu and Terran Lane. Parallel computation of rbf kernels for support vector classifiers. In *SDM*, pages 334–345. SIAM, 2005.
[143] Caoxie Zhang, Honglak Lee, and Kang G Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *International Conference on Artificial Intelligence and Statistics*, pages 1398–1406, 2012.
[144] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel. D-admm: A communication-efficient distributed algorithm for separable optimization. *Signal Processing, IEEE Transactions on*, 61(10):2718–2723, 2013.
[145] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.
[146] Timo M. Deist, A. Jochems, Johan van Soest, Georgi Nalbantov, Cary Oberije, Sen Walsh, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Andre Dekker, and Philippe Lambin. Infrastructure and distributed learning methodology for privacy-preserving multi-centric rapid learning health care: eurocat. *Clinical and Translational Radiation Oncology*, 4(Supplement C):24 – 31, 2017.
[147] H. Wang, Y. Gao, Y. Shi, and R. Wang. Group-based alternating direction method of multipliers for distributed linear classification. *IEEE Transactions on Cybernetics*, PP(99):1–15, 2016.
[148] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):23, 2012.
[149] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.
[150] H. Wang, Y. Gao, Y. Shi, and R. Wang. Group-based alternating direction method of multipliers for distributed linear classification. *IEEE Transactions on Cybernetics*, 47(11):3568–3582, Nov 2017.
[151] Dmitry Pechyony, Libin Shen, and Rosie Jones. Solving large scale linear svm with distributed block minimization. In *NIPS workshop on Big Learning*, 2011.
[152] Roberto Daz Morales and ffingel Navia Vzquez. Improving the efficiency of irwls svms using parallel cholesky factorization. *Pattern Recognition Letters*, 84(Supplement C):91 – 98, 2016.
[153] Roberto Daz-Morales and ffingel Navia-Vzquez. Libirwls: A parallel {IRWLS} library for full and budgeted {SVMs}. *Knowledge-Based Systems*, 136:183 – 186, 2017.
[154] R. Daz-Morales, H. Y. Molina-Bulla, and ffi. Navia-Vzquez. Parallel semiparametric support vector machines. In *The 2011 International Joint Conference on Neural Networks*, pages 475–481, July 2011.
[155] Doan Thanh Nghi. Parallel, imbalanced bagging power mean svm for large scale visual classification with million images and thousand classes. 2015.
[156] Stephen Winters-Hilt and Kenneth Armond Jr. Distributed svm learning and support vector reduction. *Submitted to BMC Bioinformatics*, 2008.
[157] Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour, Kai Yu, Liangliang Cao, and Thomas Huang. Large-scale image classification: fast feature extraction and svm training. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1689–1696. IEEE, 2011.
[158] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
[159] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.
[160] H. Tao, B. Wu, and X. Lin. Budgeted mini-batch parallel gradient descent for support vector machines on spark. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 945–950, Dec 2014.
[161] Thanh-Nghi Do and Jean-Daniel Fekete. Large scale classification with support vector machine algorithms. In *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*, pages 7–12. IEEE, 2007.
[162] Katarina Grolinger, Michael Hayes, Wilson A Higashino, Alexandra L'Heureux, David S Allison, and Miriam AM Capretz. Challenges for mapreduce in big data. In *2014 IEEE World Congress on Services*, pages 182–189. IEEE, 2014.
[163] Yang You, Haohuan Fu, Shuaiwen Leon Song, Amanda Randles, Darren Kerbyson, Andres Marquez, Guangwen Yang, and Adolfy Hoisie. Scaling support vector machines on modern hpc platforms. *Journal of Parallel and Distributed Computing*, 76:16 – 31, 2015.

PART II
ARTICLE II

Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM

S. Tavara[12], H. Sundell[1], and A. Dahlbom[2]
[1]Department of Information Technology, University of Borås, Borås, Sweden
[2]School of Informatics, University of Skövde, Skövde, Sweden

***Abstract*—We present a significantly improved implementation of a parallel SVM algorithm (PSVM) together with a comprehensive experimental study. Support Vector Machines (SVM) is one of the most well-known machine learning classification techniques. PSVM employs the Interior Point Method, which is a solver used for SVM problems that has a high potential of parallelism. We improve PSVM regarding its structure and memory management for contemporary processor architectures. We perform a number of experiments and study the impact of the reduced column size p and other important parameters as C and γ on the class-prediction accuracy and training time. The experimental results show that there exists a threshold between the number of computational cores and the training time, and that choosing an appropriate value of p effects the choice of the C and γ parameters as well as the accuracy.**

Keywords: Parallel SVM; Processor Technology; Training Time

I. Introduction

High Performance Computing (HPC) tools are promising for improving performance in respect of time efficiency. As more computational power can be spent on less time, this enables the accuracy of results to be improved as well. The importance of utilizing HPC tools has been growing and parallel computing as the underlying method of HPC plays an important role for improving the time efficiency. Message Passing Interface (MPI) is one of the well-known parallel library standards that was originally designed for distributed memory systems, although it can handle shared and hybrid (combination of shared and distributed) memory architectures as well. The advantages of using the MPI library standard is well-known, albeit the efficiency of the parallel processing can be degraded due to data dependency, memory bandwidth, synchronization and communication bottlenecks.

Digital data is growing exponentially and hence analysis and calculation processes regarding big data are becoming computationally expensive. Within this context, machine learning is one of the fields that can take advantage of using HPC tools for improving the performance. Support Vector Machine (SVM) [17] is one of the classification machine learning techniques that has a wide area of applications and has got considerable attention during the last decade. The SVM problem is set up as a minimization problem and can thereafter be solved using classical optimization algorithms. Interior Point Method (IPM) [18] is a popular choice thanks to the high degree of parallelism inherent in it. However, IPM requires computing the inverse of a matrix which is computationally expensive. Besides, in most of cases the coefficient matrix derived from the system is ill-conditioned, meaning that the matrix is either singular or close to singularity which makes the problem computationally unstable. Therefore approximation and preconditioning methods are applied to prevent the ill-conditioning and to reduce the computational costs.

Cholesky Factorization (CF) [20] is one of the techniques that is used for achieving stable numerical solutions. CF factorizes matrix $A \in R^{n \times n}$ into a lower triangular matrix, i.e., $A = LL^T$, where $L \in R^{n \times n}$. Incomplete Cholesky Factorization (ICF) [20] is a truncated form of CF, i.e., $A = \hat{L}\hat{L}^T$, where $\hat{L}$ is a $n \times p$ sparse lower triangular matrix close to L, where p is the rank of $\hat{L}$. In ICF approximation, only p column vectors are calculated which makes this approximation quick and economical to compute since $p \ll n$. However, calculating the appropriate column rank value, p, is non-trivial. A lower value of p degrades the accuracy and a higher value of p increases the computational time. In this paper, we study the trade-off between the class-prediction accuracy and time efficiency for different p settings and different kernel functions. Furthermore, we study the correlation between the choice of p and the hyperparameters C and the γ value, in respect to the effect of the Gaussian and Laplacian kernels on the class-prediction accuracy and the training time.

The advantage of using distributed parallelism as MPI on SVM problems is well-known, although how to choose the appropriate numbers of computational cores is still unclear and non-trivial. In theory, increasing the number of machines from 1 to 10 will enhance the time efficiency 10 times, although this is way too idealistic. The reason for that is due to data communication, memory bandwidth and synchronization between different machines. In this paper, we investigate when the data communication part overtakes the parallel computation part in SVM, i.e., when increasing the number of cores no longer is beneficial. As Chang et al. [20] mention in their paper, due to communication and synchronization overheads, the speed-up is not linearly increased by increasing the number of cores, and after a specific number of cores the time efficiency even degrades.

In this respect, it is interesting to study the existence of a threshold that can give an idea of the appropriate number of cores. We theoretically show that there exists a threshold that suggests a minimum number of computational cores, while the maximum number of cores depends on the machines used.

In the following sections, we briefly describe SVM using PSVM software. We mention important processor technologies and then describe the preparation for experiments including the complexity analysis of SVM algorithm using PSVM. We continue with the experiments and the results and finally we discuss the obtained results.

II. Support Vector Machines

The basic idea in SVM is to search for a bipartite hyperplane that has a furthest possible distance to the closest training data points from both sides of the hyperplane. In order to find the optimal bipartite hyperplane, a simple SVM problem can be formulated as an primal quadratic optimization problem as following,

$$\begin{aligned} min \quad & P(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi \\ s.t. \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad , i = 1, 2, ..., n \\ & \xi_i > 0 \qquad\qquad\qquad\qquad , i = 1, 2, ..., n \end{aligned} \tag{1}$$

Where $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}$ is the vector of observations, y_i is the class label and $y_i \in \{+1, -1\}$, b is the bias parameter, $\Phi(\mathbf{x})$ is the map function that maps the input vector $\mathbf{x}$ to the feature space, ξ_i is a classification error for sample i, and C is the parameter that makes a balance between the classification error and maximum margin.

With the help of Lagrangian multipliers, the primal optimization problem (1) can be reformulated into another optimization problem called dual optimization problem. Lagrangian multipliers relax the constraints of the primal optimization problem and reformulates the problem into a new quadratic optimization problem with simpler constraints as follows:

$$\begin{aligned} min \quad & D(\alpha) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{1}^T\boldsymbol{\alpha} \\ s.t. \quad & \sum_{i=1}^{n} y_i\alpha_i = 0 \qquad , i = 1, 2, ..., n \\ & 0 \leq \alpha_i \leq C \quad , i = 1, 2, ..., n \end{aligned} \tag{2}$$

Where $\boldsymbol{\alpha}$ is Lagrangian multipliers and $\alpha_i \in \boldsymbol{\alpha}$, $\mathbf{1}^T$ is a vector of ones and Q is a matrix of size $n \times n$ where $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. Equation (2) is known as dual equation. We reformulate Q_{ij} as $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ is called a kernel function. The advantage of using kernel function is that we can compute Q with out knowing the map function $\Phi(.)$ explicitly and instead we can choose an appropriate kernel function to calculate Q. Four well-known kernel functions are as follows:

- Linear kernel that is an inner product or dot product of input vector and is used for linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$,
- Gaussian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$,
- Laplacian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma|\mathbf{x}_i - \mathbf{x}_j|)$,
- Polynomial kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + const)^d$, where $const = 0$ for homogeneous polynomial kernels and $const = 1$ for inhomogeneous kernels.

Different mathematical solvers are utilized to solve the primal, the dual, or the primal-dual optimization problem. Interior Point Method (IPM) starts from an initial point located in the interior feasible region and moves towards the optimal point(s) in an iterative manner. One of the advantages of IPM is its high degree of inherent parallelism compared to other solvers. IPM can use approximation methods such as Incomplete Cholesky Factorization (ICF) to approximate the original matrix $Q_{n\times n}$ to a smaller matrix as $H_{n\times p}$, where $p \ll n$. This approach can improve the time efficiency of the computations. In this paper, we have chosen the PSVM software that solves the SVM problem by utilizing IPM solver.

A. Parallel Support Vector Machines

We have conducted our experiments using the Parallelizing Support Vector Machines (PSVM) software that is originally written by Chang et al. [20]. PSVM uses ICF to reduce the problem size and IPM to solve the primal-dual optimization problem (1) and (2). ICF approximates the original $n \times n$ linear system Q to the smaller $n \times p$ linear system H, i.e., $Q \approx H^T H$ where n is the number of samples or instances, p is the reduced column size and $p \ll n$. The parallel ICF (PICF) is computed by a row-based round-robin algorithm and distributed evenly to the machines. The primal-dual problem is then solved by a parallel implementation of IPM and makes use of PICF. All the parallelization is done by utilizing the MPI library standard. Chang et al. claim that based on their empirical results when the reduced column size p is chosen as $\sqrt{n}$, then the error ϵ is negligible, where $trace(Q - H^T H) < \epsilon$. On the one hand they show the class-prediction accuracy obtained with some different values of p smaller than $\sqrt{n}$ in table 1 in their paper [20], albeit on the other hand they do not discuss further the impact of varying the p value for different kernel functions in a SVM problem. Therefore we further study the impact of different value of p on the class-prediction accuracy.

III. Related Works

SVM problems has got considerable attention in the last decade and the optimization problem derived from SVM problems are well studied. Challenges in SVM problems can be mentioned as the need for a large amount of memory for training samples [8][19] and the intense training time [1] when the problem size is large. This gives a motivation to use optimization methods along with HPC tools and parallel programming to involve more computational power to spilt the original problem into smaller sub-problems in order to

fit into memory [19]. Decomposition methods [12][6][16] are one of the highly invested methods in SVM. In the decomposition methods only a subset of variables is updated [16]. Based on this idea software as LIBSVM [4], SVM^{light} [10] have been implemented. Although software as LIBSVM and SVM^{light} using decomposition methods are efficient to solve SVM problems however when the problem size is large their performance is degraded since their computations are done in serial manner.

IV. Technologies for High Performance Computing

In this section, we briefly describe the technologies needed for the study of the PSVM communication and the improvement of the code.

A. Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is an architecture of parallel machines that use multiple processing elements to operate a single instruction on multiple data elements simultaneously. Today most compilers can automatically optimize simple code structures using vectors to take benefit from SIMD instructions. A common mistake is to make data dependent whereby the compiler can't optimize it [14]. In such cases the code needs to be restructured to make use of SIMD. The gain from SIMD is dependent on CPU architecture where the old SSE instructions gives a two fold improvement while the newest AVX with FMA can give up to 8 times improvement.

V. Preparation for Experiments

Before we describe the conducted experiments, we do a study of the SVM algorithm using the PSVM software and point out areas of interest that we have our main focus on. These parts are chosen based on their computational time relative to the total calculation time. The areas we focus on can benefit from HPC improvements which affect the total calculation time. Minor HPC improvements on heavy computational parts can have a large effect on the total computation time when the size of the problem is very large, and if the calculation time is not the issue of interest, we can improve accuracy for the same calculation time.

A. Used Factorizations

CF function calculates the Cholesky factorization of the original matrix A, i.e., CF calculates a lower triangular matrix G such that $A \approx GG^T$. CF solves a linear system by forward and backward substitutions. ICF approximates the original $n \times n$ matrix Q to a smaller $n \times p$ matrix H, i.e., $(Q - H^T H) \leq \epsilon$, where $p \ll n$ and ϵ is the error.

B. Algorithm

The solving process regarding (1) is done in two steps, first create,

$$Q \approx H^T H + \epsilon \tag{3}$$

then solve by IPM which is similar to solve by Newton steps [18]. The detailed information about Newton method used in IPM is mentioned by Boyd [3] and Mehrotra [15].

$$\triangle\boldsymbol{\lambda} = -\boldsymbol{\lambda} + \mathbf{vec}(\frac{1}{t(C-\alpha_i)}) + diag(\frac{\lambda_i}{(C-\alpha_i)})\triangle\mathbf{x} \tag{4}$$

$$\triangle\boldsymbol{\xi} = -\boldsymbol{\xi} + \mathbf{vec}(\frac{1}{t\alpha_i}) - diag(\frac{\xi_i}{\alpha_i})\triangle\mathbf{x} \tag{5}$$

$$\triangle\nu = \frac{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{z} + \mathbf{y}^T\boldsymbol{\alpha}}{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{y}} \tag{6}$$

$$D = \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{7}$$

$$\triangle\mathbf{x} = \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \mathbf{y}\triangle\nu) \tag{8}$$

Minimize $P(\mathbf{w}, b, \boldsymbol{\xi})$ and $D(\boldsymbol{\alpha})$ along $\triangle\boldsymbol{\xi}$ and $\triangle\boldsymbol{\alpha}$ respectively (9)

$$\boldsymbol{\Sigma} = \mathbf{Q} + \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{10}$$

$$\mathbf{z} = -\mathbf{Q}\boldsymbol{\alpha} + \mathbf{1}_n - \nu\mathbf{y} + \frac{1}{t}\mathbf{vec}(\frac{1}{\alpha_i} - \frac{1}{C-\alpha_i}) \tag{11}$$

To compute $\Sigma^{-1}z$ the Sherman-Morrison Woodbury formula is used:

$$\begin{aligned}\Sigma^{-1}z &= (D+Q)^{-1}z \approx (D+HH^T)^{-1}z \\ &= D^{-1}z - D^{-1}H(I + H^TD^{-1}H)^{-1}H^TD^{-1}z \\ &= D^{-1}z - D^{-1}H(GG^T)^{-1}H^TD^{-1}z\end{aligned} \tag{12}$$

The equation above containing Σ^{-1} is the most interesting parts of the algorithm with respect to amount of computations and therefore it is divided up into sub steps as following,

$$E = I + H^TDH \tag{13}$$

$$GG^T = E \tag{14}$$

C. Complexity

By going through the SVM algorithm using the PSVM software, we calculate the complexity of both the computation and the communication. We denote the amount of rows on each CPU by η where η is calculated by the amount of training samples divided by the amount of cores, i.e., $\eta = \frac{n}{k}$. Notice that all equations are solved once per iteration except equation (3) which is only solved once. Equation (3) needs $O(p^3 + \eta p^2)$ computations and $O(log(k)(p+f))$ communication where f is the number of features, p is the amount of columns on each CPU, η is the amount of rows on each CPU, and k is the amount of cores. Equation (4), (5) and (7) are just vector operations and therefore has a complexity of $O(\eta)$ computations. Equation (6) is solved by using the result from (13) and (14). Since the system is solved by backward and forward substitution with GG^T for both Σ^{-1}z and Σ^{-1}y therefore we get a complexity of $O(p\eta + p^2)$ computation and $O(log(k)p)$ communication. In a similar manner we

calculate the corresponding complexities for equation (8), i.e., $O(p\eta + p^2)$ for computation and $O(log(k)p)$ for communication. The line search (9) has a complexity of $O(p^2)$ computations. The equation (13) is a matrix multiplication and therefore has the complexity $O(\eta p^2)$ for computations and $O(log(k)p^2)$ for communication. The last equation (14) which is the CF is done serially on the master and has complexity $O(p^3)$.

Among the above mentioned equations, i.e., equations (3) to (14), we focus mainly on equations (13) and (14), since they are the most computationally expensive functions relative to the total computation time and thus they are more interesting for further study and investigation for potential improvements.

VI. Experimental Design

Our goal is to explore, and study the behaviour of the SVM algorithm regarding high performance computing point of view. In this respect, we choose an exploratory approach to discover new insights regarding SVM algorithm that can affect the class-prediction accuracy and the training time using PSVM. PSVM is implemented by Chang et al.[20]. We improve the PSVM code regarding structure, memory allocation, de-allocation and parallelism point of view as mentioned in section IV-A. We conducted experiments 1) to study the impact of changing hyperparameter C and γ on total training time by considering target class-prediction accuracy, 2) to study the impact of changing the number of columns p on the training time and the class-prediction accuracy using Gaussian and Laplacian kernels and to study the trade-off between the class-prediction accuracy and the time efficiency by changing p settings, 3) to evaluate the existence of a threshold for the appropriate number of computational nodes in order to get the advantage of parallelism as much as possible.

We measure the class-prediction accuracy of the models based on the number of correct predictions among all the correct and incorrect predictions.

$$Accuracy = \frac{T_- + T_+}{T_- + T_+ + F_- + F_+}$$

Where T_+, T_-, F_+ and F_- are true positive, true negative, false positive and false negative respectively.

The reduced number of columns in ICF is denoted by $p = n^r$ where n is the number of samples and r is the reducing ratio between 0 and 1. For all the experiments unless stated otherwise, we use the improved PSVM and the publically available *cod-rna, covertype, webspam* and *url* datasets provided by UCI data repository for machine learning [11] and by Fan et al. [5].

A. Experiment 1: Sensitivity of PSVM Regarding C and γ

One of the challenges of SVM problems using Gaussian and Laplacian kernels is to choose appropriate values for hyperarameter C and γ [8][7], since the class-prediction accuracy and time efficiency [13] are influenced by these parameters. Intuitively, γ means how far the influence of a single training sample is. A large value of γ shows the low influence of a single training sample while a low value of γ shows the high influence. The γ parameter has an inverse relationship with the radius of influence of support vectors [2]. The hyperparameter C makes a balance between the misclassification and the maximum margin. A low value of C makes the decision surface smooth while a large value of C gives freedom to the model to choose more support vectors among samples that results in more precise and accurate classification of the training samples [2]. One of the common way to find a suitable hyperparameter C and γ is cross-validation [9], however finding the best value of these parameters are still unclear.

In the first experiment, we study the impact of C and γ parameters on the sensitivity of training time meaning how much the training time varies by changing the C and γ parameters. In this experiment, we chose C between 0.1, 1, ... , 100000 and chose γ between 0.1, 1, ... , 1000. We conduct this experiment on two datasets, *cod-rna* and *covertype* and we study the total calculation time for training the samples.

B. Experiment 2: Reduced Column p

The reduced number of columns p in ICF has impact on the class-prediction accuracy and the training time and finding an appropriate value for p in ICF is non-trivial and controversial. Larger value of p results in higher class-prediction accuracy but slower total training time while smaller value of p results in fast training of samples but poor class-prediction accuracy. Although Chang et al. [20] suggest $p = \sqrt{n}$ based on their experimental results, however the trade-off between the class prediction accuracy and the time efficiency by changing p has been unclear. It is also unclear how the value of p influences different kernels.

In experiment 2, we study the impact of different p settings on the class-prediction accuracy and the training time. To study the performance of PSVM for different p settings, we divide the second experiment into sub-experiments. In the first sub-experiment, we have chosen a range of different p from $n^{0.3}$ to $n^{0.6}$ for the fixed values of C and γ and we measure the class-prediction accuracy. In the second sub-experiment, we have improved the first sub-experiment by choosing the best C and γ parameters for each p settings and we measure the class-prediction accuracy. In the third sub-experiment, we study the impact of p settings on total training time and training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts. In addition, we compare the training time regarding the original PSVM with the improved PSVM and do a short study of how the changes that we did affect the proportions. We conduct experiment 2 for webspam dataset with 300000 samples and 254 features and covertype datasets with 500000 samples and 54 features. We use both Gaussian and Laplacian kernel functions in this experiment.

C. Experiment 3

Although using HPC tools is promising for higher performance, however due to communication overheads choosing

appropriate number of computational powers such as number of computational nodes are still non-trivial. Based on the Amdahl's law , the maximum speed up of a program using parallel computing with multiple processor is limited regardless of the number of processors.

In experiment 3, we study the relation between the complexities we found in earlier chapter and the actual values when running the datasets. We run the experiment on improved PSVM with 8, 16, 32, 64, 128, and 256 computational nodes and we measure the total training time and the training time regarding E, CF, updating variable. For clarity, we measure the proportional training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts and we also measure the time for communication as the communication in E and the communication for Updating variables.

VII. Results

In this section we present the results of all three experiments and the corresponding sub-experiments.

A. Experiment 1

The results of first experiment are presented in Tables I and II. Table I represents the total time for training 59535 samples with 8 features for cod-rna dataset. In the first experiment, we choose a target class-prediction accuracy inside 10 percentage point of the best accuracy obtained. The red cells in table I shows that for the given C and γ the target accuracy is not achieved. Table I shows no trends between different settings of C and γ and the total training time. As the table shows the lowest total training time is 8 times slower than the highest total training time. The results of the first experiment on covertype dataset is shown in table II and it represents the total training time for 500000 samples with 54 features. In table II the target accuracy for $\gamma = 0.1$, $\gamma = 1$ and $\gamma = 10$ is not achieved. As the table shows the lowest total training time is 5.5 times slower than the highest total training time and same as table I, we do not detect any trends between C and γ and total training time. The kernel function that is used during this experiment is Gaussian kernel.

TABLE I
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COD-RNA DATASET WITH 59535 SAMPLES AND 8 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	3.33	1.83	1.28	1.25	0.73
C=1	1.87	1.23	1.06	0.79	0.68
C=10	9.2	1.09	1.07	0.77	0.7
C=100	9.24	1.35	1.59	1.13	1.04
C=1000	8.91	9.11	8.69	0.99	9.02
C=10000	8.99	8.95	9.18	1.27	8.98
C=100000	9.17	8.96	9.24	1.96	10.41

B. Experiment 2

Figure 1 is related to the first sub-experiment in experiment 2 and it shows the class-prediction accuracy with respect to different column sizes. We have selected C and γ parameters as $C = 64$ and $\gamma = 2$ mentioned by Hsieh, Si and Dhillon [8] for webspam dataset. As figure 1 shows by increasing the number of columns, p from $n^{0.3}$ to $n^{0.5}$, the class-prediction accuracy degrades drastically, where for $p = \sqrt{n}$ the accuracy reaches it's lowest value, by increasing the value of p more than $n^{0.5}$, the class-prediction accuracy increases. We observe that the class-prediction accuracy is unstable by increasing the number of columns and for the fixed values of C and γ for all p columns.

TABLE II
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COVERTYPE SCALED DATASET WITH 500000 SAMPLES AND 54 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	332.45	300.26	225.48	241.24	133.39
C=1	85.25	96.55	90.96	137.25	95.99
C=10	42.81	55.24	77.88	104.35	106.94
C=100	52.27	63.12	77.5	71.31	135.23
C=1000	68.58	80.54	92.37	42.99	178.69
C=10000	88.54	137.79	171.65	141.97	415.62
C=100000	284.75	178.96	367.56	417.05	414.29

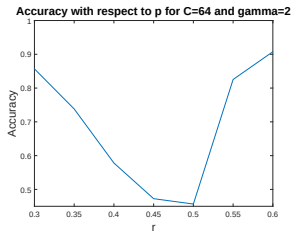


Fig. 1. The class-prediction accuracy with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using C = 64, $\gamma = 2$ and Gaussian kernel function.

Figure 2 shows the second sub-experiment results for both Gaussian and Laplacian kernel functions. This figure gives a better insight that we can observe that replacing C and γ parameters with the best C and γ for each r results in stable class-prediction accuracy where $r = log(p)/log(n)$.

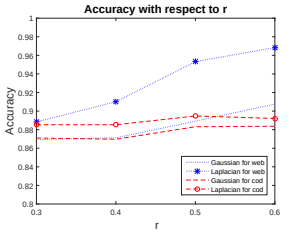


Fig. 2. The class-prediction accuracy with respect to different r ($r = log(p)/log(n)$) for webspam and cod-rna datasets using best C and γ for each r.

Figure 3 shows that the best C and γ stays close when p is changed. Figure 4 is related to the third sub-experiment of experiment 2 and shows the elapsed training time for E, CF, updating variables and other part of SVM algorithm regarding different p settings in webspam dataset. The upper sub-plot shows the elapsed time in seconds and the lower sub-plot shows the proportion time of each parts. The increased proportion of CF and E follows the results of the complexity analysis earlier. The difference between original and the improved PSVM can be seen in figure 5. As figure 5 shows the elapsed time regarding parts E and CF decreases in improved PSVM compared to the original PSVM when r increases.

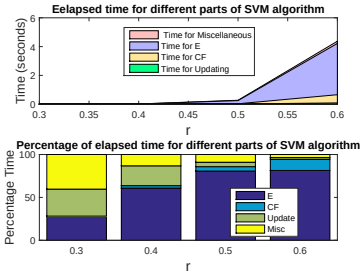


Fig. 4. The elapsed time for different parts of SVM algorithm with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using Laplacian kernel function and best C and γ for each p.

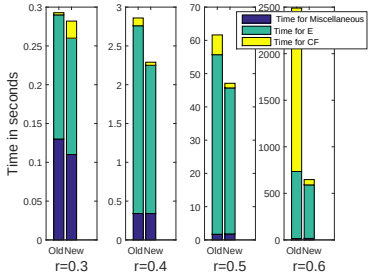


Fig. 5. The comparison between the original PSVM (Old) and the improved PSVM (New) software with respect to r.

C. Experiment 3

Figure 6 shows the training time and the time for calculating E, CF, updating variables of the SVM algorithm with respect to the number of computational nodes for covertype dataset. The upper sub-plot shows how the corresponding training time decreases by increasing the number of nodes from 8 to 64 while increasing the number of nodes more than 64 nodes does not show further improvement in training time. The lower sub-plot gives a better insight about the proportion of training time in E, CF, updating variables, ICF, and other parts of PSVM algorithm for covertype dataset along with communication time for E and communication time for updating variables. With this dataset, 128 nodes were enough to reach the threshold predicted in the complexity analysis.

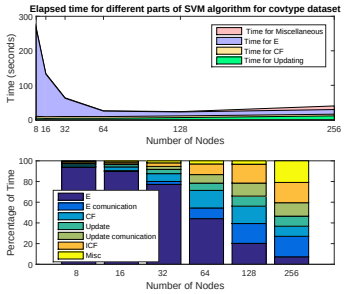


Fig. 6. The elapsed time for different parts of SVM algorithm with respect to the number of nodes, N for covtype dataset with 500000 samples and 54 features. Note the large proportion of communication at 128 nodes.

Figure 7 shows the same experiment as above for URL dataset. The upper sub-plot in figure 7 shows how the corresponding training time decreases by increasing the number of nodes from 8 to 128 while increasing the number of nodes more than 128 shows not remarkable improvement in the training time. The lower sub-plot gives a better insight about the proportion of training time in E, communication in E, CF, update variables, communications for updating variables, ICF, and other parts of the PSVM algorithm for URL dataset.

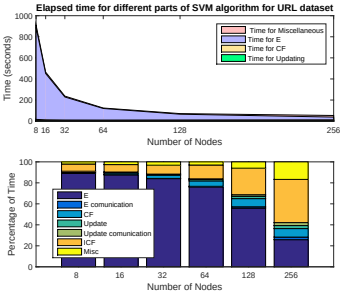


Fig. 7. The elapsed time for different parts of SVM algorithm with respect to the number of nodes N for URL dataset with 2150000 samples and 3231961 features.

VIII. Conclusion

In experiment 1, we study the impact of C and γ parameters on the training time considering the target accuracy. We did not find any interesting trend in the training time by changing these parameters. However this experiment helped us to get better insight about experiment 2 where we observed an improvement in the class-prediction accuracy while

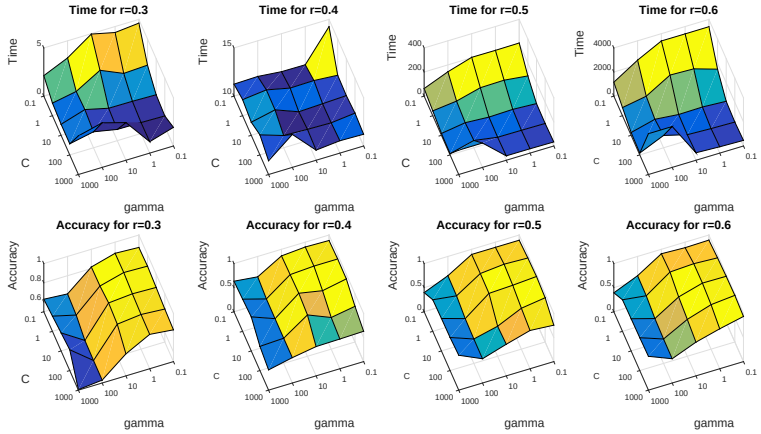


Fig. 3. The training time and accuracy with respect to different column numbers (p) for cod-rna considering different C and γ settings for Laplacian kernel.

increasing the number of columns. The result of experiment 2 showed that choosing appropriate value of p affects the choice of C and γ, this is clearly shown by figure 1. The common way to choose C and γ is done by cross-validation. The result of experiment 2 showed that choosing the best values for C and γ for special column size p is not necessarily the best value for another p. The complexity analysis for CF did predict a fast growth of calculation time for CF when p is increased which could clearly be seen by experiment 2. In figure 5, the original software was tested against the improved software for different p and showed 4 times improvement of performance by our modification on PSVM at large p because of the CF calculation. Already at smaller p we got an 20% improvement on the calculation of E. In experiment 3, we showed the existence of a threshold between the training time and the number of cores as predicted in a complexity analysis.

IX. Acknowledgements

Our experiments were using the Triolith system from National Supercomputer Center (NSC) at Linkoping University.

References

[1] A fast parallel optimization for training support vector machine. In Petra Perner and Azriel Rosenfeld, editors, *Machine Learning and Data Mining in Pattern Recognition*, volume 2734 of *Lecture Notes in Computer Science*. 2003.

[2] scikit-learn developers (BSD License) 2010 2014. RBF SVM parameters. http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, 2015.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[5] Chih-Chung Chang, Chih-Jen Lin, and Rong-En Fan. LIBSVM data: Classification, regression, and multi-label. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, 2015.

[6] Fu Chang, Chien-Yang Guo, Xiao-Rong Lin, and Chi-Jen Lu. Tree decomposition for large-scale SVM problems. *The Journal of Machine Learning Research*, 11:2935–2972, 2010.

[7] Asdrúbal López Chau, Xiaoou Li, and Wen Yu. Support vector machine classification for large datasets using decision tree and fisher linear discriminant. *Future Generation Computer Systems*, 36:57–65, 2014.

[8] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. A divide-and-conquer solver for kernel support vector machines. *arXiv preprint arXiv:1311.0914*, 2013.

[9] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2003.

[10] T. Joachims. Making large-scale SVM learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[11] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[12] Chih-Jen Lin. On the convergence of the decomposition method for support vector machines. *Neural Networks, IEEE Transactions on*, 12(6):1288–1298, 2001.

[13] Gaëlle Loosli and Stéphane Canu. Comments on the "core vector machines: Fast SVM training on very large data sets". *J. Mach. Learn. Res.*, 8:291–301, May 2007.

[14] S. Maleki, Yaoqing Gao, M.J. Garzaran, T. Wong, and D.A. Padua. An evaluation of vectorizing compilers. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 372–382, Oct 2011.

[15] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[16] John Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methodssupport vector learning*, 3, 1999.

[17] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson International Edition. Pearson Addison Wesley, 2006.

[18] Tamás Terlaky. *Interior point methods of mathematical programming*, volume 5. Springer Science & Business Media, 1996.

[19] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *The Journal of Machine Learning Research*, 7:1467–1492, 2006.

[20] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. Parallelizing support vector machines on distributed computers. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. Curran Associates, Inc., 2008.

PART III
ARTICLE III

Effect Of Network Topology On The Performance Of ADMM-based SVMs

1st Shirin Tavara
Information Technology
University of Borås
Borås, Sweden
shirintavara@hotmail.com

2nd Alexander Schliep
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
alexander.schliep@cse.gu.se

***Abstract*—Alternating Direction Method Of Multipliers (ADMM) is one of the promising frameworks for training Support Vector Machines (SVMs) on large-scale data in a distributed manner. In a consensus-based ADMM, nodes may only communicate with one-hop neighbors and this may cause slow convergence. In this paper, we investigate the impact of network topology on the convergence speed of ADMM-based SVMs using expander graphs. In particular, we investigate how much the expansion property of the network influence the convergence and which topology is preferable. Besides, we supply an implementation making these theoretical advances practically available. The results of the experiments show that graphs with large spectral gaps and higher degrees exhibit accelerated convergence.**

***Index Terms*—ADMM, SVMs, Expander Graphs, Distributed Optimization, Convergence**

I. Introduction

Distributed optimization methods are key for solving large-scale machine learning problems due to the exponential growth of digital data. Serial methods are no longer capable of solving today's large problems due to the lack of scalability. Even centralized parallel optimization methods may perform poorly due to communication overheads. Therefore, decentralized distributed optimization methods play an important role in solving problems with big data. Alternating Direction Method Of Multipliers (ADMM) is one of such successful distributed methods since it is robust, distributedly parallelizable, and it has convergence guarantees. However, even ADMM may suffer from slow convergence in specific circumstances [1, 2]. In decentralized ADMM through a network, distributed agents/nodes with the knowledge of local data solve local optimization problems and only communicate with their neighboring nodes with the common goal of reaching consensus. Franca and Bento [2] point out that the network topology has impact on the convergence rate of ADMM in the context of a specific consensus problem. This leads to the natural question, whether their observation also arises in different circumstances. In this paper, we investigate the impact network topology has on ADMM-based Support Vector Machines (SVMs) [3]. In particular, we investigate how much the expansion property and connectivity of the network influence the convergence and which topology is preferable. We also supply an implementation making these theoretical advances practically available. The outline of the paper is as follows. We briefly discuss the basics of network topology in section II. In section III, we explain expander graphs and their properties followed by a brief summary of SVMs and ADMM in section IV and V. We in section VI describe the method and corresponding materials used in this article. The results of the experiments are presented in VII and we analyze the results in section VIII. Finally, the summary and conclusion of the paper are briefly described in section IX.

II. Network Topology

Network topology and the connectivity of the underlying graph have impact on the performance of network-based distributed algorithms in terms of convergence and the number of iterations until convergence. Such impact is shown for solving linear equations [1] and for ADMM with a specific optimization problem not related to SVMs [2]. In this paper, we investigate the impact of the network topology and the connectivity of the underlying graph on the performance of ADMM-based SVMs in terms of convergence. Consider the network of distributed agents/nodes as a graph $G(V, E)$, where $V = \{1, 2, 3, \ldots, n\}$ is the set of the nodes and $E \subseteq V \times V$ is the set of edges between nodes. We assume that G is an undirected and connected graph and that there are no multiple edges between any two nodes. The network topology of the graph is shown by the corresponding adjacency matrix $A(G) = [a_{ij}]_{n\times n}$, where

$$a_{ij} = \begin{cases} 1 & \text{for}(i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

By definition, the adjacency matrix of an undirected graph is symmetric and has real eigenvalues. The spectral gap/connectivity of a graph is related to the expansion property of the graph. In order to define the spectral gap and connectivity of a network, we first define the Lagrangian matrix corresponding the adjacency matrix $A(G)$. The Laplacian matrix of an adjacency matrix, denoted $L(A)$, and has -1 for a connected pair nodes and the degree of each node on its diagonal, i.e.,

$$l_{ij} = \begin{cases} -1 & \text{for}(i,j) \in E \quad \& \quad i \neq j \\ k_i & \text{for} i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Here, k_i is the degree of node i. The eigenvalues of $A(G)$ satisfy

$$k_{max} = \mu_1^{'} \geqslant \mu_2^{'} \geqslant ... \geqslant \mu_n^{'}, \tag{3}$$

and the eigenvalues of $L(G)$ satisfy

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n \leq 2k_{max}. \tag{4}$$

Here, k_{max} is the largest degree of all nodes.
The algebraic connectivity or the spectral gap of a graph network is related to the non-trivial eigenvalues of $A(G)$ and $L(G)$, i.e., it relates to $\mu_2 = max\{\mu_2^{'}, |\mu_n^{'}|\}$ and λ_2 [4]. The first smallest eigenvalue of $L(G)$ is trivial ($\lambda_1 = 0$) and corresponds to the first largest and trivial eigenvalue of $A(G)$, i.e., $\mu_1 = k_{max}$ and in some cases $|\mu_n| = |-k_{max}|$ [5].

A small spectral gap relates to a small number of edges required to be taken away to generate a bipartite graph. In contrast, a large spectral gap relates to non-modularity of the corresponding graph. For details refer to [4].

III. Expander Graphs

A group of well-studied connected graphs are expander graphs, in which any subset expands through all nodes in a robust manner, i.e., any subset of the graph efficiently connects to many nodes. Properties such as effective communication, high- and well-connectivity, and sparseness make expander graphs good choices for designing efficient networks. The expansion property of an expander graph can be defined by the Cheeger or isoperimetric constant [4]. The Cheeger constant shows that whether the graph has bottlenecks, i.e., whether there are two large subsets of vertices connected by only few edges. A large Cheeger constant indicates many edges between the two large subsets of vertices. In contrast, a small constant shows there is a bottleneck between the two subsets of vertices and they are connected with only few edges. The Cheeger constant of graph $G(V, E)$ is denoted as $h(G)$ and it can be defined as follows,

$$h(G) = min_{S \subseteq V, |S| \leqslant \frac{|V|}{2}} \frac{|\partial S|}{|S|}. \tag{5}$$

Here, $\partial S = \{(e, e^{'}) \in E : e \in S, e^{'} \in V \setminus S\}$. The Cheeger constant is related to the spectral gap by Cheeger inequalities, i.e.,

$$\frac{\lambda_2}{2} \leq h(G) \leq \sqrt{2d\lambda_2}. \tag{6}$$

The expansion properties can be enhanced towards increasing the spectral gap. In this regard, d-regular random graphs in which each node is connected to d other nodes are expanders if and only if the corresponding spectral gap is lower bounded [4]. In this paper, we study the impact of d-regular graphs on the convergence performance of the ADMM-based SVMs.

IV. SVMs

SVMs are a set of supervised machine learning techniques developed from statistical learning theory to solve classification and regression problems. The basic idea of SVMs in a simple binary classification problem is to search for a hyperplane that is the farthest to the closest training data points from both sides of the hyperplane. This process has two phases, training and testing. In the training phase, the machine is trained to find a plane that separates the given data samples into two classes. After the machine is trained, the training model is extracted and then the testing phase is carried out. In the testing phase, the SVMs model predicts which class label a new unseen test sample should have [6]. SVMs give a good generalization performance [7] and minimize the upper bound of the generalization error [8]. SVMs have special characteristics that can be used to implement efficient parallel algorithms in terms of time and memory. One characteristic is the sparsity of solutions [9], i.e., the solution is obtained by only a few samples called support vectors that determine the maximum margin separating hyperplane [10]. Another characteristic of SVMs is to perform the nonlinear mapping without knowing the mapping function using predefined functions called kernels for calculating the inner product of mapping functions [10]. Other characteristics of SVMs are the simple structure of SVMs constraints and the definition of the kernel function in a linear case, i.e., the inner product is a simple dot product in a linear case [11]. The optimization problem addressed by SVMs can be written as follows,

$$\begin{aligned} &Primal: \\ &\quad min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad \forall i: \quad y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad , i = 1, 2, ..., N \\ &\qquad \forall i: \quad \xi_i \geq 0 \quad , i = 1, 2, ..., N. \end{aligned} \tag{7}$$

Here $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}$ is a vector of observations, y_i are the class labels with $y_i \in \{+1, -1\}$, b is the bias parameter and $\Phi(\mathbf{x})$ is the map function. In the case that data can be classified by a linear classifier, $\Phi(\mathbf{x}) = \mathbf{x}$, but real-life data cannot always be classified by a linear classifier [8]. In non-linear cases, one can map the data from the input space into a high dimensional feature space using a non-linear transformation, i.e., $\Phi(\mathbf{x})$ maps the input vector $\mathbf{x}$ to the feature space [8]. In the feature space, the data can be linearly separable. Consequently, the dual form of equation (7) is represented in equation (8),

$$\begin{aligned} &Dual: \\ &min \quad D(\alpha) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ &s.t. \quad \sum_{i=1}^{N} y_i\alpha_i = 0 \\ &\forall i: \quad 0 \leq \alpha_i \leq C \quad , i = 1, 2, ..., N. \end{aligned} \tag{8}$$

Here $\boldsymbol{\alpha}$ is Lagrangian multipliers and $\alpha_i \in \boldsymbol{\alpha}$, $\mathbf{1}^T$ is a vector of ones and Q is a matrix of size $N \times N$ and $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$.

V. ADMM

Distributed methods are one of the important approaches for solving large-scale machine learning problems. One of the popular distributed methods is ADMM since it has properties such as robustness, scalability of solving problems with big data, easily distributable and parallelizable. The robustness of ADMM refers often to no requirement of differentiability of the objective function. ADMM guarantees the convergence of convex functions [2] which is the case in SVMs. The convergence rate of ADMM is $O(1/N)$ for convex functions, where N is the number of iterations [12, 13]. Note in practice the convergence rate of ADMM is still not well-understood [2].

The optimization problems of the following form can be solved using ADMM.

$$\begin{aligned} min \quad & f(w) \\ s.t. \quad & w - v = 0 \end{aligned} \tag{9}$$

In order to solve the problem (9) in a distributed manner, one can reformalize the optimization problem in the form of (10). This problem is solvable using ADMM because of the decomposability of ADMM in which each node in a network has own independent objective function and constraints. ADMM can solve consensus optimization in which nodes only communicate with one-hop neighboring nodes. To do this, consensus constraints are defined to force local variables to agree across neighbors. The distributed consensus-based optimization can be formalized as follows,

$$\begin{aligned} min \quad & \sum_{i=1} f_i(w_i) \\ s.t. \quad & w_i - w = 0, \quad i = 1, 2, ..., n. \end{aligned} \tag{10}$$

Here, w is the consensus variable across the neighboring nodes. For detailed information, refer to [14, 15]. In this paper, we have followed the approach given by Forero, Cano and Giannakis in [14] for solving consensus ADMM-based SVMs.

VI. Material and Method

We designed experiments with datasets gathered by LIBSVM Data [16] from several machine learning data repositories such as UCI [17]. Table I shows the datasets with the corresponding number of training and testing instances and features we used in the experiments. In order to study the impact of network topologies on the number of iterations until convergence, we implemented several random d-regular expander graphs. For each graph, upper and lower bounds of the spectral gap are calculated using the formula given by Joel Friedman [18]. The formula is given for the second largest eigenvalue of the adjacency matrix $\mu_2 = max\{\mu_2^{'}, |\mu_n^{'}|\}$. We adapted the formula for the second smallest eigenvalue of the Lagrangian matrix λ_2 as follows. For $\epsilon > 0$,

$$d - 2\sqrt{d-1} - \epsilon \leq \lambda_2 \leq d + 2\sqrt{d-1} + \epsilon. \tag{11}$$

Note $\lambda_i = d - \mu_i^{'}$, $i = 1, 2, \ldots, N$. This holds for every random d-regular graph of size N for sufficiently large Ns. We implemented a special type of regular graphs called Ramanujan graphs. A d-regular graph is Ramanujan if and only if $\mu_2 \leq 2\sqrt{d-1}$ holds, where μ_2 is the second largest eigenvalue of the adjacency matrix of the graph as defined above.

For simplicity, we focus on the second smallest eigenvalue λ_2 of the Laplacian matrix as the spectral gap instead of using the second largest eigenvalue of the adjacency matrix μ_2 since the spectral gap is defined as $\lambda_2 = d - \mu_2^{'}$, where $\mu_2 = max\{\mu_2^{'}, |\mu_n^{'}|\}$.

For a Ramanujan graph, formula (11) appears as follows [19],

$$d - 2\sqrt{d-1} \leq \lambda_2 \leq d + 2\sqrt{d-1}. \tag{12}$$

To run experiments, we used several random regular graphs with different degrees and number of graph nodes depending on the size of training datasets. The first group of graphs, denoted $G1$, consists of d-regular expander graphs with low degrees designed for training small datasets, where $d = \{3, 5, 7, 9, 11, 13, 15\}$ and the second group of graphs, denoted $G2$, consists of d-regular expander graphs with higher degrees for training large datasets, where $d = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

In this paper, we have focused on binary classifications since a multi-class classification can be transformed into several binary classifications using the one-versus-all technique. The classification accuracy of a consensus ADMM-based SVM is influenced by parameters such as ρ in ADMM and JC and γ in SVMs, where J is the number of graph nodes and γ is the RBF kernel parameter. To estimate sufficiently good values of the parameters, we used a grid search with cross-validation. Besides, we use normalizing and standardizing scaling techniques to further improve the classification accuracy if needed. To evaluate the results, we used standard statistics such as true positive/negative rate, positive/negative predictive rate, and accuracy metrics.

To measure the impact of expander graph topology, the algorithm trains the datasets for each d-regular expander graph using shared memory parallel programming. Thereafter, the number of iterations and the corresponding elapsed time are measured until convergence. To stabilize our analysis for some of the datasets, we shuffled the training data 10 times. Each shuffled data are trained and the number of iterations and the elapsed time are measured and then we calculated the average value for the final analysis.

TABLE I
Dataset information

Datasets	Training Points	Testing Points	Features
ionosphere	300	51	34
svmguid	3089	4000	4
phishing	11055	1655	68
a9a	32561	16281	123
ijcnn	35000	14990	22
skinNonskin	37492	5000	3

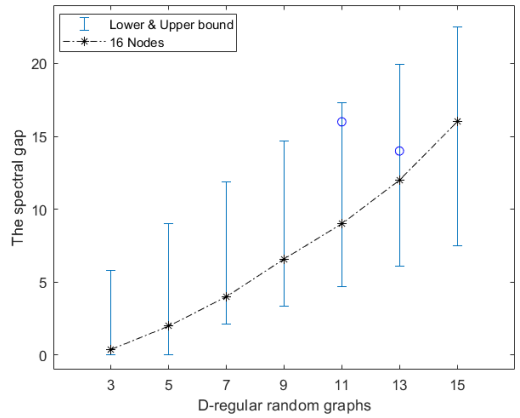


Fig. 1. The upper and lower bounds of the spectral gap for random d-regular expander graphs with 16 graph nodes, where $d = \{3, 5, 7, 9, 11, 13, 15\}$. "*" symbols show the spectral gaps of the d-regular graphs implemented in the experiments. "o" symbols show that a random 11-regular graph can have a higher spectral gap than a random 13-regular graph.

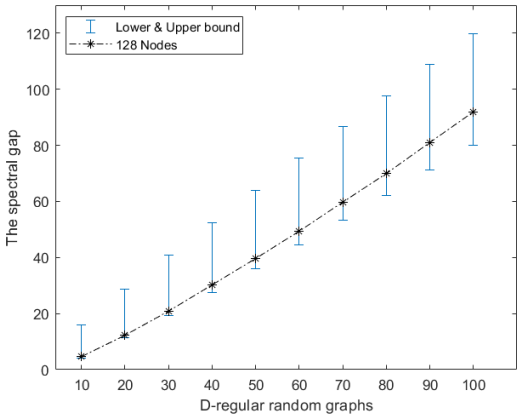


Fig. 2. The upper and lower bounds of the spectral gap for random d-regular/Ramanujan graphs with 128 graph nodes and $d = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

VII. Results

The lower and upper bounds of the spectral gap (the second smallest eigenvalue of the Lagrangian matrix) λ_2 for low degree regular graphs $G1$ is shown in Fig. 1. The same concept for high degree regular graphs $G2$ is shown in Fig. 2. We use the low degree expander graphs for training small datasets and the high degree expander graphs for training larger datasets. The vertical bars show the bounds calculated using formula (12).

In Fig. 1, " $*$ " indicates the spectral gap of the Ramanujan graphs implemented for 16 graph nodes in the experiments. Similarly, in Fig. 2, " $*$ " shows the spectral gaps of the Ramanujan graphs implemented for 128 graph nodes. As the figures show, the spectral gaps of all of the expander graphs we used in the experiments for 16 and 128 graph nodes are inside the allowed bounds. As both figures show, the connectivity of the regular/Ramanujan graphs increases as the degree of the graphs becomes larger.

Fig. 3 shows the number of iterations and the elapsed time for training `ionosphere` and `svmguid` datasets until convergence using group $G1$ of d-regular graphs with 16 graph nodes. As shown in the figure, the number of iterations decreases as the degree of the graph increases. The trend is most visible between 3 to 9 regular graphs for `svmguid` and between 7 to 11 for `ionosphere`. In contrast, the decrease in the number of iterations saturates as the degree of the graphs is close to the number of graph nodes, i.e., 16 nodes in this case. This is visible between 11 to 15 regular graphs for both datasets.

Fig. 4 shows the number of iterations and the elapsed time for training `phishing`, `a9a`, `skinNonskin`, and `ijcnn` datasets using group $G2$ of d-regular graphs with 128 graph

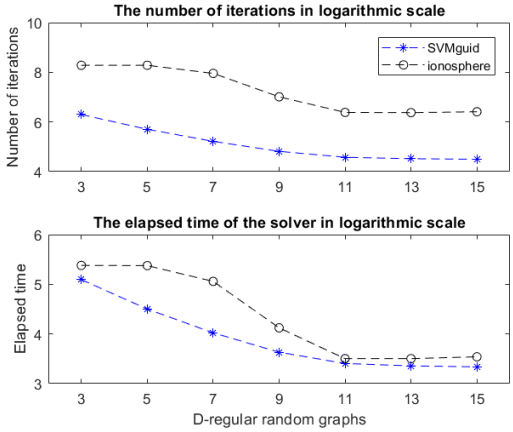


Fig. 3. Impact of d-regular graphs on the number of iterations and time for `svmguid` and `ionosphere` datasets using 16 graph nodes.

nodes. The results highlight the performance improvement in terms of the number of iterations as the spectral gap and the connectivity of the regular graphs increases with the fixed number of nodes (128 nodes). This trend is more stable for `phishing` and `a9a` datasets throughout all degrees and it is most noticeable between 30 to 70 regular graphs for `skinNonskin` and `ijcnn` datasets. Note for the degree (d) close to the number of graph nodes, the performance improvement saturates or becomes negligible. This is most visible for degree between 90 to 100 for all datasets.

As shown in Fig. 4, `ijcnn` did not converge for 10 and 20 regular graphs in less than 10000 iterations which we put as the maximum iteration for the convergence. This might

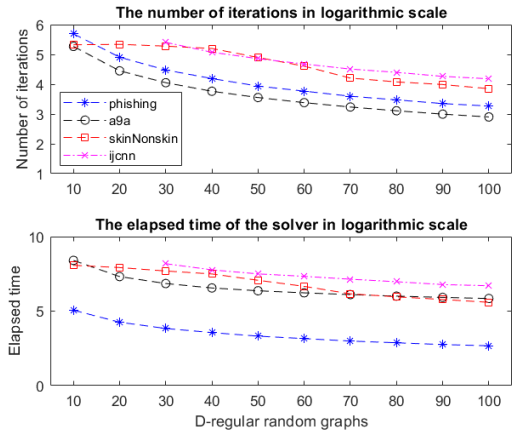


Fig. 4. Impact of d-regular graphs on the number of iterations and time for `phishing`, `a9a`, `skinNonskin`, and `ijcnn` datasets using 128 graph nodes.

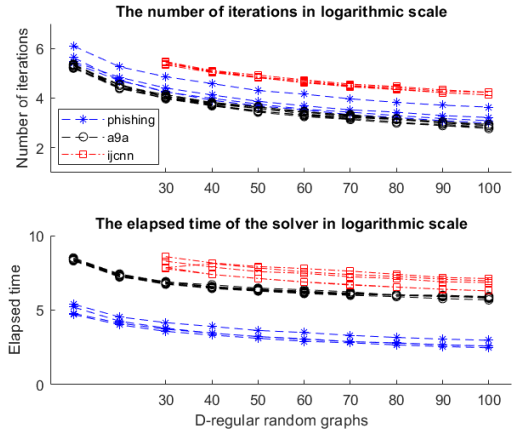


Fig. 5. Several rounds of shuffling data. Shuffling increases the classification accuracy while keeping the trend of decreasing the number of iterations and time for d-regular graphs.

be explained due to inefficient communication between graph nodes when the degree of the graph is low.

Fig. 5 shows the number of iterations for several rounds of shuffling data. Shuffling data follows the trend of decreasing the number of iterations and time, besides improving the classification accuracy and leading to a stable classifier.

VIII. Disussion

The performance of a network-based ADMM implementation of SVMs was improved based upon the efficient connection between neighbouring nodes using expander/Ramanujan graphs. The graphs with high degrees and spectral gaps and consequently many neighbouring nodes exhibit accelerated convergence (Fig. 3 and 4). This finding is consistent with that of Cao et al. [1] whose theoretical analysis suggested networks with higher mean degree. Although their approach was not tested for network-based ADMM algorithms, nor were tested for nonlinear systems, the authors reported the performance improvement of a distributed algorithm in terms of convergence.

The bounds of the spectral gap expanded as the degree of the graph and consequently connectivity increased for the fixed number of nodes, thus efficient communication between neighboring nodes. Note that a random d-regular graph may have better connectivity than a random $d^{'}$-regular graph even if $d < d^{'}$. This is likely if the spectral gap of d-regular graph is larger than $d^{'}$-regular graph, e.g., as shown in Fig. 1 by "o" symbols, a random 11-regular graph has a higher spectral gap than a random 13-regular graph. Therefore, increasing the degree of a graph will not necessarily lead to better connectivity.

The results of our experiments showed accelerated convergence for increasing the degree of expander graphs, however the decrease of the number of iterations was saturated or became very small as the degree of the graph got closer to the number of graph nodes (Fig. 3 and 4). This may be explained due to the increasing number of neighbors, although the convergence is guaranteed, many graph nodes should communicate and exchange their local results to reach the consensus. This leads to slow convergence.

Based on the line of reasoning above, it is interesting to find an appropriate degree of regular graphs with regards to the number of graph nodes in which the degree should be sufficiently high that leads to efficient communication and sufficiently low that leads to good connectivity.

The impact of many communicating neighbouring nodes is minor in the shared memory parallelism. In contrast, in the distributed memory parallelism, the communication between many nodes reaching the consensus might cause communication overhead. Thereby, increasing the degree of graphs close to complete graphs may not be efficient as in the shared memory parallelism. We will further investigate the impact of expander graphs on the performance of our algorithm using distributed parallelism in the future work.

Beside the discussion of graph topology, several factors showed their importance for the classification accuracy. One factor was the balance between the two classes. To keep the balance between two classes, we randomly shuffled data and it exhibited higher classification accuracy versus unshuffled data while keeping the trend of decreasing the number of iterations and time. This might not possible for networks that supply their data from distributed sources and do not allow to combine and shuffle the data due to privacy issues. Another factor was data cleansing and improving data quality. Although we did not investigate different techniques of data cleansing, simply removing missing and corrupted data exhibited higher accuracy. Our implemented algorithm is able to solve the dense and sparse formats of data and we plan to further improve the performance in regards to special treatment of sparsity.

IX. Summary and conclusion

A consensus-based ADMM implementation of SVMs involves a consensus agreement among neighboring nodes, consequently the number of neighbors and the way they are connected impact the performance of the algorithm in terms of convergence and communication. Thereby, the network topology used for communication of neighboring nodes plays an important role in performance improvement of the distributed algorithm.

Based on the line of reasoning above, complete graphs are well-known for their connectivity, however, high connectivity does not come cheap in particular using distributed memory parallelism, i.e., all nodes need to communicate with each other and this may increase the number of iterations until convergence and increase the communication complexity. Random d-regular expander graphs are good sparse approximations of complete graphs [20] in which good connectivity property is inherited with efficient communication between nodes. Note the better the expander property of regular graphs is, the faster nodes communicate.

Acknowledgment

This work is supported by universities of Borås and Gothenburg.

References

[1] H. T. Cao, T. E. Gibson, S. Mou, and Y. Y. Liu, "Impacts of network topology on the performance of a distributed algorithm solving linear equations," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 1733–1738.

[2] G. França and J. Bento, "How is Distributed ADMM Affected by Network Topology?" *ArXiv e-prints*, Oct. 2017.

[3] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Microsoft Research, Tech. Rep. MSR-TR-98-14, April 1998. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=69644

[4] L. Donetti, F. Neri, and M. A. Muoz, "Optimal network topologies: expanders, cages, ramanujan graphs, entangled networks and all that," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2006, no. 08, p. P08007, 2006. [Online]. Available: http://stacks.iop.org/1742-5468/2006/i=08/a=P08007

[5] Y. T. Chow, W. Shi, T. Wu, and W. Yin, "Expander graph and communication-efficient decentralized optimization," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov 2016, pp. 1715–1720.

[6] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[7] J. Platt *et al.*, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[8] H. Byun and S.-W. Lee, "Applications of support vector machines for pattern recognition: A survey," in *Pattern recognition with support vector machines*. Springer, 2002, pp. 213–236.

[9] A. Carpenter, "cusvm: A cuda implementation of support vector classification and regression," *patternsonscreen. net/cuSVMDesc. pdf*, 2009.

[10] O. Ivanciuc, "Applications of support vector machines in chemistry," *Reviews in computational chemistry*, vol. 23, p. 291, 2007.

[11] G. Zanghirati and L. Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel computing*, vol. 29, no. 4, pp. 535–551, 2003.

[12] B. He and X. Yuan, "On the $o(1/n)$ convergence rate of the douglasrachford alternating direction method," *SIAM Journal on Numerical Analysis*, vol. 50, no. 2, pp. 700–709, 2012. [Online]. Available: https://doi.org/10.1137/110836936

[13] R. Monteiro and B. Svaiter, "Iteration-complexity of block-decomposition algorithms and the alternating direction method of multipliers," *SIAM Journal on Optimization*, vol. 23, no. 1, pp. 475–507, 2013. [Online]. Available: https://doi.org/10.1137/110849468

[14] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *The Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[16] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[17] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[18] J. Friedman, "A proof of alon's second eigenvalue conjecture and related problems," *CoRR*, vol. cs.DM/0405020, 2004. [Online]. Available: http://arxiv.org/abs/cs.DM/0405020

[19] M. Morgenstern, "Existence and explicit constructions of q + 1 regular ramanujan graphs for every prime power q," *Journal of Combinatorial Theory, Series B*, vol. 62, no. 1, pp. 44 – 62, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0095895684710549

[20] C. Williamson, "Spectral graph theory, expanders, and ramanujan graphs," *University of Washington*, 2014.

PUBLICATIONS IN THE DISSERTATION SERIES

Berg Marklund, Björn (2013). *Games in Formal Educational Settings: Obstacles for the development and use of learning games*. Dissertation Series, No. 1. University of Skövde, School of Humanities and Informatics, The Informatics Research Centre. ISBN: 978-91-981474-0-7. URN: urn:nbn:se:his:diva-8627.

Aslam, Tehseen (2013). *Analysis of manufacturing supply chains using system dynamics and multi-objective optimization*. Dissertation Series, No. 2. University of Skövde, School of Technology and Society, The Virtual Systems Research Centre. ISBN: 978-91981474-1-4. URN: urn:nbn:se:his:diva-8661.

Laxhammar, Rikard (2014). *Conformal anomaly detection: Detecting abnormal trajectories in surveillance applications*. Dissertation Series, No. 3. University of Skövde, School of Informatics. ISBN: 978-91-981474-2-1. URN: urn : nbn : se : his : diva-8762.

Alklind Taylor, Anna-Sofia (2014). *Facilitation matters: A framework for instructor-led serious gaming*. Dissertation Series, No. 4. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-981474-4-5. URN: urn:nbn:se:his:diva-9923.

Holgersson, Jesper (2014). *User Participation In Public e-service Development: Guidelines for including external users*. Dissertation Series, No. 5. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-981474-5-2. URN: urn:nbn:se:his:diva-10169.

Kaidalova, Julia (2015). *Towards a definition of the role of enterprise modeling in the context of business and IT alignment*. Licentiate Thesis. Dissertation Series, No. 6. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-981474-6-9. URN: urn:nbn:se:his:diva-11695.

Rexhepi, Hanife (2015). *Improving healthcare information systems: A key to evidence based medicine*. Licentiate Thesis. Dissertation Series, No. 7. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-981474-7-6. URN: urn:nbn:se:his:diva-11019.

Berg Marklund, Björn (2015). *Unpacking Digital Game-Based Learning: The complexities of developing and using educational games*. Dissertation Series, No. 8. University of Skövde, The Informatics Research Centre, School of Informatics. ISBN: 978-91-981474-8-3. URN: urn:nbn:se:his:diva-11805.

Fornlöf, Veronica (2016). *Improved remaining useful life estimations for on-condition parts in aircraft engines*. Licentiate Thesis. Dissertation Series, No. 9. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-981474-9-0. URN: urn:nbn:se:his:diva-12653.

Ohlander, Ulrika (2016). *Towards Enhanced Tactical Support Systems*. Licentiate Thesis. Dissertation Series, No. 10. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-982690-0-0. URN: urn:nbn:se:his:diva-13133.

Siegmund, Florian (2016). *Dynamic Resampling for Preference-based Evolutionary Multi-objective Optimization of Stochastic Systems: Improving the efficiency of time-constrained optimization*. Dissertation Series, No. 11. University of Skövde, School of Informatics. ISBN: 978-91-982690-1-7. URN: urn:nbn:se:his:diva-13088.

Kolbeinsson, Ari (2016). *Managing Interruptions in Manufacturing: Towards a Theoretical Framework for Interruptions in Manufacturing Assembly*. Licentiate The-

sis. Dissertation Series, No. 12. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-982690-2-4. URN: urn:nbn:se:his:diva-12791.

Sigholm, Johan (2016). *Secure Tactical Communications for Inter-Organizational Collaboration: The Role of Emerging Information and Communications Technology, Privacy Issues, and Cyber Threats on the Digital Battlefield*. Licentiate Thesis. Dissertation Series, No. 13. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-982690-3-1. URN: urn:nbn:se:his:diva-13468.

Brodin, Martin (2016). *Mobile Device Strategy: A management framework for securing company information assets on mobile devices*. Licentiate Thesis. Dissertation Series, No. 15. University of Skövde, School of Informatics, The Informatics Research Centre. ISBN: 978-91-982690-5-5. URN: urn:nbn:se:his:diva-13125.

Ericson, Stefan (2017). *Vision-Based Perception for Localization of Autonomous Agricultural Robots*. Dissertation Series, No. 16. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-982690-7-9. URN: urn:nbn:se:his:diva-13408.

Holm, Magnus (2017). *Adaptive Decision Support for Shop-floor Operators using Function Blocks*. Dissertation Series, No. 17. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-982690-8-6. URN: urn:nbn:se:his:diva-13503.

Rexhepi, Hanife (2018). *BRIDGING THE INFORMATION GAP: Supporting Evidence-Based Medicine and Shared Decision-Making through Information Systems*. Dissertation Series, No. 19. University of Skövde, School of Informatics. ISBN: 978-91-984187-1-2. URN: urn:nbn:se:his:diva-15210.

Schmidt, Bernard (2018). *Toward Predictive Maintenance in a Cloud Manufacturing Environment: A population-wide approach*. Dissertation Series, No. 20. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-984187-2-9. URN: urn:nbn:se:his:diva-15120.

Linnéusson, Gary (2018). *Towards strategic development of maintenance and its effects on production performance: A hybrid simulation-based optimization framework*. Dissertation Series, No. 21. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-984187-3-6. URN: urn:nbn:se:his:diva-15036.

Amouzgar, Kaveh (2018). *Metamodel Based Multi-Objective Optimization with Finite-Element Applications*. Dissertation Series, No. 22. University of Skövde, School of Engineering Science. ISBN: 978-91-984187-4-3. URN: urn:nbn:se:his:diva-15145.

Bernedixen, Jacob (2018). *Automated Bottleneck Analysis of Production Systems: Increasing the applicability of simulation-based multi-objective optimization for bottleneck analysis within industry*. Dissertation Series, No. 23. University of Skövde, School of Engineering Science, The Virtual Systems Research Centre. ISBN: 978-91-984187-6-7. URN: urn:nbn:se:his:diva-15214.

Karlsson, Ingemar (2018). "An Interactive Decision Support System Using Simulation-Based Optimization and Knowledge Extraction." PhD thesis. University of Skövde, School of Informatics. ISBN: 978-91-984187-5-0.

Andersson, Martin (2018). “A Bilevel Approach To Parameter Tuning of Optimization Algorithms Using Evolutionary Computing.” PhD thesis. University of Skövde, School of Informatics. ISBN: 978-91-984187-7-4.

Tavara, Shirin (2018). “High-Performance Computing For Support Vector Machines.” PhD thesis. University of Skövde, School of Informatics. ISBN: 978-91-984187-8-1.

Bevilacqua, Fernando (2018). “Game-calibrated and user-tailored remote detection of emotion: A non-intrusive, multifactorial camera-based approach for detecting stress and boredom of players in games.” PhD thesis. University of Skövde, School of Informatics. ISBN: 978-91-984187-9-8.