Datalogisk Institut
KØBENHAVNS UNIVERSITET

Using Support Vector Machines for Distributed Machine Learning

Rasmus Ulslev Pedersen, B.Sc., MBA

Technical Report no. 05/01
ISSN: 0107-8283

DIKU
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Using Support Vector Machines for Distributed Machine Learning

by

Rasmus Ulslev Pedersen, B.Sc., MBA

Dissertation

Presented at Department of Computer Science

University of Copenhagen

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

University of Copenhagen

August 2004

Acknowledgments

I would especially like to thank my thesis supervisor Dr. Eric Jul for kind guidance and support in academic as well as non-academic issues. Dr. Joydeep Ghosh, who provided the setting for me at University of Texas in Austin, is someone to whom I will also be forever grateful. Thank you also for the hospitality while Dr. Eric Jul visited UT in the summer of 2002. Now, I would like to thank Stig Jørgensen and Aage Knudsen for kind support while fulfilling their responsibilities as part of the thesis control group. The same extension of gratitude goes to Willy Bergstrøm and Dr. Preben Alstrøm. I would also like to acknowledge the members of DistLab and especially Martin Leopold. Moreover, without the grant support of the Danish Academy of Technical Sciences, this would have been possible. Finally, but not least, I owe thanks my lovely wife, son, and the rest of my family for endorsing this endeavor.

RASMUS ULSLEV PEDERSEN

University of Copenhagen

August 2004

Contents

List of Tables

List of Figures

Abstract

In this thesis we investigate the potential use of support vector machines (SVMs) for distributed machine learning. An SVM is an algorithm out of the machine learning field, which can be used for classification, regression, and other important tasks. The novel approach in this thesis is to apply the SVMs as co-active learning units while respecting the distributed setting of the problem. We construct the problem in such a way that significant constraints in the distributed learning system are introduced to add further dimensions to the research problems. In this framework, which we for simplicity label Distributed Support Vector Machine, it is not enough only to look at the objective function of the SVM. In many instances various constraints of the distributed units would need to be considered in the training and deployment aspects of the distributed learners. The ideas of using SVMs as the base learner for co-active, distributed learning is not previously tested. Therefore, this thesis is about presenting the basic framework and justify further research in the field.

Chapter 1

Introduction

This research is interdisciplinary by nature. Accordingly, we use components from three disciplines to attain our research objectives of describing an intelligent, energy-aware distributed learning system.

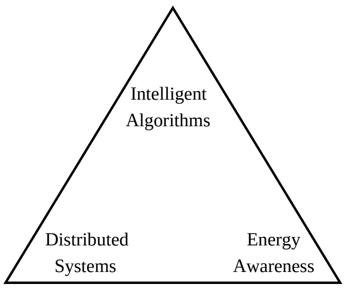

Figure 1.1: Interdisciplinary nature of thesis. The study is rooted in three interrelated areas of computer science, namely intelligent algorithms/machine learning, distributed systems, and energy awareness.

Machine learning in distributed systems differs from non-distributed machine learning as it is subject to a number of additional constraints related to communication, computational, spatial, and temporal issues. A general characteristic of machine learning is that it covers fundamental learning tasks such as classification, regression, and clustering. The application scope of distributed machine learning is wide, and the selection of an appropriate algorithm is often ad hoc. In a quest to develop a distributed machine learning framework capable of solving many learning tasks yet configurable toward additional distribution-related constraints, a certain algorithmic family seems particularly interesting: support vector machines (SVMs). Its main asset in this context—as a dual formulated maximal margin algorithm—is the well-known characteristic that the full model is inherently specified by only a smaller subset of the available data points. This basic quality and other useful modifications are combined with the sequential minimal optimization algorithm to specify a framework capable of solving fundamental machine learning tasks and also implement and document a working prototype in Java. The framework is labelled Distributed Support Vector Machine (DSVM). Our description, research, and results are positive in terms of the applicability of the SVM for this purpose. The future of this framework is likely but maybe not immediate.

A brief and high-level introduction to SVMs would be to present it as a mathematical function that works on data generated by some system. Some examples of data could be sensor data such as acceleration, light, and temperature, or data related to a group of people in some pervasive domain such as mobile telephony. The overall goal of using SVMs on such data is to gain insights related to the nature of the data such that we are able to make or refrain from certain actions. Following the sensor data example, the sensor nodes can be programmed to take specific actions if they detect a certain combination of light, sound, and temperature. Data input quickly become too complicated for ad-hoc programming in order to infer meaningful patterns in the data, and therefore, one can make use of an appropriate family of algorithms such as SVMs to address this issue. The second example of using SVMs in a mobile and pervasive setting could be motivated by the idea that we want to offer relevant services and information to the users. It would be the same goals as hold for the large online bookstore www.amazon.com: to find out what the customer preferences are based on previous individual and collective actions. SVMs are one set of algorithms, which has received significant interest over that last decade. They have been applied to various problems with great success, and this thesis is subject to a new line of research: the potential use of SVMs in distributed systems using properties of the SVMs to further make them useful for distributed machine learning.

1.1 Research Question

This thesis describes a framework for using the SVM in a distributed environment with an emphasis on constrained computing. Our definition of a distributed environment is characterized by multiple input data collection nodes for each inference problem. In this supposedly machine-learnable environment, we attempt to combine one family of algorithms (support vector machines) with one context (distributed learning) to establish a framework for distributed inference founded in statistical learning theory. More specifically, we research if the SVM is a suitable learning machine for working in a distributed environment due to its inherent advantages in terms of scalability and robustness of a possible solution. With respect to the SVM, we develop approaches to exchange information, which is applicable to SVM-learnable problems such as classification, regression, ordinal regression, clustering, and single class novelty detection. This approach is used to create a scalable solution framework to address some inference problems in a distributed environment.

The two main research questions are:

- Is the standard support vector machine useful for distributed machine learning?
- Can the support vector machine be applied specifically for distributed machine learning?

It has already been established that SVMs are useful in many machine learning problems such as optical character recognition. Therefore, that is not raised as a research question in this thesis. We define distributed machine learning as a problem domain characterized by computational and other system constraints. Accordingly, the research design centers on a review of the SVM literature with an emphasis on those parts that can be used directly or indirectly to control the resource and energy consumption of SVMs. The work is empirical in nature with a focus on low dimensional data examples to allow visualization of the key points. A goal of the thesis is also to provide implementations of the SVM on multiple constrained platforms.

The overall problem of optimizing a distributed learning system over its entire lifecycle can be expressed as a related set of functions:

$$W(R, T, S) \tag{1.1}$$

$$R(L) \tag{1.2}$$

$$T(D, S) \tag{1.3}$$

$$S(CC, HW, SW) \tag{1.4}$$

The overall optimization problem in consideration is W, which is the overall performance of a distributed inference system over its entire lifespan. It is a function that depends on three broad variables. The quality of the learning machines in the distributed inference system can be summarized by R, which is a function of a given loss function L. Each type of learning problem can be associated with a particular loss function, L. Next, W depends on T, which is a time-related variable. Some systems will have long T and other systems, such as sensor networks, currently have a shorter span of life. The last variable of importance to W is S. It captures the distributed system properties and associated costs. These costs are related to communization cost and security costs. With W being introduced, we can look at the next three equations. The risk functional R, is dependent on the loss function L of the underlying learning machine(s). The time related costs T, depends on D, the data available to the system, and S, the system properties of the system. S is defined to be related and determined by CC, communication costs and constraints in the system, the hardware used, HW, and the software used, SW. This broad range of properties for a distributed inference system is useful for explaining the scope and motivation of the thesis.

The motivation of the thesis is mostly hinged on CC, the communication cost, and HW, the hardware related constraints. However, significant effort has also gone into finding methods, within the SVM class of learning machines, that can facilitate a small software implementation/maintenance cost, SW, which is the primary reason for choosing a well-established language (Java) as the main prototyping language. Standard security issues, which would be related to S have not been an area of immediate focus.

We investigate how SVMs can be used as the means for constructing a distributed inference model. It is possible to construct both a distributed clustering model as well as a distributed classification model. As this is a new application of SVMs and because distributed clustering is a large topic, we mainly discuss distributed classification in this thesis. The use of support vectors for clustering is discussed in a paper by A. Ben-Hur et al. [6]. The choice of starting with a distributed classifier is natural as the concepts and ideas in this thesis can be analyzed in this setting without the need to work in parallel on regression and clustering applications.

The organization of the thesis serves to support our research objectives with respect to a distributed support vector machine (DSVM). In chapter 2 the background and related work for the support vector machine (SVM), distributed data mining (DDM), and distributed systems are discussed. The discussion on SVMs provides the references to the papers that pioneered this field. These sources provide the foundation for the thesis. There are several a priori reasons why the DSVM framework is interesting from both an application and theoretical point of view.

1. Vectors (the data) given in the system are either used by the SVM or not, which leads to a natural reduction in data storage by the distributed nodes.

2. The use of kernels—an important mathematical object for SVMs—on distributed nodes allows for inference in high-dimensional vector spaces while raw data can still be exchanged in its potentially lower-dimensional input space.

3. Each node in the distributed system can employ a specific kernel to its local data set.

4. The distributed support vector machine classification framework can be extended to other learning tasks such as distributed regression or distributed clustering.

5. The existence of error bounds rooted in statistical learning theory [93] allows for an analytical approach to models for trading error margin, or variance of same, for communication load, CPU power, or speed of training new models.

It is for these reasons that we foresee the SVM model as a viable learning algorithm for distributed machine learning environments. The literature review covers existing research within the three main areas relating to this thesis: Machine learning/support vector machines, distributed systems, and distributed data mining. The review especially focuses on the segment of the SVM research that explores methods for reducing the computational burdens of working with SVMs, such that it can be applied to the DSVM framework to address problems in constrained environments. The SVM can be used for regression and clustering problems. Therefore, a brief review of those two important tasks are included in the review. Following the review of SVMs, the already established work in distributed data mining is presented. A large body of literature related to software agents and intelligent versions of the same is available. Since we do not use the agent perspective to a great extent in the DSVM framework, literature addressing this issue is included mainly for completeness. The constraints, which give meaning to the DSVM framework, are mainly taken from the science of distributed systems. We include references to recent research such as wireless sensor networks [99] to point to an area of distributed systems that is particularly dominated by different forms of computational constraints. It is mainly CPU, storage, and communication constraints that are of interest when reviewing the literature.

The main part of the thesis consists of two chapters that represent the principal discussion on the use of SVMs for distributed learning tasks in constrained environments. The first of those chapters addresses the issues related to using the SVM on a single node, while the following chapter covers the multi-node aspects. The chapter detailing the single-node SVM employs the central algorithmic aspects. It presents methods for reducing the storage requirements of the SVM while training. This is based on work originally done by John Platt, Microsoft Research, followed by further optimizations by Kerthii et. al [60]. There are several methods available to enhance the SVM's usefulness on the distributed nodes, and topics related to both training and testing are incorporated into the research. After addressing the single-node SVM, we introduce multi-node SVMs.

In chapter 4 we discuss the segment of the DSVM that relates to exchange of data points on multi-node SVMs. This is an indirect discussion of the important issues related to radio communication in distributed (wireless) systems. Experiments are conducted on several combinations of exchange schemes. We introduce a method motivated by information retrieval to evaluate the exchange effectiveness in the multi-node setting. A body of proxies that connect the SVM parameters to computational constraints is also presented. We later use a portion of these proxies in chapter 6 for our energy awareness experiments. The main implementation has been done in Java, and working prototypes on several platforms are available. This is the topic of the next chapter.

A systems analysis is presented in chapter 5. We divide the responsibilities within the DSVM framework into a number of components. The analysis is on an abstract level, and the implementation can be found at the online location www.dsvm.org [32]. We conclude the main portion of this work with a series of energy-related experiments that shows the relative energy consumption for small Java-based systems and a TinyOS sensor node system.

A goal of a distributed learning system could be energy efficiency. With that in mind, we have conducted a series of energy-related experiments based on the models of chapter 3 with respect to the single-node SVM and some experiments on the multi-node SVM. The chapter on energy awareness is therefore a series of energy-related experiments, which can be divided into three categories. Syntectic experiments, which are aimed at testing a small portion of the proxies presented in chapter 3 and 4, is the first category. The second category is direct energy consumption using the single-node DSVM on different software and hardware platforms. Although not directly comparable, we present the first results of running an SVM classifier on a sensor network node. Finally, we peek into the multi-node SVM by measuring the energy usage of single-node classification vs. multi-node radio transmission of the data.

Finally, we discuss and summarize the results in chapter 7. A short plan for future research projects is presented. As this work is interdisciplinary, we introduce the main terms and abbreviations up front, see Table 1.1. Other terms used in the thesis are defined at the point of usage.

Table 1.1: Definition of Key Terms

Term	Definition
SVM	Support vector machine that classifies novel points on the nodes and only transmits key information such as support vectors across the network.
SMO	Sequential Minimal Optimization, which is a method for training an SVM efficiently [73].
SV	Set of support vectors belonging to an SVM model.
SV_i	A support vector, which is one of the points that make up the SVM model.
$\boldsymbol{\alpha}$	A Lagrange multiplier, which is a key parameter denoted $\boldsymbol{\alpha}$ in the SVM model. Each point with $\alpha_i > 0$ is an SV and its importance can be interpreted as the size of α_i
KKT	The Karush-Kuhn-Tucker conditions, which is a set of constraints that the SVM model must obey within a given error tolerance when trained to optimality.
tol	Tolerance parameter for the SVM that is used as a stopping criterion.
$k(\boldsymbol{x}_i, \boldsymbol{x}_j)$	Kernel evaluation that is a central computation in the SVM model. It is a dot product similarity measure between two data points in a space (usually) of higher dimensionality than the input space associated with the raw data observations.
$\#k$	A count of the number of kernel evaluations the SVM uses while computing the optimal $\boldsymbol{\alpha}$ for the SVM. $\#k$ is used as a proxy for computational time complexity in the machine learning community.
Compression	A term that we use to measure how many or how much is saved in network traffic by running remote SVMs. A compression of 90% is to be understood such that as just 10% of the raw data is exchanged in the distributed system. Compression is also to be read as *filtering*.
ms	Milliseconds, which is the unit used to measure how long it takes to classify a novel point. [30]
mW	Milliwatt, which is the power consumption unit calculated for each of the node systems. [30]
μ J	Microjoule, which is the unit used to measure the energy consumption per point. [30]
p	Number of points sensed at a sensor node, and a key variable in consuming energy of the SVM on the individual node.

Chapter 2

Background and Related Work

This thesis is interdisciplinary by nature, and therefore we identify and position the work in relation to well-established research areas such at those identified in Figure 1.1 and Figure 2.1. Machine learning is the provider of the intelligence part of the distributed support vector machine (DSVM). The chosen algorithm to be studied is SVM. A brief survey of different SVMs is conducted such that it will be possible to combine and apply the appropriate SVM to address a specific problem area of distributed learning. First, we will look at the chosen machine learning algorithm: SVM. The history of the SVM is briefly covered along with the state of the current research in this area. While the SVM is the primary target of the later analysis, the distributed systems research will provide much of the motivation for using SVMs in the distributed context of this thesis. Problems related to resource constraints provide the guidance of the DSVM design. Therefore, the focus on distribution of nodes, mobility of nodes, and resource constraints of nodes is the second area of study and a foundation for this thesis due to its central position in the distributed systems area. Sensor networks, which are usually characterized by a number of energy constraints, is an research area of significant focus [48]. We look at these small devices taking into account the constraints they bear. An emerging branch of sensor networks is wireless ad-hoc networks, which is the establishment of connected groups of sensor radios for some duration of time. Sensor network research and some open problems within this research area are included in the background study. A natural way to combine machine learning and distributed systems is provided within the context of distributed data mining. The survey of distributed data mining centers somewhat on classification as there is a large body of existing work that can be used as a starting point in the analysis. A summary of the background and related work is provided in Figure 2.1.

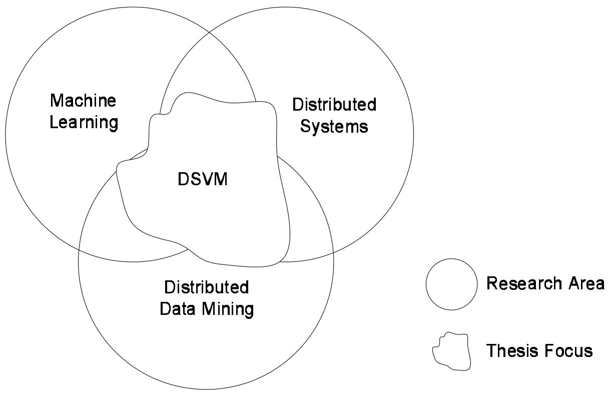

Figure 2.1: The overlap between machine learning, distributed systems, and distributed data mining.

2.1 Machine Learning

Statistical learning theory (STL) has had a profound impact on learning theory over the last two decades, which could be supported by the over 700 references at the SVM related site www.kernel-machines.org. STL has been developed and synthesized primarily by Vapnik [93]. He led the work on the support vector algorithm upon which this theory is based.

In much of the work on distributed data mining (DDM), the primary goal is identical to that of single-source data mining, namely the ability of the learner to generalize across unseen data examples. To achieve this behavior of the machine-learner, we need to present to it a series of data points the represent well the underlying phenomenon that we seek to make a model for. As classification learners are typically evaluated on their ability to generalize well, so are SVMs when used in classification problem settings. A classification problem covers a broad range of application. One application could be to determined if an image is a tree or not. Another application could be to determine a state model for a distributed sensor network. The generalization capability comes from the fact that SVMs are large margin classifiers. This means the training algorithm seeks to separate the positive points from the negative points, if the learning process is a binary classification problem. The problem of identifying an vectorial image as either a tree or not is an example of a binary classification problem. This is achieved by trying to separate the two classes of points with a plane that is equidistant from the closest points of each class. If there was two measurements on a line and each one belonged to different classes, then the SVM would place the discriminant function mid-way between the two points. The solution can also be restated as the plane that separates the convex hulls of each class with the largest possible margin. A convex hull is set of points on the boundary of some class of points. Some classification problems are either so noisy or impossible to separate completely due to the lack of information in the chosen dimensions describing properties of the class instances. Noise is present when poor measurements is part of the problem domain or there are lacking key attributed to achieve separation of the data points. The solution to this problem was proposed by C. Cortes and V. Vapnik in [27]. In their paper, the soft margin SVM is introduced, which allows the SVM to be trained using slack margins. Slack margins is a method incorporated in this algorithm to balance the misclassification rate against generalization ability. Misclassification is related to those data instances that can not be classified correctly during the training process of the model. This is balanced by introducing an upper limit to the size that the individual Lagrange multipliers can take. The Lagrange multipliers is part of a linear combination of training points in feature space. A common choice is to set this upper limit to 10, but it will be problem dependent. Given that the kernel has been chosen, the choice of C, which is the name for this parameter, is optimized to the training set by use of a validation set. The value of C is varied over a range of numbers while the classification accuracy is monitored against a validation set. There are variations of this scheme using the parameter C to control the soft margins. A complementary SVM is the

ν-SVM that depends on a parameter, ν, to control the margins. The parameter ν is interesting in that it provides an upper bound on the number of support vectors in the solution.

Classification ability is one way of using the system. In particular, distributed classification can take place using a consolidated global model. A global data model is conceptually a data matrix of all the data points in rows and the associated data dimensions in the columns. This model can be constructed using two principles. One is for which the local classifiers remain local and a consolidated classifier is constructed using some query and meta voting scheme [49] [50]. The majority vote is perhaps the most-used method. In this case each of the local classifiers would be queried on the classification of a new point. The class with the majority of votes would be the label the meta-voter would assign to the class. An alternative method is to merge the models by extracting information from the local classifiers. If the local classifiers were multi-layer perceptron (MLP) neural networks (NN) (see Bishop's book *Neural Networks for Pattern Recognition* [12] for a complete introduction) this would not be a straightforward task; it would be impossible in many instances. For an algorithm such as the SVM this would be easier as it is characterized by the dependency on a subset of the training points and, furthermore, it exhibits the feature that the optimal solution is unique. More specifically, the Lagrange multipliers (α) are the same even if the model is trained again with the same data. This is different from a MLP NN as it would be unlikely to arrive at the exact same solution as the weight parameters of the network are randomized prior to training. It is common to reduce multiclass problems to binary classification problems [2]. The multiclass classification problem can be solved, if the N binary classifiers are real valued functions by assigning the point to the class with the largest functional output [76]. In some cases, it can be advantageous to resort to query-based learning. A strategy used for SVMs by Campbell et al. [20] is to request the labels of the data that are closest to the separating hyperplanes of a partially trained SVM.. In the same category of algorithms, which bears the potential to address some computational constraints, is the incremental SVM [21]. It retains all previous Karush-Kuhn-Tucker (KKT) information while new points are added to the solution one at a time.

2.2 Distributed Data Mining

The combination of distributed systems and the pervasive/ubiquitous computing paradigms can lead into the area of distributed data mining. It is a broad area characterized by combinations of multiple learning algorithms as well as multiple learning problems. These learning problems are also directed by constraints such as security and energy restrictions. One distinction in distributed data mining (DDM) is if the data is partitioned across rows or columns. The column separation scheme has been researched by Kargupta et al. [58].

It is proposed [66] [94] to split distributed artificial intelligence (DAI) up into two areas: multi agent systems (MAS) and distributed problem solving (DPS). MAS is more concerned with broader agent skills and reasoning capabilities while the DPS approach is more problem and domain specific. We place distributed inference systems (DIS) and the even more specialized distributed classification efforts in the same category as DPS. The main reason is because the inference problem is well known: i.e., there is full control of which data, how much data, and where data might appear in the system.

In some stand-alone data mining problems, it is necessary to partition the data set into smaller subsets to make a fit into the available main memory. This leads toward a discussion on parallelization of the learning task as well as an early pointer toward resource-constrained data mining (DM) problems. It has been proposed to split the data into subsets that are subsequently used for training a model. Later, the results of the local base classifiers are combined by some voting scheme such a meta voting scheme [22]. It was proposed to use a meta learning algorithm for combining the predictions of the base classifiers. An opposite path is represented in the *WoRLD* [3], Worldwide Relational Learning Daemon, which is an inductive distributed database learning technique. It works by propagating markers indicating an item in a database as either relevant or not. Propagating these markers and looking for accumulations form the basic functionality of *WoRLD*. Another system for meta-learning in distributed environments is the *JAM* system, Java Agents for

Meta-learning [87] as described below.

The research on agents is extensive. There are basically two types of agent architectures, one in which the agents are tied to one node and one in which the agents move between nodes. In systems such as Objectspace's *Voyager* system, the agents move between the machines, while a peer-to-peer system such as *JXTA* only allows agents to be tied to the nodes. In both systems the agents either reside or migrate to nodes that are prepared for receiving. Distributed objects can be seen as a premise for agents: only some additional characteristics need to be adorned to the distributed object. It has been noted that agents normally differ from machine learning algorithms in that the nature of the problems addressed by the agents are larger than those traditionally associated with machine learning [28]. Intelligent agents are more involved software than just an agent, which can be compared to a distributed object. The term distributed object perhaps belongs to systems in which the focus is more on the distributed system than on the interaction between the agents. One such example is the *Java Agents for Meta-Learning* (JAM) system developed by Stolfo et al. [87]. The agent can communicate with other agents using the *Knowledge Query and Manipulation Language* (KQML) [35] together with the Knowledge Interchange Formalism (KVI) [41] agent information exchange format. These communications and associated exchange formats allow the agent to express first order requests. A complementary system for agent corporation is the *Open Agent Architecture* (OOA) [64]. In some cases the agents will query other agents and perhaps exchange examples. It is optimal not to exchange all the training examples but rather resort to using the system in query mode. Grecu and Becker [44] discuss coactive learning [47] in distributed environments. This leads to a framework in which the individual agent can decide to update its decision model based on the information it receives. The use of mobile filter agents was found [90] to reduce the load on communication lines by up to 90%. The work done by Theilmann and Rothermel was investigated in the context of distributed clustering. An advanced overview of clustering is found in a book chapter by J.Ghosh [34].

This section covers the area of distributed knowledge systems. *Knowledge* is understanding about desires and degrees of belief. A distinction exists about whether or not the agent has beliefs about beliefs. If this is not the case, then the agent is said to have first order beliefs. Knowledge is the gathering of information for the purpose of presenting it in distilled formats to the user. A knowledge query language [35] and a knowledge information interchange format is used to gather knowledge. This lets an agent express queries toward other agents to fulfill its goals. The knowledge interchange format lets an agent pack this knowledge in a format understood by other agents. The degrees of belief that an agent uses can be generated in many ways.

Information Retrieval (IR) has its roots in the library sciences in which document retrieval is a key process. Despite its roots, IR has moved to occupy a significant role in terms of making the Internet document base manageable for users. The Web search engine Google makes use of a system to mark the importance of individual pages in terms of the page-rank algorithm [68]. It models the Web as a Markov process, which follows hyperlinks on Web pages to assign a rank to Web pages on the Internet. There is a mechanism to ensure that deadlocks and a certain random behavior is effectuated. To access the quality of the queries submitted by the user, some measures of quality of the retrieved document collection are introduced. First, precision is measured, which is the number of relevant documents retrieved versus the total number of retrieved documents. The second measure is recall, which is the number of relevant documents retrieved versus the total number of relevant document in the entire collection. A measure that seeks to combine these two approaches is the harmonic mean. Joachims [51], among others, introduced SVMs for document classification. Work on transductive SVMs was also done by Bennet and Demiriz [7]. In the transductive setting, the learning function is constructed specifically for the unlabelled points and not for all points, as is usually the case.

In a review from 1982 Gayle W. Hill [47] analyzes the aspects of group decision making vs. individual decision making from a psychological viewpoint. It is an interesting parallel to the tasks that are common in machine learning. Hill creates four groupings of performance indicators. First, the group decision making performance is compared to the individual decision performance.

Second, the group performance is compared to the most competent member of the group. Third, the group performance is compared to the statistically pooled responses of the individuals. Fourth, he compares the group performance against mathematical models. Looking at these individual comparison modes in detail is an interesting aspect of the thesis. There are a number of comparisons possible, with similar ideas in machine learning. With some model choices, such as an MLP neural network, the individual responses are usually not particularly stable, and in some instances it can be advantageous to overtrain the networks before the individual classifications are polled by a meta learner [92].

The ubiquitous paradigm is generally attributed to Mark Weiser, as he introduced the subject in 1991 [95]. It takes a more human oriented approach to what can be regarded as the presence of computers everywhere. A main vision in ubiquitous computing is that computers should improve the lives of human beings. Weiser regards this computing paradigm as the opposite of virtual reality: In a virtual reality world, computers dominate whereas humans dominate in the real world while being transparently supported by computers for everyday tasks. The ubiquitous paradigm touts a conversion from many people sharing one computer such as a mainframe toward a scenario in which one computer shares many people. This setting can be context-aware, and it is a subject of current and future research [1]. As demonstrated by Bill Bodin, IBM Senior Technical Staff Member, at a presentation at University of Texas, Austin, in 2002, these ubiquitous ideas can carry over to the ideas of *pervasive* computing [98]. Bodin demonstrated a living area largely connected by computers put in most places imaginable. His demonstration and implementation did not include distributed machine learning to add a layer of intelligence to the system. However, there were rules encoded into the system such as that paintings should change to certain people's preferences as they entered the house. Bodin predicted that there would be a need for adding machine learning capabilities to these kinds of intelligent solutions in the future [9].

2.3 Support Vector Machines

The SVM is an algorithm that is based mainly on work performed by Vladimir N. Vapnik and coworkers. It was presented in 1992 and has been the subject of much research since. We look at the algorithm from an application viewpoint and review its characteristics. A secondary purpose of this review is to introduce the definitions that play a central part in the later chapters that define the DSVM framework.

The SVM algorithm is a maximal margin algorithm. It seeks to place a hyperplane between classes of points such that the distance between the closest points are maximized. It is equivalent to maximum separation of the distance between the convex hulls enclosing the class member points. Vladimr Vapnik is respected as the researcher who primarily laid the groundwork for the support vector algorithm. The first breakthrough came in 1992 when Boser et al. [15] constructed the SVM learning algorithm as we know it today. The algorithm worked for problems in which the two classes of points were separable by a hyperplane. In the meantime Corinna Cortes was completing her dissertation at AT&T labs, and she and Victor Vapnik worked out the soft margin approach [27]. It involves the introduction of slack variables, or error margins that are introduced to absorb errors that are inevitable for non-separable problems. The SVM was primarily constructed to address binary classification problems. This has been adapted by introducing versions of the SVM that can train a multiclassifier concurrently. Other approaches involved the use of voting schemes in which a meta-learner takes the votes from the individual binary classifiers and casts the final vote. A particularly easy voting scheme is the one-against-all voter [76], which for SVMs amounts to training C classifiers and finding the C_i classifier with the hyperplane furthest away from the new point to be tested. The SVM has been extended to other learning areas as well. The areas relevant for this work are the extension toward regression and clustering. The regression algorithm extension has been refined by Alex Smola and Bernhard Schölkopf and pioneered by Vapnik [93]. The regression case is carried out by using the slack variable approach once again. A so-called ε-tube is constructed around the regression line. The width ε constitutes the error free zone, and the points that fall within this zone are regarded as error free. If a point falls outside the tube, then a slack vector approach is introduced, which for the L_2 norm case amounts to minimizing the square distance to the regression line. It can be noted that the regression line is identical to the OLS regression should the width of the tube be set to zero.

Our aim is to provide a framework for addressing inherently distributed inference tasks. This includes, but is not limited to classification, regression, and clustering. We focus on distributed problems for which the node set, N, and the inference node set, IN, can be loaded with an appropriate algorithm from the SVM family of models. There are three types of SVMs that we investigate: classification, regression and clustering SVMs. A short study on using a single-class SVM [81] in a multi-node setting is also done. Each one is presented below together with a brief introduction to its set of parameters. The binary classification SVM provides a decision function:

$$f(\boldsymbol{x}, \boldsymbol{\alpha}, b) = \{\pm 1\} = sgn\left(\sum_{i=1}^{l} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right) \tag{2.1}$$

for which $\boldsymbol{\alpha}$ has been found by solving this optimization problem:

$$maximize\, W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{2.2}$$

subject to

$$0 \le \alpha_i \le C, \text{ and} \tag{2.3}$$

$$\sum_{i=1}^{l} y_i \alpha_i = 0. \tag{2.4}$$

The functional output of (2.1) is ± 1, which works as a classification or categorization of the unknown datum $\boldsymbol{x}$ into either the $+$ or $-$ class. An SVM model is constructed by summing a linear combination of training data (historical data) in feature space. Feature space is implicitly constructed by the use of kernels, k. A kernel is a dot product, also called inner product, in a space that is usually not of the same dimensionality as the original input space unless the kernel is just the standard inner product $\langle \, , \, \rangle$. The constraints (2.3) and (2.4) are the Karush-Kuhn-Tucker (KKT) conditions. The main characteristic of SVM solutions, with respect to the KKT conditions, is for $\alpha_i > 0$: the Lagrange multipliers, $\boldsymbol{\alpha}$, are greater than zero for active constraints in the dual formulated optimization problem. Each of the training data, $\boldsymbol{x}_{i \in 1..l}$ is associated with a class label with value $+1$ or -1. This information is contained in the binary variable y_i. The classifying hyperplane induced by (2.1) is offset with a bias parameter b. Estimation of the Lagrange multipliers $\boldsymbol{\alpha}$ is conducted by solving (2.2) subject to the constraints of (2.3) and (2.4). The optimization problem is constructed by enforcing $|f(\boldsymbol{x}, \boldsymbol{\alpha}, b)| = 1$ for the support vectors. Support vectors (SV), are those data, $\boldsymbol{x}_i$, which have active constraints, $\alpha_i > 0$. If the data is not separable by a linear hyperplane in a kernel induced feature space, then it would not be possible to solve the problem if there was not an upper limit to the values of the active Lagrange multipliers. Consequently, the constraint, C, in (2.3) ensures that the optimization problem in (2.1) remains solvable. In those situations when a Lagrange multiplier is equal to C, the problem is called a soft-margin problem, otherwise it is called a hard-margin problem. An important term, which is used extensively is *bound* and *non-bound (NB)*. A training datum, $\boldsymbol{x}_i$ is bound if $\alpha_i = C$. Further introduction to linear classifiers and non-linear extensions such as SVMs can be found in Duda et al. [33].

The binary decision function can be extended to a multi-classifier by training $\binom{CL}{2}$ pairwise classifiers, P, and choosing an appropriate voting scheme, such as the majority vote,

$$\max_{P}(CL_m) \tag{2.5}$$

where CL_m is the class voted by the P'th classifier.

The following is the introduction to the Sequential Minimal Optimization (SMO) approach. The notation and setup is based on *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*' by Christianini and Shawe-Taylor [26]. John Platt [73] discovered a method for solving (2.2) without the need for specialized optimization software [73]. His work is central to the use of SVMs in constrained environments. Therefore, we will look at the main formulas as well as a simple toy example.

There are some equations that must be obeyed to train an SVM using the SMO algorithm.

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta} \tag{2.6}$$

The SMO algorithm chooses two data points at a time based on certain selection heuristics. The calculation, see above, of α_2^{new} is based on the previous value α_2^{old} and the difference between the errors E_1 and E_2 divided by η. The variables E_1, E_2, and η are explained below. If the classification problem is not linearly separable, and thus a soft-margin problem, then the following forced clipping of α_2^{new} is needed.

$$\alpha_2^{new,clipped} = \begin{cases} H, & \text{if } \alpha_2^{new} > H \\ \alpha_2^{new}, & \text{if } L \leq \alpha_2^{new} \leq H \\ L, & \text{if } \alpha_2^{new} < L, \end{cases} \tag{2.7}$$

Finally, α_1^{new} can be estimated as a linear combination of α_1^{old}, α_2^{new} and α_2^{old}.

$$\alpha_1^{new} = \alpha_1^{old} + s(\alpha_2^{old} - \alpha_2^{new}) \tag{2.8}$$

where

$$s = y_1 * y_2 \tag{2.9}$$

$$\eta = k(\vec{x}_1, \vec{x}_1) - 2k(\vec{x}_1, \vec{x}_2) + k(\vec{x}_2, \vec{x}_2). \tag{2.10}$$

The value of η in (2.10) is the negative squared distance in the kernel induced feature space.

$$E_i = f^{old}(\vec{x}_i) - y_i = \left(\sum_{j=1}^{l} \alpha_j y_j k(\vec{x}_j, \vec{x}_i) + b \right) - y_i, i = 1, 2 \tag{2.11}$$

The forced interval for $\alpha_2^{new,clipped}$ in (2.7) is calculated as follows:

$$\begin{aligned}
&\text{If } y_1 \neq y_2 : \\
&H = min(C, C + \alpha_2^{old} - \alpha_1^{old}), \\
&L = max(0, \alpha_2^{old} - \alpha_1^{old}), \\
&\text{If } y_1 = y_2 : \\
&H = min(C, \alpha_1^{old} + \alpha_2^{old}), \\
&L = max(0, \alpha_1^{old} + \alpha_2^{old} - C).
\end{aligned} \tag{2.12}$$

The bias b is calculated like this:

$$\begin{aligned}
&b = -\tfrac{max_{y_i=-1}\langle \vec{w}\cdot\vec{x}_i\rangle + min_{y_i=1}\langle \vec{w}\cdot\vec{x}_i\rangle}{2} \\
&\text{where} \\
&\vec{w} = \textstyle\sum_{i=1}^{l} y_i \alpha_i \vec{x}_i.
\end{aligned} \tag{2.13}$$

The geometric margin is:

$$\gamma = \tfrac{1}{\|\vec{w}\|_2} \tag{2.14}$$

The following example is used for clarification because it shows the first iteration of a SMO algorithm and puts the majority of the previously formulas to use: This can be visualized in Figure 2.2.

There are two classes shown in the Figure 2.2. The circle depicts the positive class $y_1 = +1$ and the cross depicts the negative class $y_2 = -1$. The examples are $\vec{x}_1(1, 1)$ and $\mathbf{x}_2(2, 2)$. The exact analytical solution is calculated like this: Initialization:

$$\begin{aligned}
&b = 0, \\
&\alpha_1 = 0, \\
&\alpha_2 = 0, \\
&C = 10
\end{aligned} \tag{2.15}$$

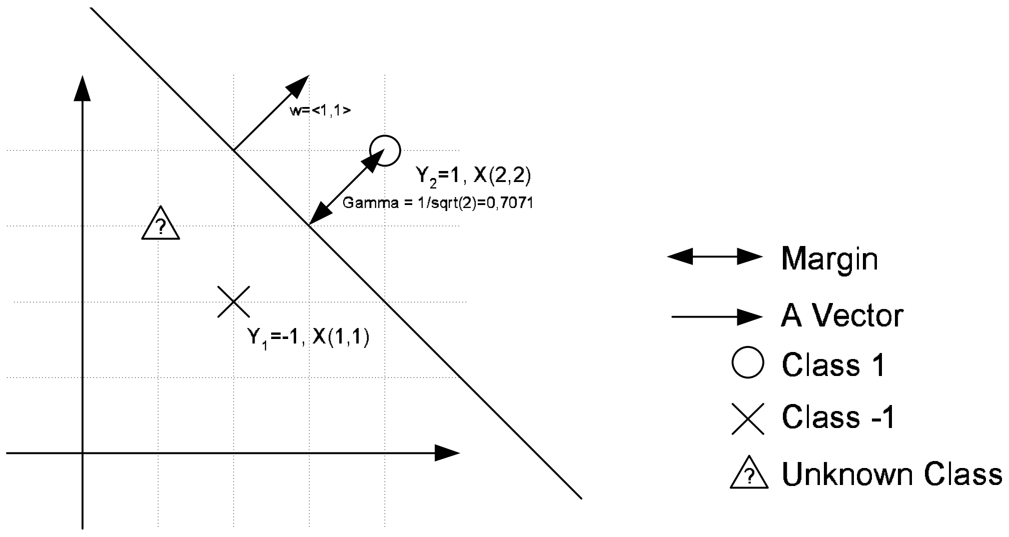

Figure 2.2: A simple two-class example with two examples y_1 and y_2. Here we can see the geometric margin and the weight vector.

then we get:

$$
\begin{aligned}
&y_1 \neq y_2 : \\
&L = max(0, \alpha_2^{old} - \alpha_1^{old}) = max(0, 0-0) = max(0,0) = 0, \\
&H = min(C, C - \alpha_1^{old} + \alpha_2^{old}) = min(10, 10-0+0) = min(10,10) = 10 \\
\\
&E_1 = f^{old}(\vec{x}_1) - y_1 = (\textstyle\sum_{j=1}^{2} \alpha_j y_j k(\vec{x}_j, \vec{x}_1) + b) - y_1 \\
&= (0 \cdot (-1)\langle 1,1\rangle\langle 1,1\rangle + \\
&0 \cdot 1\langle 2,2\rangle\langle 1,1\rangle + 0) - (-1) \\
&= 0+0+0+1 = +1 \\
&E_2 = f^{old}(\vec{x}_2) - y_2 = (\textstyle\sum_{j=1}^{2} \alpha_j y_j k(\vec{x}_j, \vec{x}_2) + b) - y_2 \\
&= (0 \cdot (-1)\langle 1,1\rangle\langle 2,2\rangle + \\
&0 \cdot (1)\langle 2,2\rangle\langle 2,2\rangle + 0) - (1) \\
&= 0+0+0-1 = -1 \\
&\eta = k(\vec{x}_1, \vec{x}_1) - 2k(\vec{x}_1, \vec{x}_2) + k(\vec{x}_2, \vec{x}_2) = \\
&= \langle 1,1\rangle\langle 1,1\rangle - 2\langle 1,1\rangle\langle 2,2\rangle + \langle 2,2\rangle\langle 2,2\rangle = \\
&= (1\cdot 1 + 1 \cdot 1) - 2(1 \cdot 2 + 1 \cdot 2) + (2 \cdot 2 + 2 \cdot 2) = \\
&= (1+1) - 2(2+2) + (4+4) \\
&= 2
\end{aligned}
\tag{2.16}
$$

$$\begin{aligned}
&\alpha_2^{new} = \alpha_2^{old} + \tfrac{y_2(E_1-E_2)}{\eta}\\
&= 0 + \tfrac{(1)\cdot(1-(-1))}{2}\\
&= \tfrac{2}{2}\\
&= 1\\
&L \leq \alpha_2^{new} \leq H \Rightarrow\\
&0 \leq \alpha_2^{new} \leq 10 \Rightarrow\\
&\alpha_2^{new} = \alpha_2^{new,unc} = 1\\
&\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new})\\
&= 0 + (-1)(1)(0-1)\\
&= 1.
\end{aligned}$$

To get the bias we make the following computation:

$$\begin{aligned}
&\vec{w} = \textstyle\sum_{i=1}^{l} y_i \alpha_i \vec{x}_i\\
&= (-1)(1)(1,1)' + (1)(1)(2,2)'\\
&= (-1,-1)' + (2,2)'\\
&= (1,1)'\\
\\
&b = -\tfrac{max_{y_i=-1}(\langle \vec{w}\cdot\vec{x}_i\rangle) + min_{y_i=1}(\langle \vec{w}\cdot\vec{x}_i\rangle)}{2}\\
&= -\tfrac{\langle 1,1\rangle\langle 1,1\rangle + \langle 1,1\rangle\langle 2,2\rangle}{2}\\
&= -\tfrac{2+4}{2}\\
&= -3
\end{aligned} \tag{2.17}$$

The geometric margin is now (see also Figure 2.2):

$$\begin{aligned}
&\gamma = 1/\parallel \vec{w}^* \parallel_2\\
&= \tfrac{1}{\sqrt{1^2+1^2}}\\
&= \tfrac{1}{\sqrt{2}}\\
&= 0,7071
\end{aligned} \tag{2.18}$$

If we try to classify an example point $x_3(0.5, 1.5)$ the functional output would be:

$$\begin{aligned} & f(\vec{x}_3) = \textstyle\sum_{j=1}^{2} (\alpha_j y_j k(\vec{x}_j, \vec{x}_3) - 3) \\ & = 1 \cdot -1 \cdot \langle 1, 1\rangle\langle 0.5, 1.5\rangle + 1 \cdot 1 \cdot \langle 2, 2\rangle\langle 0.5, 1.5\rangle - 3 \\ & = -2 + 4 - 3 \\ & = -1, \end{aligned} \tag{2.19}$$

which is what we would expect by cross-checking Figure 2.2. Pseudo code for the SMO algorithm: Outer loop (first choice heuristic): Alternate with a scan of the full data set and multiple partial scans of the non-bound, NB, (not 0 or C) data set. For each point—from either scan type—find those that violates the KKT conditions (greater than ϵ), and call inner loop for each such violating point. Terminate the outer loop when all points obey the KKT conditions (within ϵ)
Inner loop (second choice heuristic): SMO chooses the second point such that numerator in the calculation of α_2^{new} is likely to maximize the step size. For positive E_1 SMO chooses minimum E_2. For negative E_1, SMO chooses maximum E_2. If no progress is made from first and second choice heuristics then a new non-bound example is searched for, and if that also fails then an entire iteration is started.

The regression SVM will be referred to as ϵ-SVM [93], which refers to a regression problem that takes the form

$$f(\boldsymbol{x}, \boldsymbol{\alpha}^*, \boldsymbol{\alpha}, b) = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) k(\boldsymbol{x}_i, \boldsymbol{x}) + b.$$

The SVM for regression is based on the same principles as the SVM for classification. The difference is the Lagrange multipliers are used in a different way here. There are two Lagrange multipliers associated with each of the points in the joint optimization. Efficient regression with the SVM was extended to SMO by Flake [36] and Flake and Lawrence [37]. Schölkopf et al. [79] introduced a parameter, ν, such that the fraction of points that fall outside the regression tube is at most ν. Steinwart [85] found that optimal value of ν can be set to twice the optimal Bayes risk; should this value be known a priori. Regression with SVMs is not restricted to scalar values as the algorithm has been extended to cover ordinal regression as well [46]. Ordinal regression draws on both the finite set of numbers in feature space as well as the scalar ordering found in the metric space of regression. This type of variable is an ordinal variable.

The third type of SVM algorithm that we want to look at is support vector clustering. The SVM has also been applied to clustering problems that are characterized by the missing label on the data [81]. The algorithm has two parameters that determines the number of clusters and the shape of same. Clustering takes place in the kernel induced feature space [43]. Support vector clustering was introduced by Asa Ben-Hur et. al. in 2001 [6]. The algorithm is also discussed by Schölkopf in [81].

2.4 Training and Testing Complexity of Support Vector Machines

An SVM has a computational cost—when trained—that is exactly related to the number of support vectors. Burges introduced the idea of simplified support vectors [18]. Burges and Schölkopf later [19] improved the speed of the SVM with 20-fold gains. The resulting support vectors would generally not be the same as the original support vectors but this was addressed by a elegant and simple solution by Downs et al. [31] in which those support vectors being linearly dependent were eliminated from the solution and the Lagrange multipliers of the remaining support vectors were modified accordingly. It is common to tune the parameters of the SVM using cross-folding. This further adds cost to the training of the SVM. It has been suggested to use the Lagrange multipliers of a previous SVM to initiate the training of subsequent SVMs [29]. The rationale is that the overall training cost of the SVM is dominated by the cost of the first training on the data set. This is of particular interest when training is performed to reach the leave-one-out error estimate. A straightforward approach [29] is to use the SVM of the N^{th} crossfold directly in the case of a zero Lagrange multiplier. If example i to be tested and the N^{th} crossfold had a non-zero Lagrange multiplier then this multiplier value, α_i, is to be redistributed among the rest of the support vectors.

One distribution scheme was to redistribute the Lagrange multiplier among the non-bound Lagrange multipliers. The traditional SVM classifies points according to the sign of the functional output. Fung and Mangasarian [38] proposed training a classifier by training two parallel planes pushed as far apart as possible such that the positive and the negative points of each class cluster around the two resulting planes. The algorithm is called a proximal SVM, and it seems to have much lower computational complexity than a traditional SVM, but is has been questioned by Ryan Rifkin in his Ph.D. thesis. The basic SVM was improved by Platt in terms of providing a solution that does not need to cache the full kernel matrix or even parts of it [73]. This basic algorithm was improved be Keerthi et al. by using two threshold values instead of one [60]. Several approaches exist to reduce the memory requirements of the SVMs, but in work done by Mitra et al. a method was suggested that successively draws small samples from a large database while training and keeping an SVM classifier [65]. Similar work has been done to speed up SVM classification by making parallel implementations of the SVM [74]. The approach is to exchange raw data blocks among the nodes in the system and exchange the raw data to obtain the same solutions as the stand-alone SVM algorithm would have done. Chapelle and Vapnik proposes to use a gradient descent algorithm related directly to the desired generalization performance of the SVM [24]. This allows for finding multiple parameters concurrently. The method is to find the derivatives of the parameters with respect to the margin.

2.5 Distributed Systems

Distributed computer systems is the setting of a number of computational units participating in some task. It often involves the exchange of data between these computational units. The data can be associated with methods that are appropriate for using the data and in those instances it is called an object. Early work has been done by Jul et al. [55] [54] [53] on using objects in distributed systems. Their system is named Emerald, and it supports the autonomous use and movement of objects with the system. There is support for a number of refinements such as grouping of data before the objects are exchanged. Later work on Emerald garbage collection was done by Juul [56] and Juul/Jul [57]. In this thesis *garbage* is data that is not really needed for the statistical models to work in the distributed setting, while in Juul's thesis it was the ”dead” objects that were the aim of the analysis.

The framework is applicable as a resource aware protocol in distributed systems. We explain how it can be used in a purely distributed mode as a P2P system or a sensor network, but the system can also be used for traditional client/server models, and we show how that could be arranged.

In the distributed systems that we consider, there is seldom an unlimited amount of computational power available. Working with constrained environments is a relative measure that takes meaning only when considering the problem to be solved. For applications that involve machine learning and possible large amounts of data, the distributed systems can without simplification be regarded as constrained devices. Some distributed devices do not have long expected battery lifetime.

Sensor systems are a type of distributed system that usually are associated with small computational devices that can communicate using some wireless means. These systems can be battery driven, and thus the computations taken by these systems can be expensive in terms of CPU cycles. One popular system is the system offered by Crossbow named Motes. These units run the small, open source operating system TinyOs. Sensor networks are often ad hoc networks that self-assemble in the given context [45]. It has been reported that the wireless part of ad hoc sensor networks could be addressed using an appropriately configured Bluetooth stack [62]. The results show energy consumption per bit sent. The information and data flow in sensor networks can be structured by incorporating some of the strengths of SQL such as was done in COUGAR by Bonnet et al. [14]. This has been worked on by Madden et al. [63] in a power-aware query extension for TinyDB. Identifying targets in distributed sensor networks [16] can be done. The work centered on tracking vehicles across sensor cells based on acoustic inputs. Some authors [17] have introduced the idea of using SVMs for sensors but without addressing the distributed nature or utilizing the sparseness or addressing energy awareness of the solution to reduce inter-node communications.

As the distributed systems become more and more aware of one another, we can use the term peer-to-peer (P2P) [67], which means that connected computers do not have centralized servers as the main paradigm as the widely popular client-server model depicts. However, there can be situations in which it is necessary to create roles that, at the local level, resemble a client-server system. JXTA, a distributed Java system, operates with roles that gives more responsibility to certain nodes in the system and not to others. A topic that is related to P2P computing—as well as sensor networks—is the notion of security. To address this concern in a systematic manner, Zhou and Haas [101] propose to work the problem using the following set of properties: availability, confidentiality, integrity, authentication, and non-reputation.

In distributed settings, and especially for sensor networks, energy aware implementations of machine learning algorithms play a key role. Energy consumption can be split into the energy that is used on computational issues and the energy that is used for communication purposes. Important data mining algorithms have been tested for energy consumption on a PDA. Bhargava et al. [8] concluded that distributed power consumption of executing principal component analysis and k-means clustering on a data set from a mobile vehicle data collection system is more energy saving than first transmitting the data and then running the algorithms at a centralized location [8]. At the language level, the standard Java JVM does not offer any resource awareness or control mechanisms in terms of CPU usage and communication bandwidth usage. Some modifications that address these issues are described by Binder et al. [10] and also from a security perspective by Binder and Roth [11]. A small survey of reduction techniques in instance-based learning algorithms can be found in Wilson [96]. He also draws parallels to early work done by P.E. Hart on the condensed nearest neighbor rule (CNN).

We also investigate research work in terms of energy aware sensor networks. Then the choice of intelligent algorithm is investigated to understand how the SVM can assist the local and distributed decision-making on the sensor nodes. It appears that more research is conducted on the TinyOS sensor networks than on Java-based sensor networks [48] [62]. However, energy awareness in wireless sensor networks focus to some extent on the routing aspects of the sensed data [75] [62]. The reason for this can lie in the observation that sensor networks are deployed in groups formed by a large number of individual nodes. An approach similar to the one opted for in this chapter is to use local novelty detection algorithms to limit the radio communication and data transmission to and from the server [72]. This interesting approach is formalized to a larger extent by the energy-aware clustering algorithm by Ghiasi et al. [42]. Our measurements on the individual nodes can be used as input to the basic configuration of the routing algorithms in instances in which a local inference is appropriate. The local inference can be performed by SVMs [15] as this class of algorithms are founded in statistical learning theory [93]. The idea of using SVMs on sensor nodes is introduced by Jordaan [52]. In this work, sensors are not analyzed in terms of the constraints often present when working with sensors: memory constraints, computational constraints, and battery constraints. The main part of mapping the support vector algorithm to constrained environments is addressed by Pedersen [71].

2.6 Distributed Java

Java is an interpreted language that runs using a virtual machine that is running in native code by some system. This level of abstraction from the hardware and native software comprise Java's strength in terms of portability. On the other hand, the use of a virtual machine also sets some minimum requirements for the hardware. For example, the KVM from Sun Microsystems requires 40 kilobytes to 80 kilobytes. It allows programs to span heterogenous environments. It does not matter if the host is a Windows machine or a Symbian mobile phone. This makes the code portable when the programmer needs to change the executing environment. More interestingly, in the distributed systems settings the Java language makes it possible to connect a system that spans multiple operating systems and multiple connection protocols. The other alternative of using hardware specific code can limit the portability of the application.

In this section, the Java programming language is discussed and some of its main characteristics are mapped to the context of the DSVM. Java is a popular language that was incepted as a Sun-sponsored project in 1991. It had the codename Oak. The first application aims were TV devices, but that did not succeed as such. However, the first versions of Java that were available for download became hugely popular (even to the extent that some people believed Sun was blocking the downloads), but the real explanation was that the system was so popular the communication lines that it used were saturated. Today, Java has become become more than a language. It is widely popular within with specific implementations ranging from micro devices to enterprise applications. One of the goals of a distributed inference system is that it can work over a large range of devices, which is why Java is a viable choice. It seems to have its merits in the academic world as well as in the commercial. At the time of writing Sun has released version 1.5 of its popular Java virtual machine. It is likely that the JVM will come on successively smaller chips in the future in some micro version under the product line that Sun calls Java Micro Edition. The mobile phone that has been used as a testing unit comes with a built-in JVM that adheres to the specifications for a Personal Java, or pJava, device. It means that the Swing libraries are removed and the programmer then needs to use the abstract windows toolkit, AWT. The open source community is actively developing new JVMs that comes with freely available source code. There is even an open source FPGA Java processor implementation available [78]. Another aspect of Java that makes it useful in distributed computing is its networking capabilities. Object level communication is built into the language to enable objects residing on different JVMs to communicate without the need to perform message passing on the socket level. The built-in solution in Personal Java and the standard packages is called Remote Method Invocation (RMI). RMI makes it possible, with few extensions and modifications, for the code to have objects on different servers communicate as if they were running in the same address space within the same JVM. That mechanism provides the level of abstraction that is needed to strike the balance between a working DSVM prototype and the desire to keep much of the focus on the functionality rather then on the implementation details. RMI is flexible in terms of how it can be used. The three main services provided by RMI are the naming service, the remote methods invocation service, and the distribution of objects. The naming service works such that an object that has been prepared for remote invocation is bound to an URL-like address. Clients can obtain a remote reference to this object via this address. To enable clients to make calls on the remote object, they need to know how this object is represented. That is done via a Java interface that describes the remote object to the client. RMI provides the service of downloading all these objects over the communication line the client is using, and that makes it easy to control versioning, even in a distributed environment. The third service that is provided by RMI is the functionality surrounding the remote method calls. It involves marshalling the parameters and unmarshalling the parameters on the remote side and vice versa. These details are hidden for the programmer, who only needs to implement objects that can be serialized. In this way, a complete distributed system can be created with small naming servers and clients who only collect, calculate, and distribute.

Should the synchronous method call be prohibitively slow, then the call-back ability of RMI can be used. Call back allows the caller to leave a reference with the remote object such that it can be called when some event occurs, such as the completion of the method call just mentioned above. It is possible to use applets to implement the call-back functionality. It has even been proposed that the remote references involved can be distributed to all nodes to provide a remote address book [5]. Finally, the RMI method calls automatically serialize the arguments in the method call should they not implement the required *UnicastRemote* interface. These three mechanisms leave it to the programmer to construct the distributed invocation profiles that are appropriate in the given instances.

There are a number of Java peer-to-peer (P2P) projects defined. One open source project that stands out is the JXTA project [25]. It is a P2P system defined in Java that allows for interaction of messages in a network of JXTA nodes. The JXTA system incorporates the service aspect such that the JXTA nodes can either provide the service to relay the request to a node that provides such a service.

The existing body of literature is large on the subjects that have been covered in this chapter. We discussed the main publications in terms of the SVM and its development to the mature algorithm it is today. In addition, important subjects within distributed systems such as sensor networks were briefly covered. Now, we start to use the reviewed literature to address the research questions stated at the beginning of the thesis.

Chapter 3

Single-Node Support Vector Machine

On the basis of our research to date, we believe that statistical learning theory—and especially the derived SVM—can be a valuable contributor for the distributed data mining (DDM) community. We hope to gain acceptance for this approach by presenting a framework of how SVMs could address selected DDM problems. Our hypothesis is that the SVM algorithm provides enough flexibility to be configured and tailored specifically toward many resource-constrained problems. To our knowledge, except for own work [70] [69], no similar efforts have been presented previously using SVMs in DDM while respecting the constrained nature of distributed nodes, but we hope this changes to some degree with our introduction of the resource-aware distributed support vector machine, DSVM. The DSVM addresses inherently distributed problems, which differ from the parallelization efforts laid forth by F. Poulet [74]. The most similar work is by Jordaan [52], in which she focus on the sensors as individual remote inference machines.

The mapping of SVM-related tools to the CPU and memory constraints are the main theme of this chapter. In later chapters, the focus will be directed to the communication segment of the system. In Table 3.1 we can present some SVM tools, which are discussed below. The main tool for saving on CPU cycles is the SMO algorithm, which is adapted to the Keerthi modifications (see Table 3.1 for references). When we work with non-linear kernels it is generally necessary to use

Table 3.1: Mapping SVM Tools to Constraints

Constraint	SVM Mode: Training	SVM Mode: Testing/Filtering
Computational	SMO(Keerthi Mod. 2 [60]) $\boldsymbol{\alpha}$ reuse [29] Relaxed KKT conditions, section 3.6	Generalized SVs [18] Orthogonal SVs [31] Linear SVM [93]
Memory	SMO [73]	ν and δ exchange, section 4.8

cross-validation to estimate model parameters to minimize the error on the validation set. Contrary to MLP NN, the SVM arrives at an exact solution, and it is thus possible to reuse the weights of the model, which are the Lagrange multipliers $\boldsymbol{\alpha}$ in (2.1). A method for the SVM to achieve its stopping condition faster is to relax the KKT conditions (see (2.3) and (2.4)), and this is also a topic of this chapter. The memory constraint of the distributed node is addressed by our choice of the SMO algorithm over a traditional approach involving a dedicated software package. In terms of testing and filtering tools, we reduce the time for testing new points by utilizing the concepts of general or orthogonal support vectors. Obviously, if the SVM is linear, the number of kernel evaluations is reduced as a weight vector in input space can be constructed. Finally, the single-node memory requirements can be controlled with a scheme we call ν and δ exchange, but this is included in the table for completeness and discussed in the next chapter.

The significance of the DSVM system is reflected in the discussion regarding the future directions of distributed data mining by quoting H. Kargupta, C. Kamath, and P. Chan, chapter 15, ”Regardless of the increase in the bandwidth capacity, the ratio of the bandwidth and the quantity of available data is likely always to be low. Therefore, downloading large amounts of raw data may not be a practical option for mobile data mining applications. Because DDM often requires as little [network] communication as possible, it is likely to play a significant role in mobile environments.” [59]. The DSVM can greatly reduce the exchange of data, which could mark it as one of the systems perhaps referred to above. Similar goals are addressed by D.R. Wilson and Tony R. Martinez in their paper on storage reduction techniques for instance-based algorithms [96].

The Distributed Support Vector Machine is a specification and reference implementation for a practical framework of applying kernel-based learning to some (distributed) learning task. Within the DSVM framework, the fundamental idea is that a set of SVMs work cooperatively, yet independently, in a digraph (vertex/edge)-based distributed environment toward shared optimal generalization ability. The focus of this section is on how the SVM can be adapted to fit on a distributed node subject to a number of system constraints.

3.1 General and Orthogonal Support Vectors

There are two related approaches in Table 3.1 for reducing the testing complexity of SVMs: generalized support vectors (GSV) [18] and orthogonal SVs [31]. Chris Burges's approach [18] generates a new set of vectors that are neither necessarily SVs nor do they have to lie on the separating margin. Furthermore, the desired number of SVs is specified a priori. With reference to (2.1), the weight vector in feature space is represented as a linear combination of the mapping function Φ:

$$\boldsymbol{\Psi} = \sum_{j=1}^{l_{SV}} \alpha_j y_j \boldsymbol{\phi}(\boldsymbol{x}_j) \tag{3.1}$$

It is possible to approximate $\boldsymbol{\Psi}$ in (3.1) by a reduced set of data points [18] to obtain an approximated Ψ'. The aim is to reduce the number of data points while keeping the weight vectors in feature space as similar as possible:

$$\rho = \|\Psi - \Psi'\| \tag{3.2}$$

The difference between the original Ψ and the estimated Ψ' is ρ. The algorithm that determines Ψ' can be found in the Burges paper [18].

Tom Downs *et al.* [31] presented a different approach using orthogonal SVs with better mathematical properties. Instead of identifying a set of new points in feature space, this work exploits the fact that support vectors are usually linearly dependent in feature space. It is expressed as $k(\boldsymbol{x}, \boldsymbol{x}_k) = \sum_{i=1, i\neq n}^{l_{SV}} c_i k(\boldsymbol{x}, \boldsymbol{x}_i)$:

$$f(\boldsymbol{x}) = \sum_{i=1, i\neq n}^{l_{SV}} \overline{\alpha_i} y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \tag{3.3}$$

Note that $\overline{\alpha_i}$ has been modified to the smaller number of SVs as the SV with index n was eliminated using row reduced echelon form. The main difference between these two approaches is that the Burges approach takes a parameter that determines the resulting compression and Downs's method automatically searches for linear dependence in feature space. Compression, as used in this thesis, is the same as filtering. For example, a compression of 90% is to be read as just 10% of the data being exchanged. The advantage of Downs's approach is the decision surface is exactly the same as before the compression, which means that the classification or regression model does not suffer a reduction in performance as measured by the error rate. An advantage of the Burges method is that the desired reduction can be specified as a parameter, which would enable a forced fit into significantly constrained devices, should that be an issue.

3.2 Support Vector Machines in Constrained Environments

The sparse solutions produced by SVMs make them important components for distributed data mining problems, especially those related to classification, regression, and clustering—as presented by the author at the COLT03 kernel impromptu poster session [69]. In this section we extend the framework to address computational, communication, spatial, and temporal constraints. A distributed digraph of SVMs, closely related to the virtual, ν, transductive, and sequential minimal optimization SVMs, is combined with the concept of generalized support vectors to produce two basic types of SVMs: one that filters data to reduce edge exchanges and another that optimizes a cost function using the reduced data set. In addition, the alterations on the support vector set by a controlled relaxation of the Karush-Kuhn-Tucker (KKT) conditions are discussed in the context of employing the recall and precision metrics commonly found in information retrieval settings. We defer discussion of the communication constraints until the next chapter. There could be advantages of combining machine learning methods with the emerging peer-to-peer (P2P), mobile computing, and ubiquitous computing paradigms. However, these paradigms could differ substantially from personal computers if compared along the dimensions of computational, communication, and spatial capabilities.

The resource constraints in our framework include, and to a certain extent focus on, the minimization of the network traffic among the constrained nodes. Therefore, the constraints in this system are not limited to the stand-alone nodes but rather applicable to the full digraph of constrained, interconnected nodes. We discuss how the data exchange addresses the constraints of the sender (producer node), the aim of minimizing edge traffic, and the task of the destination node (consumer node) to create a good SVM model.

3.3 Constraint Proxies in Single-Node Setting

The node parameters of interest in this section are CPU speed, memory availability, and battery life.

To achieve a level of abstraction from the physical constraints and costs, we propose mapping the categories to a context that is easier to configure in terms of the learning machine. This allows a principled discussion on the parameters of the DSVM system and the SVM model that can be modified to address various constraints. The main cost categories are computation, spatial, temporal, and communication. Computational costs are mainly perceived as costs associated with CPU cycles for reasons such as limited power supply. Spatial costs are some combination of the cost of using the available memory for the model and how much memory there is available. Temporal costs are those associated with a slow model, and this can translate into a constraint on the maximum time the model can train before it is ready for use.

Counting kernel evaluations is a relevant measure, especially when the SMO algorithm is used. Therefore, we will use the number of times the kernels are evaluated on a given node to derive a number to evaluate kernel caching schemes, etc. The SVM implementation named LIBSVM [23] uses 40 MB for caching as the default setting. One reason for such a direct approach is that caching would generally be much faster to access than kernels computed from time to time. Furthermore, some kernels are more expensive to use than others. For example, the Gaussian kernel uses three dot product evaluations without caching, while the standard dot product kernel apparently amounts to one dot product calculation.

On resource-constrained devices we often find limited memory availability. Therefore, the data set size of the full training set can be a measure of how well some space constraint is met. Another resource sink could be the lack of kernel caching. The Gram matrix, which holds the kernel outputs $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, spares some computational units at the expense of a larger memory footprint. To evaluate how well the system filters data, we compare the initial data set size as well as the filtered data.

In some instances, the model would need to complete the training within a certain time slot. The time it takes from when the first node starts training until the final node is trained—i.e. a steady state situation—is a proxy for how well the system would address such constraints/demands. For this reason, the time it takes the DSVM system to reach steady state is a proxy for temporal costs.

As this section addresses a system that is subject to a number of constraints, the discussion in this subsection centers on the situation in which the size of the Gram matrix would be too large to fit within the available memory. The usual approach would be to use some kernel caching scheme that seeks to achieve the highest possible hit rate of cached kernels vs. repeatedly calculating the kernels. One suggestion is to cache the working set, which is the set of vectors with non-zero Lagrange multipliers. Another alternative approach is to allocate a certain amount of memory and then use a simple FIFO (first-in-first-out) kernel-caching scheme.

The generalized support vector (GSV) concept [18] is aimed at simplifying the models represented by a full set of support vectors. Advantages to this idea include the fact that the final model becomes faster than the original model, which would have used a larger set of support vectors for classifying new observations. Experiments by T. Downs el al. show that creation of linearly independent support vectors in feature space can lead to reductions in number of SVs [31]. An additional advantage to their approach is that the memory consumption is smaller than for the original model. If the inference node would perform a high number of classifications, then the total number of kernel evaluations would likely be smaller than if the model was based on the original set of support vectors.

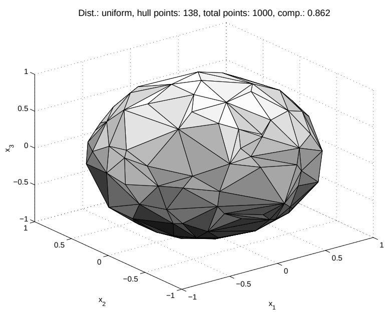

Figure 3.1: A plot of a convex hull for three dimensions. The idea conveyed in this figure is that the number of points on the convex hull is substantially lower than the total number of points. In this particular instance, the compression rate is 0.862, which means that out of the 1,000 points, 138 lie on the hull surface.

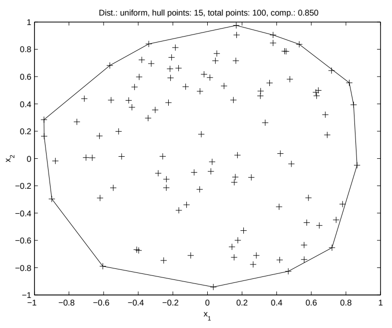

Figure 3.2: A plot of a convex hull for two dimensions. In this example, the compression is 0.850 for a two-dim uniform distribution. This suggests augmentation of the exchanged support vectors with convex hull points is inexpensive in terms of network traffic.

We will use the fact that the kernels induce a distance metric, $D_{12} = k(1,1) - 2k(1,2) + k(2,2)$, in feature space to identify points that are likely to be on the margin. Our concept addresses convex hulls of circular, ball (figure 3.1), or hyper sphere shape. On more complex shapes, it would probably not be an accurate object. First, one sorts the $D_{ij}, i \neq j, i, j \in 1, 2, ..., n$ for the pairs of

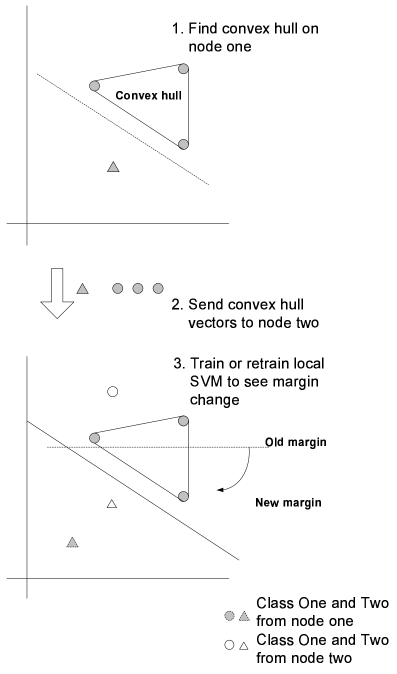
1. Find convex hull on node one
Convex hull
2. Send convex hull vectors to node two
3. Train or retrain local SVM to see margin change
Old margin
New margin
Class One and Two from node one
Class One and Two from node two

Figure 3.3: SVM classifier for two nodes with a convex hull. A producer node finds the convex hull of the circle class points, such as in Figure 3.1, and exchanges this information along with the support vectors to the destination node. Note that only one extra point was exchanged as the two other circle class points already were support vectors and thus subject to exchange. Finally, the consumer node receives the points and retrains using the new information.

points and ensure that no point P_i or P_j participated in the list of pairwise distances more than once. Secondly, the approach can be optimized by using only a predefined fraction of the sorted points to address possible external constraints related to memory or reduction of edge traffic. It can be noted that this is similar to the idea of the ν-SVM [80]. This simple convex hull approximation facilitates the exchange of approximated convex hulls along with the support vector set if permitted by the set of constraints. A visualization is displayed in Figure 3.3.

Table 3.2: The Constraints and the Proxies

Memory	Tool	Cost Proxy
CPU	SMO, GSV, Convex Hull	#k
Time	KKT relaxation	Convergence time
Communication	SV, α ordering	Count data points

It is will not always be necessary to have an accurate model for predictions *per se*; it is sufficient that it identifies the relevant support vectors and sends them to a consumer node. Therefore, the producer node will only need to find the support vector set, and it would not matter if this was done with KKT accuracy of 10^{-2} or 10^{-5}. It is clear that this is a candidate for experiments to see the extent to which the support vector set changes as a function of the KKT conditions. A relevant stopping criterion would be to cease training when the working set has stabilized.

An advantage of the SVM is the ability to work with the input data using just the Gram matrix, which is the matrix of dot products for linear SVMs or the matrix of other valid Mercer kernels. The input space can be high dimensional, of which text categorization [51] is one example. The array of stemmed words can be a possibly large object leading to high memory requirements to store the individual elements of the array. One approach that spares memory on the distributed nodes is to save only information [73] for those dimensions in which the attributes are information bearing. This approach requires two arrays to keep track of the indices in the attribute vector with actual values. The first array is an index array with the integer value of the original attribute index. All the actual values are kept in the second value array, so even if an an attribute vector is of size 1,000, then the actual storage requirements for that vector is at most twice the size of the vector given that all of the slots are filled out. In the worst case, the sparse storage mechanism can lead to more memory consumption than the usual single-array storage mechanism. A flexible method could be to mark each record as a candidate for sparse storage, and then activate each storage mechanism, if needed. The sequential access to each attribute using the sparse solution with only two storage arrays as described above is not a problem. However, when random access to the attributes are needed, then a third mechanism must be introduced. This mechanism is a map that links the original attribute index to the index for which it is stored in the sparse index array. For example, the 20^{th} attribute might be stored in the 11^{th} position in the sparse index array. The introduction of this third storage mechanism puts an additional overlay into the sparse implementation.

3.4 Global and Local Datasets

In this section, we will evaluate the differences in datasets that might occur in the DSVM system. The system is basically comprised of two kinds of SVMs that run on two types of nodes. A filtering SVM runs on a node labelled as the producer node, and the inference SVM runs on something called the inference node. In this context, it is easiest to comprehend scenarios in which the data is collected on the producer nodes and thus would not be overlapping. In such a case, the producer nodes would filter and then exchange the data. However, in some scenarios the observations are not spatially separated. Observations may overlap on a row basis or on a column basis. The SVM is likely to be well suited for row data that is repeated throughout the system. In that case, the producer SVM would locate the exchange set and, at the consumer nodes, a preprocessing step would eliminate the repeated data. Therefore, the system would perhaps see repeated data coming from the producer nodes toward the consumer nodes. It some circumstances it might be necessary to introduce a new layer of producer nodes, which would work as a relay node. This node would act as a relay of the information that was not repeated. Repeated information would be lost in the training session of the relay node, and thus there would not be extra traffic on the final edges leading up to the consumer node. A more problematic scenario could be imagined if the system of sensors were monitoring an object from different angles. The problem is the last columns on one sensor might become the first columns on the next sensor's data. There would be no easy way to eliminate this kind of data as it is incorporated into the final vector that needs to be formed. The local datasets might be small enough to work with a notion an edge node. We envision a new type of node that we could call a hyper-edge, which would not run an SVM but would keep some kind of cache that enables it to eliminate repeated observations before they are relayed to the real producer node. After receiving the data from the nodes, the producer nodes would send the exchange set to the consumer nodes, which then would perform the inference tasks on the datasets available in the system. This allows for a solution that maps toward the Java frameworks. First we have hyper-edges that perform local inference tasks. Then close producer peers filter the data by loose training of SVMs. Finally, the exchange sets are migrated to a server, which finalizes the model in a consumer node setting. A different scenario follows if this is viewed as a distributed system of autonomous peers. In this setting each peer would go through two states: training, in which they capture data and exchange it with the consumer nodes, and a state in which data is received from the producer nodes and the role of the consumer node is assumed. Upon final training of the model, the nodes would become consumer nodes, and each node in the system could participate in distributed inference. It would also be possible to update the nodes online after the models were trained, an advantage of SVMs. A BP neural network would not lend itself as easily to retraining because it typically has higher capacity than used in the model. Therefore, it is difficult to unlearn and learn in partial steps with a BP neural network. In the SVM, the operation merely amounts to injecting the data point with a reset Lagrange multiplier and commence training.

3.5 Experiments with Code Profiling of DSVM Gui

The first experiment is to profile the DSVM [32] code in a testing mode. The DSVM interface allows the user to enter data for a binary classification problem, such as the choice of kernel. Subsequently, the SVM is trained, yielding the results to be evaluated. To distinguish the decision regions, the DSVM classifies and colors each of the pixels on the DSVM gui screen. This is achieved by converting each pixel to a set of coordinates and calculating the functional output from the DSVM. The color is then presented in RGB by making functional output closest to the decision region blue or red depending on the sign of (2.1). The red or blue component is then decreased such that the decision regions have a dark color in areas in which the model is more certain as measured by a large functional output.

The DSVM gui in Figure 3.4 presents the DSVM in two modes: training and classifying. These two modes become the unit of analysis with respect to a code profiling of the DSVM core.

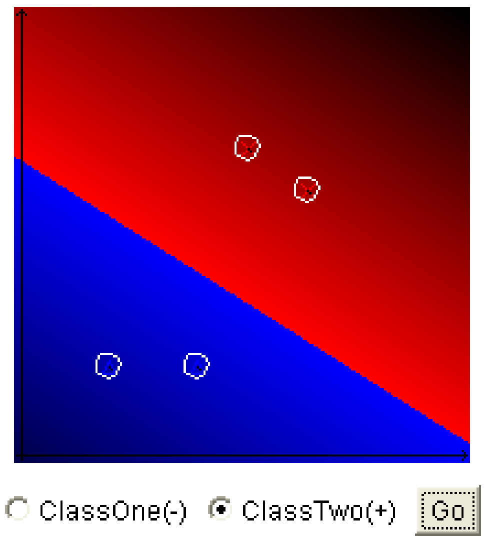

Figure 3.4: A binary classification problem. The DSVM first makes the model, then classifies a 208 by 208 pixel area pixel for pixel. The 43,264 classified pixels are then used to color the background.

A code profiler is available to the Eclipse IDE, which can point out how much CPU time is spent in each method. It can thus create a profile of CPU time is spent in the DSVM. This tool can be used to identify bottlenecks in the code. There are dominant program states that can be profiled to provide insight into the code. These states include training and testing. Training consists of running the loop of pairwise kernel evaluations in the SMO loop continuously until the stopping criteria is met. The configuration in Figure 3.5 is a code profile without any caching of the kernel. As expected, a majority of the time is spent evaluating the kernel, as this is a dominant object in (2.1).

3.6 Experiment with Stability of DSVM for Different Kernel Configurations

The SVM algorithm relies on the kernel to measure the dot product of two data instances in the induced feature space [26]. A widely used kernel is the Gaussian kernel $\frac{exp(-\|x_i,x_j\|)}{\sigma^2}$ that has one parameter. The parameter is the width of the Gaussian, σ. Another important parameter is

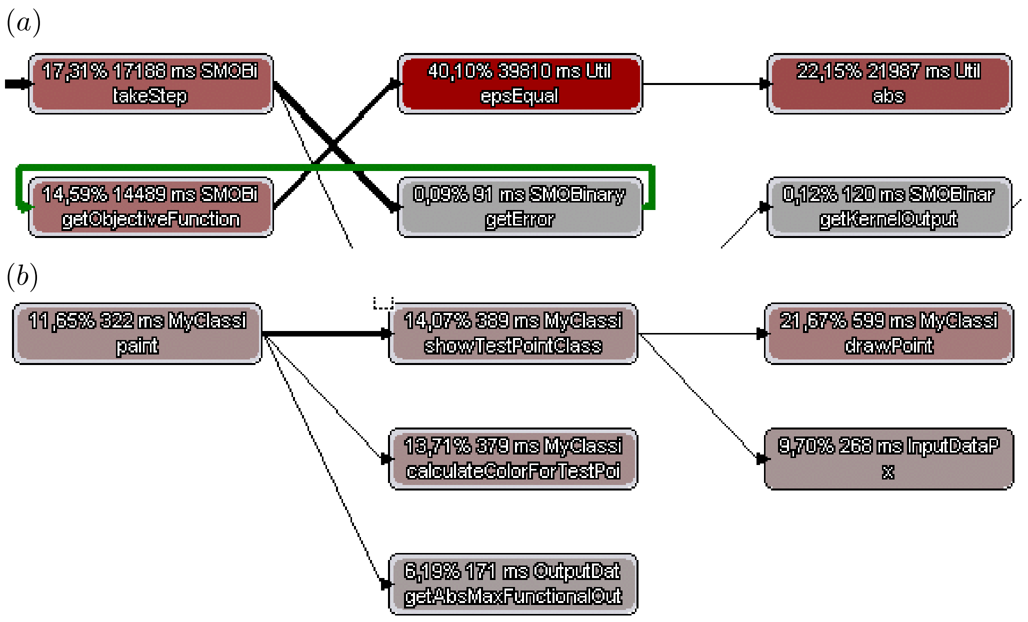
(a)
17,31% 17188 ms SMOBi takeStep
40,10% 39810 ms Util epsEqual
22,15% 21987 ms Util abs
14,59% 14489 ms SMOBi getObjectiveFunction
0,09% 91 ms SMOBinary getError
0,12% 120 ms SMOBinar getKernelOutput
(b)
11,65% 322 ms MyClassi paint
14,07% 389 ms MyClassi showTestPointClass
21,67% 599 ms MyClassi drawPoint
13,71% 379 ms MyClassi calculateColorForTestPoi
9,70% 268 ms InputDataP x
6,19% 171 ms OutputDat getAbsMaxFunctionalOut

Figure 3.5: Code execution profiling of the DSVM using an Eclipse profiler plugin [77]. The top image (a) shows the training profile. CPU time is consumed primarily by the error calculating flow that uses `takeStep`, `getError`, `getObjectiveFunction`, `epsEqual`, and `abs`. This flow sums to 94.24%. The bottom image (b) shows the display profile. The percentages on the picture show how CPU time is used by the methods in the code. To display a background color, the DSVM tests each pixel against the decision function provided by the trained SVM. If the percent of time spent in the `showTestPointClass` and the two sub methods is added to the percent of time spent in `calculateColorForTestPoint`, it sums to 59,15%.

the regularization constant C. It determines the penalty the SVM places on misclassified points. To get experimental insight into the SVM training time for different combinations of C and σ, a data set from a large public repository is chosen [13]. The Ionosphere and the Wisconsin Breast Cancer data sets are analyzed in terms of how many kernel evaluations are computed before SVM convergence. What we are interested in here is not the absolute number but rather the variation of kernel evaluations. A large variation could indicate unpredictable energy consumption on the node.

It is evident that the kernel evaluations vary to some degree depending on the kernel and SVM configuration with respect to Gaussian kernel width and the regularization of parameter C.

The stopping criteria of the SVM can also be a way to save energy on the node [71], and, therefore, we want to experiment with the tolerance that determines if the SVM obeys the Karush-

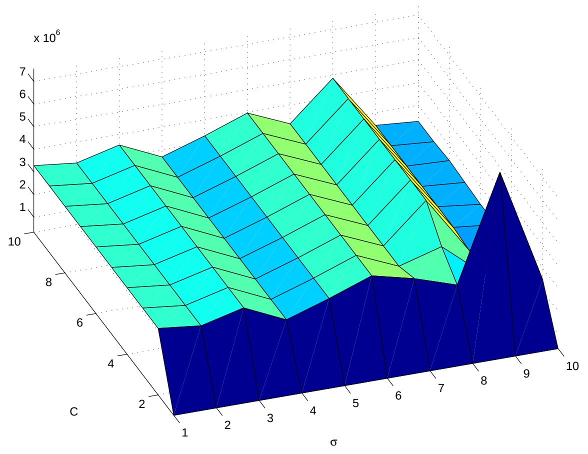

Figure 3.6: Counting kernel evaluations for the Wisconsin Breast Cancer data set. The computational burden, as measured by the number of kernel evaluations, is stable in the majority of the search space. The one exception is the observation of 7,565,601 kernel evaluations for $\sigma = 9, C = 2.0$.

Kuhn-Tucker (KKT) conditions. The larger the tolerance, the fewer kernel evaluations are needed to make the SVM converge to the specified level of accuracy.

This is an interesting result as it also shows how the DSVM can be configured to run in a more or less fine-grained mode and thus presents an additional source for tweaking the energy consumption on the node. The smaller the tolerance, the more energy the DSVM is using. This result is not energy profiled in terms of how much energy it uses, but instead it is profiled using the number of kernel evaluations, $\#k$, which is customary in machine learning. We use the modified SMO [61] to estimate the Lagrange multipliers, $\boldsymbol{\alpha}$.

3.7 Experiments with Reuse of Lagrange Multipliers

A relevant experiment is to save device energy by reusing the models generated by previous cross evaluations [29]. In contrast to backpropagation neural networks, previous SVMs can be reused in training of new models. MLP neural networks are often able to overfit the estimated function, and therefore the weights need to be estimated from scratch for each training in the crossfold. An experiment on how many kernel evaluations it takes to train an SVM using a Gaussian kernel on

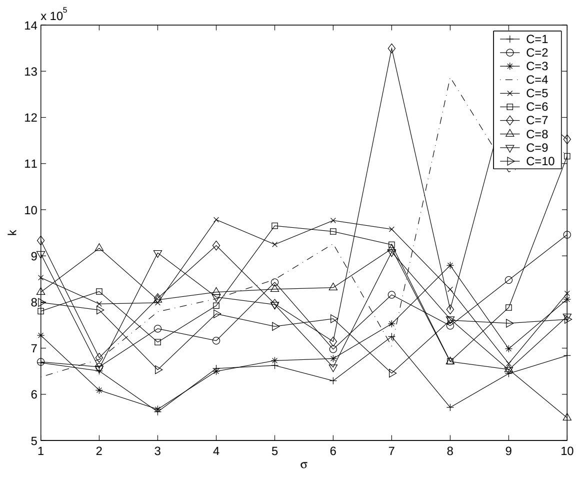

Figure 3.7: Counting kernel evaluations for the Ionosphere data set. A classification SVM model is trained on the Ionosphere data set for different values of C and σ. There is a peak in the number of kernel evaluations for some combinations.

the Ionosphere data has been conducted. We counted the kernel evaluations for training SVMs for all combinations of Gaussian kernel widths σ of 1 through 10 and the regularization parameter C from 1 to 10. Both values are stepped in increments of 1. This results in 100 runs of the SVM. The method used is to re-initialize $\boldsymbol{\alpha}$ for new kernel parameters while using an inner loop of new successively larger C values. A rescaling of $\boldsymbol{\alpha}$ is conducted between each inner loop by multiplying $\boldsymbol{\alpha}$ by C_{new}/C_{old} [29]. On the Ionosphere data the experiment for $tol = 0.01$ without reuse of alpha scaling resulted in 79,654,146 kernel evaluations. Through reuse and rescaling of $\boldsymbol{\alpha}$, the total number of kernel evaluations was 49,337,258, which is an potentially important source of energy savings.

3.8 Experiment with Gaussian Approximation

The Gaussian kernel,$k(x_i, x_j) = \frac{exp(-\|x_i,x_j\|)}{\sigma^2}$, is often chosen when using SVMs. It has one parameter in the form of the width σ of the Gaussian function. Java J2SE provides the Math.exp function to take the exponential of a number. However, under the Java CLDC, this function is no longer available. Consequently, an approximation to the exponential function is needed. The

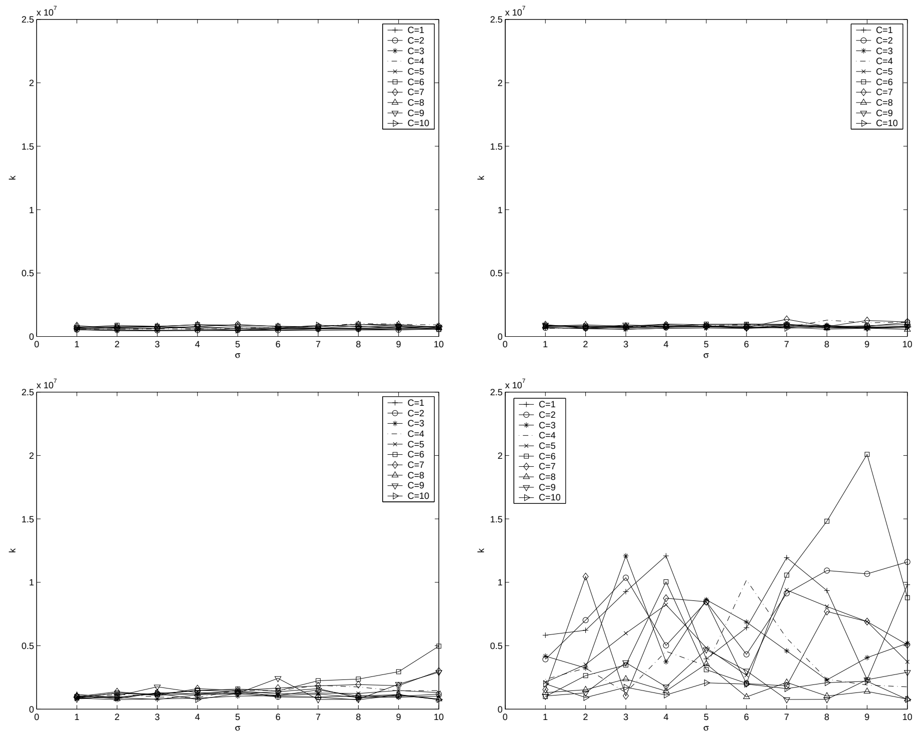

Figure 3.8: The datapoints of the Ionosphere data for different tolerances. The number of kernel evaluations increases as the required tolerance becomes smaller. The smallest and most restrictive tolerance t is for the bottom figure, which has $t = 0.001$. The other figures have $t = 0.01$, $t = 0.1$, and $t = 1.0$, with the largest value at top left. On the x-axis is the width of the Gaussian kernel function, which controls the classification performance of the SVM classifier. On the y-axis we have the number of kernel evaluations needed to train the SVM classifier to the given tolerance t.

implementation can be approximated in the form of a Taylor series expansion. Our implementation of the function is in the `exp` function in the `Util` class in the `dsvm.util` package [32]. For a configuration in which the first 12 components are added, the system yields better performance as compared to the Math.exp function. For 1,000,000 runs the Math.exp function took about 771 ms, while the implemented Util.exp function took 451 ms. The error was 0.0003% when subtracting Util.eps(2) from Math.eps(2). Rather than calculating the permutation of n in each loop, a lookup table with n! was implemented, but that did not result in further speedup, as Java spends time performing an array index check when looking up the values.

3.9 Summary of Single-Node SVM

The single-node SVM in constrained environments has been investigated in this chapter. First, we identified the relevant constraints for working under such conditions. Subsequently, the SVM was mapped to the constraints by relevant proxies. Some of the proxies were tested in experimental settings such as the possible savings by reuse of Lagrange multipliers. It was concluded that substantial CPU time can be saved by reusing Lagrange multipliers in the process of fine-tuning the parameters of the SVM. In a later chapter we specifically look at the energy consumption of the single-node SVM to understand better how the algorithm behaves on selected hardware platforms. Next, we can consider what happens when multiple SVM-based nodes are introduced into the system.

Chapter 4

Multi-Node Support Vector Machine

The main purpose of the DSVM system is to let the distributed nodes use mutual information to perform better as a group than on an individual basis. Therefore, this chapter relates to the second research question on how the SVM can be tailored to work well in distributed environments. Our goal with the distributed learning system is to distribute the information from individual nodes equipped with SVMs to the whole network of distributed, SVM-enabled nodes. The rationale is that distributed learning is possible and that we want to utilize key properties of the SVM algorithm such that goals related to minimal network communication are attained. This is linked to the implementation of distributed queries such as those discussed by Bonnet et al. [14]. One of the example queries in the paper (Query 3) is on fetching an *abnormal* temperature. To address such a goal, the DSVM framework would be a candidate as the distributed novelty detection in section 4.5 could be used for this purpose. In this chapter, we formalize the notation of the DSVM system, and introduce the key terms and definitions such that we can conduct and discuss selected experiments later in the chapter.

The DSVM is specified as a set of SVMs using a set of specialized kernels, partitioned over a set of vertices (nodes) connected by some set of directed edges. Each vertex and directed edge can be associated with a set of properties defining state, statistical characteristics, and computational limitations. Furthermore, the set of SVMs is associated with a set of states, roles and parameters describing its responsibilities within the DSVM. A relevant parallel can to compare with Tumer and Ghosh [91] as their Figure 6.1 show ”*Learner/Classifier 1 ... Learner/Classifier m*” and a ”*Meta-learner*”. In this framework the distributed learners are called a *producers* and the meta-learners are called *consumers*.

The three main challenges for the DSVM are: (1) The SVMs should only request additional information when there is sufficient reason for doing so (labelled the *communication-threshold challenge*), (2) mechanisms for exchanging optimal information-bearing data must be identified (labelled the *exchange-scheme challenge*), and (3) as limited information is exchanged, the generalization ability of the DSVM might not be fully known (labelled the *generalization-problem challenge*).

The SVM learning method provides mechanisms to address each of these challenges. The communication-threshold challenge could be addressed using ratios of support vectors vs. number of data points. Perhaps the richest set of tools is available to address the exchange-scheme challenge. We can opt to exchange information based on raw data, support vectors, generalized support vectors, convex hulls, or approximations of convex hulls to deliberately limit communication among the SVMs. We would expect that ensemble and re-sampling theory can be successfully used to quantify the generalization abilities of the DSVM. In short, there appears to be adequate theory and heuristics developed to provide a starting point with respect to each DSVM challenge.

The system has not been defined previously in a distributed formulation and, therefore, some flexibility exists when defining the objective for the system. The more distributed it is, the more data is also exchanged. For example, it is straight-forward to see that the compression ratio, CR, at each node is proportional to the amount of data that is not exchanged in the system. Therefore, high CR will result in high compression in the system. An example of a set-up in which this would not hold true is for a system of N nodes, each with minimal training sets of two points on each node. Provided the data set is not trivial, the two points would be included in the SV set and thus

Table 4.1: Exchange Schemes for Classification, Regression, and Clustering Problems

	DSVM Mode		Controlling Parameter		
	Voting	*Exchange*	Classification	Regression	Clustering
Compression scheme			SVM	ε/ν-SV	SVC
Support Vector	-	OK	C, ν	C, ε, ν	C, q
Kernel	OK	OK	Kernel family dependent		
Data redundancy	OK	OK	Problem dependent		

be candidates for immediate exchange with other consumer nodes. We use the producer/consumer terms in the rest of this thesis. A node that generates SVs is a producer, while a node that uses the SVs is a consumer. Nodes can be both consumers and producers at the same time. A node that collects data takes on the producer role, but another type of node can also come into play when data are travelling over long distances (as measured by a number of node jumps). As a consumer, the node would collect data from a number of producers while taking on a consumer role. Upon some trigger (time or data dependent), the node would train a new SVM classifier and produce a set of SVs that can be pushed toward the ultimate consumer within the DSVM system.

The DSVM exploits the fact that the SVM algorithm is rooted in the data examples. This makes the algorithm well suited for such distributed data mining problems that allow the data points themselves to be exchanged among the nodes in the system. The nodes in the system are thought to be small learning machines ready for more data such that the local optimization of the cost function can be fulfilled to maximum extent.

In previous sections, we introduced a cost function, which takes into account the communication cost in the system. We choose the number of exchanged data points, EP, as an appropriate proxy for the communication cost and minimization of same. In this section we want to motivate why the SVM algorithm could be an appropriate algorithm for intelligent exchange of data points in a distributed inference system. The three different compressions that take place in the DSVM system has been categorized in the table below.

The number of data points in SV is something that can be controlled by parameters set in the algorithm regardless if we are using the SVM for classification, regression or clustering. With respect to classification, the SV compression is dependent on C for soft margin SVMs as shown in equation (2.3). Naturally, this dependency on C vanishes if the data is linearly separable in feature space, but this is not a common phenomenon i practical machine learning as noise and other factors seems to have a constant presence. The ε-SVR [93] and the ν-SVR [80] are both regression SVMs that depend either directly or indirectly on the C parameter. In this framework compression/filtering is the key we take direct advantage of the parameter ν for controlling the number of support vectors lying outside the regression line tube. Finally, for the support vector cluster algorithm [6], we can use the same framework. The smoothness of the cluster boundaries are inversely correlated to the number of support vectors and bounded support vectors, which is an important property for using the SVC algorithms in a distributed environment. More precisely, the two parameters, which control the number of active points in the solution are the scale parameter q of the Gaussian kernel, and the well-known C parameter that controls the soft margin.

The DSVM is an implementation and a framework that takes advantage of the fact that the SVM is rooted in the data examples themselves. This makes the algorithm well suited for DDM problems that can be addressed using the fact that only the data points need be exchanged among the nodes in the system. The nodes in the system are thought of as small learning machines always requiring more interesting information such that the local optimization can be fulfilled. Each node wants new information in the form of training material that can better instruct about the true function it is trying to learn.

The notion of sets is central for the framework and Table 4.2 describes the various sets of objects that are of importance for defining the DSVM system.

We will refer to individual members of each set by using the usual indicators i, j, k, and l as indices. The most important set is the support vector set SV as it is here we find the initial compression or filtering mechanism. There is one SV set for each distributed learning machine, DLM_i and it will be the result of training an appropriate SVM on the dataset D_i belonging to node

Table 4.2: The Sets Composing the DSVM Framework

Role	Notation	Meaning
Configuration	N	Actual nodes (vertices) in the DSVM system.
	PN	Producer nodes (special vertices).
	CN	Consumer nodes (special vertices).
	L	Directed links (edges).
Learning	DLM	The set of distributed learning machines.
	IN	The set of inference nodes.
Data	GD	The global set of data points.
	D	The local set of data points.
	SV	The set of support vectors.
	CH	The convex hull vectors.
	EP	The set of exchanged points.

N_i. The nodes N are connected by a digraph with the link set L. A particular link L_{jk} can have certain properties such as not allowing a local dataset D_i to traverse the link and this particular property will in some instances make it necessary to run the DLM in *voting* mode. Depending on the mode, the CN set of consumer nodes will use the DLM differently. In *voting* mode the individual IN_i query the individual DLM_i to establish a voted answer. Alternatively, in *exchange* mode, the DML_i sends the SV_i data to DLM_j. Indeed, DLM_j will then use all of the SVs to create the CN_j model, which will be put to use on unseen data such as the test set. Quality of the exchanged SV data can be enhanced by performing a union of the SV and other schemes to identify the important points with relation to the final quality of the learning machines in IN. One example that would enhance the quality of exchanged data is to create $EP = SV \cup CH$, such that the convex hull would be exchanged as well. This enhanced exchange scheme applies to classification problems for which it is easy to see that it generally would make $D_{IN_j} = \cup_i EP_i$ more robust to local variations for the separating hyperplane at each classification DLM_i. All data points mentioned here are in the global set of data points for the DSVM system, such that $GD = \cup_i D_i$.

The DSVM system is characterized by the following elements.

Definition 1 *Define a directed graph $G_d(N, L)$ for which the edges (connections) are associated with a direction. Therefore, L is the set of edges defining the connection level in the DSVM.*

The role of this object is to define the existence of the set of nodes N and connecting edges L. We will enrich this with a connection matrix A for G_d that captures various properties of a DSVM system.

Definition 2 *Let there be defined a connection matrix A. The rows in some A, $A_{ij}, i = 1, 2, ..., n$ are nodes in G_d as are the columns of $A_{ij}, j = 1, 2, ..., n$. An entry A_{ij} defines the existence of a connection between two nodes by the following rules:*

$a_{ij} = 1$ *if an edge allows traffic from node i to node j.*

$a_{ij} = 0$ *if a connection from node i to j is not allowed or does not exist.*

Definition 3 *For G_d let there be defined a set of property lists K (not to be confused with the kernel symbol k) further defining the constraint properties of the set of nodes N.*

Definition 4 *Define a set SVM of support vector machines, $SVM_i \in SVM$.*

Definition 5 *Each $SVM_i \in SVM$ is associated with one node, $N_i \in N$.*

The definitions formalize the vocabulary for working with the DSVM system properties.

A directed graph G_d consists of a non-empty set of nodes N and a set of connections L between the nodes. Each connection is an ordered pair of nodes in N. Each node has processing capability P associated with it that can be measured MFLOPS. Depending of the complexity of the distributed system, each P in graph G_d can be viewed as a random variable with an associated probability distribution for how the processing capacity on each node varies with time. The same argumentation can be applied to the directed connections L_{jk} in the system. Furthermore, a cost CP could be associated with the processing capability P and the bandwidth B for each connection in L.

The basis of a distributed system is there is different information at different nodes. A conceptual global data matrix GD is introduced to ease notation. GD is normalized in the sense that no observations (rows) are replicated over nodes in N, and the same is also true for the features (columns).

4.1 Producer Nodes and Consumer Nodes

The nodes in a DSVM system can be divided into two categories: one that receives raw data and creates a model on those data, and one that receives a subset of the data, which we call the exchange set. It is possible for a node to fulfill both roles simultaneously.

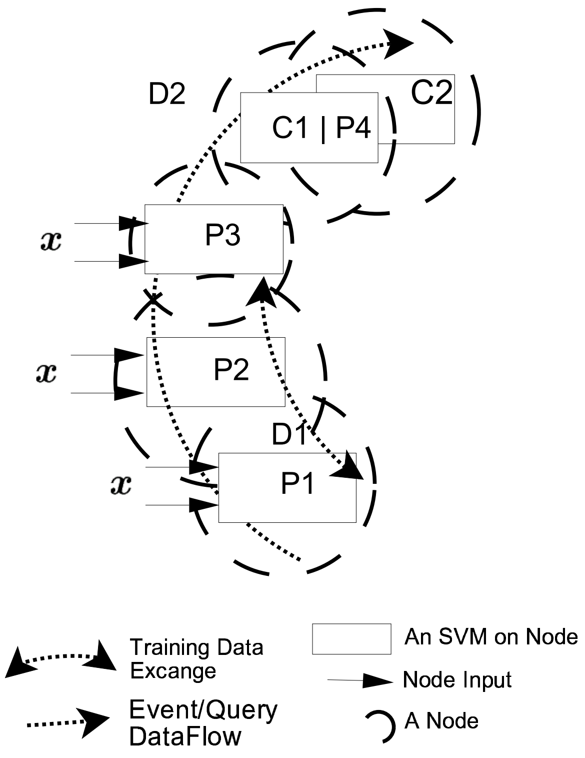

Figure 4.1: Producer and consumer nodes example. The boxes symbolizes sensor nodes, the dashed circles symbolizes radio perimeter reach, and the solid arrows are sensor data such as light and temperature. This example setup shows three nodes that receive a two-dimensional input $\boldsymbol{x}$. The input $\boldsymbol{x}$ is processed on the nodes and a data exchange among the producer nodes P1, P2, and P3 takes place. This data exchange could be the support vectors if the SVM is in training mode: D1 depicts this data exchange. A second data exchange, D2, moves selected data from P1, P2, and P3 to C1, which subsequently acts as a producer node, P4, to the final consumer node C2.

The producer nodes, $PN \subset N$, act as an information filter that transmits the relevant information along the connected edges, L, toward the network of consumer nodes. A producer node, PN_i, acts as a filter mechanism to minimize edge traffic. Its counterpart, the consumer node, CN_j, has a dual role. First, it must receive the points from the producer nodes and train its own model on those points. This is not unlike the *JAM: Java Agents for Meta-Learning* [87] project that uses the terms *local learning agent* for the producer node and *local meta-learning agent* for the consumer node. Additionally, it must act as the unit for testing new, unseen points. Let us look at the first possible state of the consumer node: the receiving state. The points from the different producer nodes arrive at the consumer node, which stores them in the training set for its local model. Each point arrives in association with its own Lagrange multiplier [82] based in the SVM model off the producer node. If the exchange scheme is to send the support vector SV set of data, then the consumer node would get only the points that belong to the support vector set. There is no guarantee that those Lagrange multipliers make sense for the consumer node on the merged support vector set. Producer nodes would most likely have trained on different subsets of the data such that the SVs would only make precise sense in the local part of the problem. One experiment of particular interest would be to test whether the consumer node can save on kernel evaluations using the given Lagrange multipliers or discard them and begin training on a nulled set of Lagrange multipliers. The KKT conditions are probably violated for the union of SV sets received from the producer nodes. In the likely situation that the SV subsets off the producer nodes are different, producer nodes are also trained to save on kernel evaluations because they do not necessarily have to send precise Lagrange multipliers along with the SVs. This is discussed in detail in a later section. It is generally only important that the exchanged SV set match the SV set that the producer node would have created using a condition of possibly satisfying the KKT (see (2.3) and (2.4)) conditions within the common 10^{-3} accuracy.

In this section various structural configurations for the DSVM system are discussed. System nomenclature was defined: nodes, edges, loops, producer nodes, and consumer nodes. The nodes are small centers of intelligence, which usually gather data, understand data, send data, or re-send data. There can also be loops in the structure. Maximum flexibility is achieved by depicting the DSVM system as a directed graph. Eulers theorem about graphs is interesting at this point as it provide insight into the DSVM node/edge/loop structures. The formula states that $N + L - Loops = 1$. This basic toolbox is needed to describe general layouts or structures for the DSVM system. Some of these layouts are well known from graph theory, and they will be revisited here. They cover star, sequential, time changing, mobile, and hierarchical configurations. The star configuration would be the most likely configuration as it pertains to a number of observer and producer nodes at the leaf nodes of the DSVM configuration. They would report back to the central star in the configuration, which could be a consumer node that was monitored by human personnel. The aspect-oriented classification in which each node takes a different angular view on the same object could be an example of a star configuration. A different view to consider is a sequential layout in which the producer nodes are along a line with a final consumer node as the main point of interest. An example of an application using this scenario could be vehicle classification as the configuration opens up for using image, sound, and spatial/time information in the final classification. A further configuration of novel interest could be to detach the producers and consumers from the physical node placement such that they could follow the problem they monitor in the DSVM system. This approach would be similar to object mobility in the Emerald system [55] if we allow the models and data to follow a problem around in a network. The system take on characterizations of an agent-based system. In an imaginary example we could consider the DSVM system as attached to one person and the data and models for this person follows him, such that it is available when his mobile device is connected to a wireless access point in an airport or shopping mall. There are also situations in which a hierarchical DSVM structure might prove optimal, such as when there is value to be gained by filtering information as it nears the final consumer node. We can see this as a star configuration with nodes that meet on their way to the consumer node. We should envision the typical tree structure graph. If we imagined the DSVM spanning the Internet, then the local producer nodes could send support vectors to intermediary consumer nodes, which then could resend the refined information to the final consumer nodes if that was the configuration of this scenario. Many of the above configurations have dealt with a consumer node that has been a set to be master node.

The problems addressed have clearly been centrally distributed, but the model has centrally set up. It would be equally possible to think of multiple consumer nodes that work within the DSVM system. This redundancy could be used to ensure higher availability or it could alternatively be used to extract different information from the system based on the same fundamental inputs. Therefore, the system is equally well suited for distributed problems and distributed data problems. The prototype is available for different types of nodes that can have various system characteristics. For example, it might be better to have an observer node pass the observed data directly to a producer node if it did not have the memory to hold the entire local data set in memory. In the hierarchical problem above, the intermediary nodes can be used to process information such that the consumer node would be presented with distilled information and thus did not have to be a more powerful node.

A main issue in the DSVM system is how well it would scale up to a large number of peers. It could be claimed that any distributed system needs to scale up gracefully to continue working on a satisfactory level. The scalability can come from two categories in the DSVM system: System related, such as the networking scalability; and model related, which results from using the SVM as the base algorithm. System-related issues could occur at two phases in the DSVM lifecycle. Initially the nodes will need to join a DSVM system, and each needs to obtain knowledge of the necessary peers. The name service peer will let each node contact it and at the same time return knowledge about other peers to the calling peers. Therefore, there are 10 nodes in the system, the first node would contact the name peer and, as the first node in the system, would not receive any references to other peers. Node two then joins by calling the name server and leaving its address and retrieves the address to node one. Node two then is able to work in a distributed fashion by calling node one to tell it about its existence. The choice here was between only letting the name sever distribute the node knowledge or letting the nodes distribute the address knowledge autonomously. The latter scheme reduces the number of calls to the name server by a high factor. If we look at the 10-node example before, then the tenth node will—just like the previous nine nodes—call the name server; register itself and retrieve the nine references to the other peer nodes in the system. It would then make one call to each of the nine peers to perform the local registration. The advantage of this autonomous registration system is twofold. It does not put a particularly high load on the name server, which would only receive 10 requests in the previous example, and it tests the connections among the peers such that each peer only has a valid list of requests when they have confirmed the connection. A graph of connected services is then formed in each peer that identified the nodes. This is similar to the JXTA rendezvous service. The name servers work among themselves to distribute references at periodic intervals. It would be typical to use one name server for each DSVM subsystem. The parallel to the JXTA system is the relay peer. Network traffic is one potential bottleneck in a distributed system and at this point the name and discover service has been addressed. Therefore, focus can be turned to other sources of scalability in the DSVM system. The SVM algorithm provides great scalability, which has also been analyzed in JMLR in the article *On sparseness of the support vector machine* [86]. Another indication of sparseness is the paper by Chris Burges, which uses the fact that some support vectors can be combined. This sparseness issue is also interesting in terms of the virtual support vector method, which trains the SVM and then perturbs the original SVs to produce a more robust representation of the data points. Bernhard Schölkopf and Alex Smola report having the best benchmark on the MINST data base using this method [82]. It is particularly interesting because it allows us to generate a larger dataset on the node but only to with the rather small SV set. How much data is exchanged among the nodes is quite problem dependent. For a hard margin problem, it can be little data, while a soft margin problem with large C would result in a higher ratio of SVs to be exchanged. In the regression case the linear problems can lead to small SV sets while non-linearity would result in more support vectors. In this scenario network traffic is controlled by the width of the discriminative tube that surrounds the regression function. If the tube width is set to zero, the data set would enter the support vector set and thus ultimately go out on the network. In essence, the tube width and the regularization parameter C provide the algorithm-dependent scalability in the DSVM system. For cluster problems parameter C to some extent controls the number of clusters.

4.2 Distributed Regression

The DSVM works for regression as well as for the other two areas that are the focus in this thesis. Regression problems are generally the fitting of a function to a real output number. The regression problem is usually cast to the fitting of a linear function to a set of targets that are associated with some inputs. The most popular regression algorithm is the ordinary least squares (OLS), which minimizes squared distance to the fitted regression line. The SVM algorithm was not directly applicable to regression, as it was originally designed for classification problems. Vapnik recast the problem to fit a tube to the data. This tube consists of the support vector set: points that are more than a distance ε from the regression line. So this makes it possible to create a regression function that essentially consists of a low number of points, as was the case for the classification problem. If the data is noisy or highly non-linear, the set of support vectors for the regression line will be enlarged. The support vector approach has an potential advantage over OLS and an MLP NN. In the first case, each node N_i would create the regression line, which would consist of a weight vector and a bias. In that sense the OLS is efficient if only those two pieces of information were exchanged. The question remains as to how the consumer nodes interpret the weight vector **w** and the bias b entering from the nodes in the DSVM system. The straightforward approach would be to take a simple average of the weight vector and biases to create the final inference model on the node. A more sophisticated approach could be to weight the average of the weight vectors and the biases according to the number of observations that were present on each producer node. Again the focus is on heuristics and not on the principled approach that the SVM framework offers. Another limitation of the OLS is that the algorithm is linear by nature. Therefore the SVM approach offers some advantages to this approach because it allows for a nonlinear fit of the data. Furthermore, the modifications to the SVM regression made by Schölkopf [82] make it possible to control what fraction of the vectors that end up in the support vector set. This is a flexibility only offered by theoretical satisfaction for problems that run on a single PC in the single model mode that is dominant for many reasons that shall not be discussed now. For the distributed problem in which the communication, or number of (support) vectors are something that we would ultimately like to control, the ability to reduce the number of support vectors in the exchange between the producers and the consumers is needed. In the case of the OLS, the amount is fixed and low, which is good in terms of the communication cost. However, as mentioned above, the algorithm is linear. The MLP neural network algorithm can also be discussed in terms of the DSVM framework. This would be a difficult algorithm to work with in a distributed environment if re-training was a goal for each of the consumer nodes. The most straightforward approach would be to train the neural network on the individual nodes and combine the models on the consumer nodes. There are various ways to combine the models as discussed by Tumer and Ghosh on linearly combined neural classifiers [92]. An alternative approach could be to retrain the models with the data that reside on the consumer nodes, but that would ignore the data that was used for training the models on the other producer nodes. The retraining of the neural network would be a difficult and fragile task. Networks are usually trained using early stopping, and this makes it difficult to retrain the models once a certain network has been locked. A separate set is often kept aside for which the error is monitored throughout the training period. Furthermore, the method of crossfolding would perhaps create ten neural models on each node.

It is clear that the implementation should allow a node to identify the important points in terms of the model. The points on the margin for a classification problem are most important for the classification model. In terms of regression the problem is a little more subtle because the regression line will be more visually strong or have weak support on different regions of the line. Therefore, the points cannot be determined as important simply by their distance to the line. For example, is a point that is far from the line important just because the square distance deems it so? The answer is perhaps yes (or no), but the algorithm must have a mechanism for excluding points of this kind from the final model if it could be regarded as an outlier. We can review the ordinary least-square model as we recover this algorithm if the SVM regression problem uses a tube with width of $\varepsilon = 0$. If this is the case then the algorithm fully recovers the OLS method, but on the expense of the sparseness of the solution as N. Christianini and J. Shawe-Taylor explain in *An Introduction to Support Vector Machines* [26]. There are ways built into the SVM regression to perform outlier detection. First, the algorithm makes use of a tube of width ε to ensure that points that fall close to the regression line do not enter the final model by becoming part of the support vector set. In other words, those points that might have been on the line in an OLS problem would not be part of the support vector set for SVM regression points. Secondly, a number of points are excluded from the solution by the regularization parameter ν. The ability to exchange the points that are part of the support vector set is an important implementation consideration. Therefore, the implementation allows this set of points to travel from producers to consumers such as in Figure 4.2. It would be natural to enhance the quality of the points by tagging along the Lagrange multipliers, which came about as part of the training algorithm. Working strictly with a linear regression, we could create the weight vector and bias, but this approach presents the same challenge as a distributed OLS, namely, the points could not enter a new solution in their natural form as they can when dual representation is used. Ordering points based on their importance and having them enter the training algorithm immediately upon arrival on the consumer node would be desirable. This would allow for more efficient training should the consumer node be equipped with a powerful CPU while employing a slow communication link. Accordingly, the exchange of points must support an ordered sequence of the point exchange and the points must be able to enter the main loop immediately upon arrival, which could be done by implementing the training algorithm as a separate thread on the consumer node.

4.3 Distributed Clustering

The cluster segment the most difficult to work on as compared to regression and classification. Clusters can be circular or have more complex boundaries, which depend on the clustering algorithm that is used [34] [100]. What constitutes a good cluster can be quantified, and that also adds a level

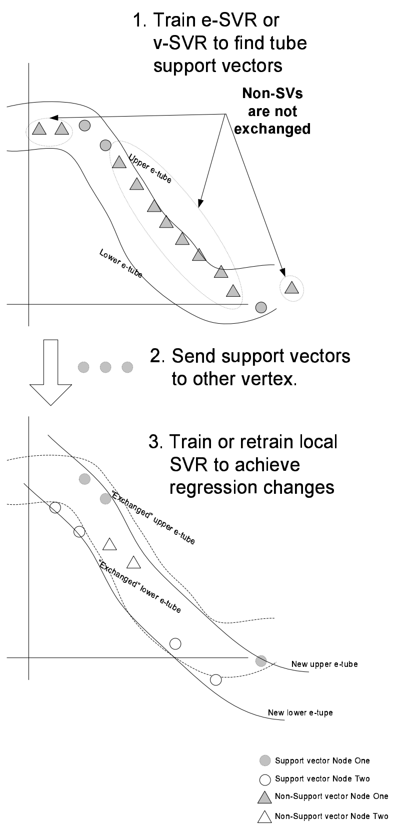
1. Train e-SVR or v-SVR to find tube support vectors
Non-SVs are not exchanged
Upper e-tube
Lower e-tube
2. Send support vectors to other vertex.
3. Train or retrain local SVR to achieve regression changes
"Exchanged" upper e-tube
"Exchanged" lower e-tube
New upper e-tube
New lower e-tupe
Support vector Node One
Support vector Node Two
Non-Support vector Node One
Non-Support vector Node Two

Figure 4.2: DSVM regression. The regression SVM on the producer node exchanges only three points. Subsequently, the SVR model takes on a new shape on the consumer node.

of subjectivity to this task. It is necessary to understand the clustering task both in a local (per node) and on a global level to know what points that are eligible for exchange. One property that is needed of the algorithm is that the identified cluster must not depend on the points that are not in the set of the support vectors or the bounded support vectors (BSV) [6].

Only those vectors are necessary to reconstruct the clusters to the original shape on the consumer nodes as they were on the producer nodes, which can be seen in Figure 4.3. Whether the cluster labels are exchanged is a choice that must be made. Network traffic is not so much the issue, since it would be an extreme situation if the extra traffic due to the cluster labels constituted a significant problem in terms of communication cost. The fundamental choice is whether cluster points are collected on the consumer node and then subject to retraining or if they enter the algorithm

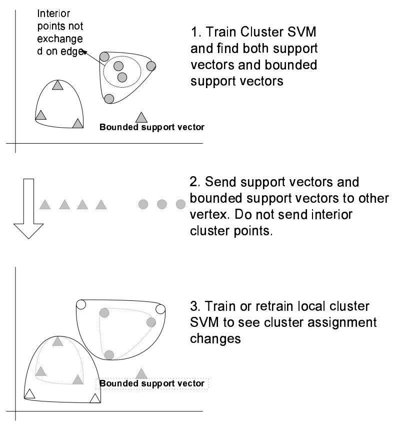
Interior points not exchange d on edge
1. Train Cluster SVM and find both support vectors and bounded support vectors
Bounded support vector
2. Send support vectors and bounded support vectors to other vertex. Do not send interior cluster points.
3. Train or retrain local cluster SVM to see cluster assignment changes
Bounded support vector

Figure 4.3: DSVM clustering. This example shows how the support vectors as well as the bounded support vectors of the producer node are exchanged to the consumer node. Next, the consumer node trains its support vector cluster model. The labels of the underlying classes are shown as circles and triangles to show the true classes but, these labels would generally be unknown in clustering problems.

with their local label as the main piece of information. That is one of the main issues that must be decided. A. Strehl and J. Ghosh propose hypergraph algorithms [88] that can used if the local labels are kept. On the other hand, the problems in the DSVM are supposed to be generated from the same underlying global distribution. Therefore, the arguments concerning different sets of input vectors are not applicable to this problem. This leads to the solution that the points should be exchanged without their labels and retrained on the consumer nodes. It should also be noted that the trivial solution is always present, which is just exchanging all the points, but the DSVM system was created for situations in which control of exchange sequence, edge traffic, storage and local producer node intelligence matters. The ability to identify points that almost became a support vector or a bounded support vector would be useful, as this additional information exchange would make the inference model at the consumer nodes more robust due to local differences in information. This difference reveals something about the robustness of the overall DSVM system. That is not particularly simple in the cluster setting and a proxy for quality could simply be the number of points that are in the approximated model vs. the number of points that are in the corresponding global model. The previously introduced usage of precision and recall would be appropriate measures here.

The question of how to determine with which points to augment the cluster points is still open. One approach would be to select points close to the support vectors and the bounded support vectors by using the kernel based distance metric $k(1,1) - 2k(1,2) + k(2,2)$ distance object. We want to do the opposite and reduce the number of points that are exchanged due to high communication cost. Consequently, points can be eliminated based on the size of the Lagrange multipliers, with the smallest multipliers being deleted first. Similar to the case of constructing the convex hulls, we can employ use of the kernel metric to identify points that are close to the clusters and thus are almost support vectors or bounded support vectors. This is easily done by working out the kernels for the remaining points and sorting them according to the closest points as prime candidates for augmented exchange.

4.4 Distributed Classification

The distributed classification task, which has been the subject of much research, is an interesting task for the same reasons that the stand-alone classification task is interesting. It is the application of how objects observed in a distributed system are classified. The basic problem is how a set of objects is classified into a set of classes. Each class represents a subject different from other classes. How classified patterns that are produced at a node are exchanged in the system to yield a distributed classification system will be examined. Again, the same basic principle persists: The main focus is the support vectors that result from training of the local SVM classification model. There are ways of augmenting this set with richer information such as the approximated convex hull measure. Envision two producer nodes of support vectors, each node having its own local classification problems. Accordingly, the DSVM system would need at least one representative from each class to train the local classifiers. The DSVM system would have four observations to work with. The consumer node—a third node—would combine the four points by retraining a model using the Lagrange multipliers that were passed to it from the two producer nodes. We, for the sake of the example, could imagine the points in class one near each other and the same for points in class two. Therefore, the resulting separating hyperplane is capable of separating all four points without error, and the shared separating hyperplane is not necessarily the same as found on each of the producer nodes. There are no easy ways of knowing this beforehand, but there are situations when there are enough CPU power or transmission capacity to augment the support set with the convex hulls or take the route of full exchange of all the points. The new problem arises for the DSVM when the points on the producer nodes are misclassified. That indicates one of two possible situations. Either the kernel running on the producer node is not capable of complete separation or the data points are inherently noisy. If this situation occurs, the producer can try performing kernel optimization and parameter adjustments using crossfolding and then retrain one consistent model. The problem of binary classification is it is a basic way of working with the classification setting. However, many problems are inherently multiclass problems, and the producer nodes should be able to solve these types of problems as well. The SVM works by producing a separating hyperplane that ensures maximum distance between the points. In the case of multiclass problems, a classifier for each class is trained and applied using the one-against-all approach [76]. It is intuitively clear there will be a strong overlap by the support vector points from this multiclass exercise. Before the producers could send the support vectors, a union of the support vector points would have to be created such that the points are not retransmitted a maximum of C times if C represents the number of classes. There are ways of associating the Lagrange information attached to each point during the exchange that takes place between the producers and consumers. In a straightforward manner, the consumer node retrains the local SVM models using the points that are exchanged between them such as displayed in Figure 4.4. At that time, the consumer node will have its set of binary support vector sets from each final binary classifier, the extra points that did not make it into the final classifier set can be retrained.

4.5 Distributed Single-Class Classification and Novelty Detection

Most distributed nodes of the sensor type gathers unlabelled data, such as temperature, acceleration and light intensity. If SVMs can be used to detect when one of these data streams is deviating from normal levels, it would be one more example of applying SVMs to distributed learning problems. For example, we can suppose (see Table (4.2)) that a set N in a DSVM system consists of sets of producer nodes PN and consumer nodes CN. Three things are needed to limit data transmission in the DSVM system: (1) an appropriate filter algorithm placed on the PNs, (2) a method for sharing knowledge among the PNs, which does not involve full data dissimilation, and (3) an opportunity to let the PNs make autonomous decisions. The base algorithm that is considered here was published by Schölkopf et al. [81]. However, their approach did not consider the distributed learning perspective. For the quadratic program please refer to [81]. The function that is used takes the value +1 for most of the examples, while a small fraction ν is labelled -1. This is done by the following function:

$$f(\boldsymbol{x}) = sgn\left(\sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) - \rho\right) \tag{4.1}$$

The dual problem that was solved to get $\boldsymbol{\alpha}$, see (4.1)is presented here:

$$\underset{\boldsymbol{\alpha}}{min}\frac{1}{2}\sum_{ij}^{l}\alpha_i\alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \text{ subject to } 0 \le \alpha_i \le \frac{1}{\nu l},\ \sum_i \alpha_i = 1. \tag{4.2}$$

Two things that make (4.2) this interesting: the parameter ν defines both an upper bound on the fraction of outliers and a lower bound on the fraction of SVs. It is possible to use the fraction of outliers to estimate radio communication of novel points once the system is in running state and new data arrives on the node. We can use the lower boundary on the fraction of SVs to estimate the amount of memory the algorithm will require when trained. Distribution of this model in the DSVM system requires only that the data with Lagrange multipliers $\alpha_i > 0$ are exchanged, and the advantages of lower communication bandwidth over exchanging raw data would have been achieved. It is similar to distributed regression, classification, and clustering. In addition, this method for extra exchanges of data based on pairwise distances in feature space can be used. This concept is formally introduced in section 4.8.

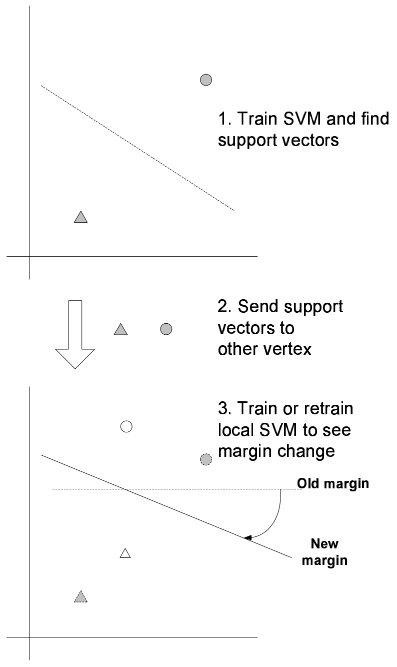
1. Train SVM and find support vectors
2. Send support vectors to other vertex
3. Train or retrain local SVM to see margin change
Old margin
New margin

Figure 4.4: DSVM classification. A producer node sends the support vectors to the consumer node. The SVM at the consumer node ends up using one point from each node.

4.6 Data versus Model Exchange Schemes

The distributed inference system exchanges information among the system nodes. There are various ways to execute this exchange. A naive but straightforward way is to replicate all data between all nodes such that each node has the full data set available prior to the training of the model. Alternatively, the inference models can be exchanged among the nodes and allowed them to use the data that is available on each node. One aspect that differentiates the SVM from a MLP NN is that the SVM can exchange the data associated with one scalar, which is the Lagrange multiplier α

Exchange of Data: It is possible to just exchange the data among the nodes. If all data is exchanged, the problem is the equivalent of having each node equipped with a standalone inference

Table 4.3: Exchange Scheme Analysis

Exchange scheme	Properties	
	Advantages	Disadvantages
Only data	Simple	Possibly high communication load
Only models	Security	Specialized

problem. It is more relevant to investigate when data is not fully exchanged between the nodes. This is a gradual from having one point $\boldsymbol{x}$ on each node N to the case in which all points are represented on all nodes.

Exchange of Models: The other approach is to exchange the models among the nodes and let each each model M be updated when it encounters new data on each node. Models can be moved around the nodes systematically or randomly.

- Systematic movement of the model
- Random movement of the model

Systematic movement of the model is implemented by ordering the model to move from node to node in an ordered fashion. For some model algorithms this could be as simple as moving from each node using a list of the nodes. If the model only needs to visit the nodes once, this is a predictable way of sending the model around to the nodes.

The random movement scheme can be used in cases in which the model may need to visit the nodes more than once and is not particularly sensitive to the order in which the data is received and used for training.

There are advantages and disadvantages to each of the exchange schemes, which are listed in Table 4.3.

The are two types of information exchange between the DLM and the IN sets that we investigate here (see Table 4.2 for the notation in use), which apply whether we are solving a distributed classification, regression, or clustering problem. The DLM set can, in principle, be used in two ways upon training of its associated set of SVM models. Each DLM_i in the DSVM system

Table 4.4: Combinations of Exchange Schemes

Mode[a]	Classification	Clustering	Regression
Exchange data	+SVs +/-Convex hulls +/-Simplified SVs -Zero α points	+SVs +/-Bounded SVs -Interior cluster points	+SVs -Zero α points
Exchange Votes	Majority vote	Hypergraph partitioning	Average

[a]Note: We denote points that are exchanged by a plus sign, $+$, and points that remain on the nodes, N with a minus sign, -. The *voting* scheme does not involve exchange of training data. Using the + and - sign indicates a design choice that is determined by a specific implementation domain.

thus has a trained SVM, and the well-known SV set of support vectors. The choice to be made now is if the inference SVM located at each IN_i in the IN set, should be able to receive the SV set from each DLM_i in the DLM or if the original data cannot be moved around within the DSVM network, which would naturally lead to the need for using the DLM set as small oracles and query them for answers based on a certain test observations. We will call these two distinct modes of operation for *exchange* and *voting*, see Table 4.4. In *exchange* mode the SV set is pushed toward the IN set, in which final training of the SVM on in_i takes place upon receipt of the SV sets from the DLM set. On the contrary we have the *voting* mode, or query mode, in which the IN set makes queries to each DLM_i in DLM. The following table summarizes the method: We perform a series of experiments using two parameterized probability distributions. We have chosen the uniform distribution and the standard normal distribution to generate data sets. The setup we will use is one in which two producer nodes PN_1 and PN_2 have been created, each with a local data set D_1 and D_2. To create an inference model on the data, a consumer node CN_1 is introduced. First, experiments will show the concept of the convex hull in one producer node. Then a series of experiments will show how the different exchange schemes introduced earlier work when used on minimal data sets.

4.7 Distributed Classification Exchange Scheme Experiments

The most straightforward way of exchanging points is to send all points from all producer nodes PN to the all of the consumer nodes CN in the system. That way all inference models would have the same data from which to work, which would be the global conceptual data matrix GD. There are also situations in which communication costs, communication capacity, or sheer data volume on one or more producer nodes in the system make it optimal to limit the exchange to some consumer nodes to points that are deemed important according to an exchange scheme.

Separable: Cases in which the global data set is separable and exchange of the full set of points will, as noted above, result in a classifier as good as if the system were a stand-alone system.

Non-separable: Cases in which the performance of the distributed system would be the same as the stand-alone system even when the data in a global sense is non-separable because all data is exchanged to all consumer nodes, the union of all the points render the same data set as in the conceptual global data set.

Issues addressed in the previous sections have been somewhat abstract, and there are many ways the distributed inference system could be configured. The focus on the conceptual level in the experimental section is basing the distributed system on exchange of only important points. We identified a setup with two producer nodes and one consumer node as the simplest configuration to illustrate the concepts we have noted. Convex hulls exist for input spaces of dimensionality two and up. Therefore, we have set the system to work with two-dimensional input data.

If each producer node has a local data set D_i, and processing capacity to execute code on that node, then some algorithm designed to identify the points on the convex hull can be used on each producer node to find the convex hull of each class C_i.

Separable: The identification and exchange of the convex hulls will be optimal in the sense that all classes on all producer nodes must also be as separable as the conceptual global data matrix is. When all convex hull points from all producer nodes are sent to one consumer node, the distrib-

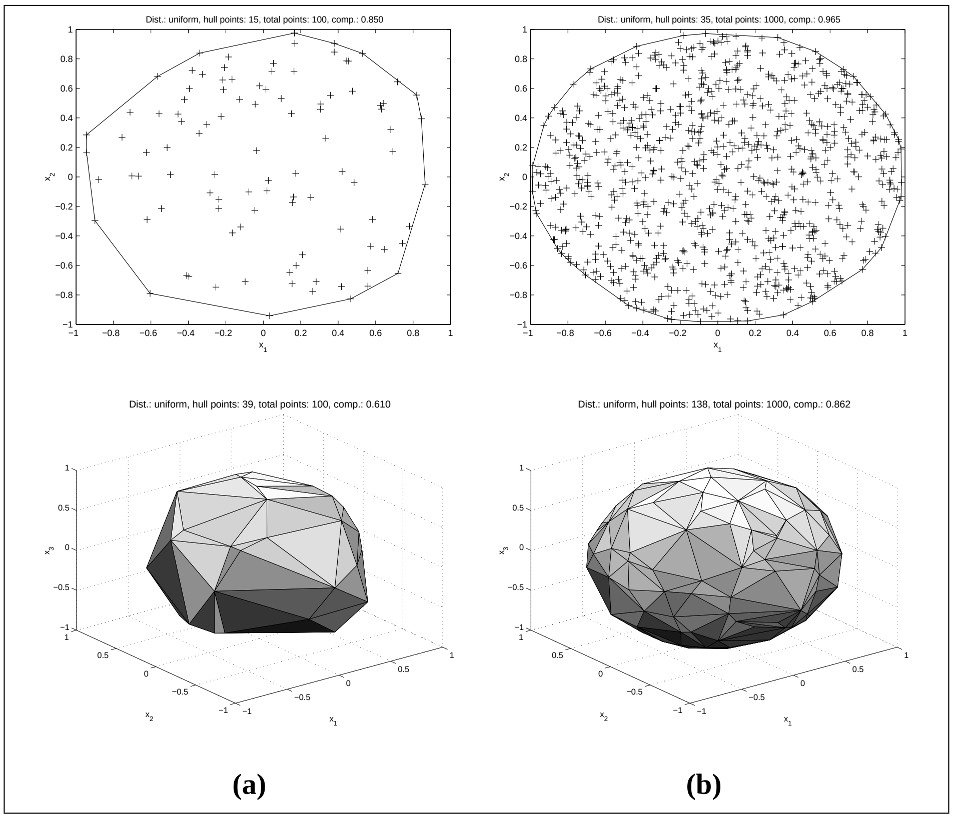

Figure 4.5: Demonstration of four different local data matrices. Two were generated from a uniform distribution that was restricted to the unit circle. The Two unit spheres were generated from a three-dimensional unit distribution.

uted model based on support vector classification, would be the same as if the full conceptual data set had been used.

Experiments show that the complexity of the Matlab `qhull` function is quite high and takes a prohibitively long time when the dimension of the input space reaches about dimension 10. Therefore, we need to establish a surrogate for the concept behind the convex hulls that uses the kernels to find the convex hull. One approach is to sort the points for each class according to the kernel output. The first two points with the highest kernel output is on the convex hull in the feature space. The third point we are not certain about. The reason is that it could be lying on the line between the two other points. It gives us the option to have some sort of mechanism likely to

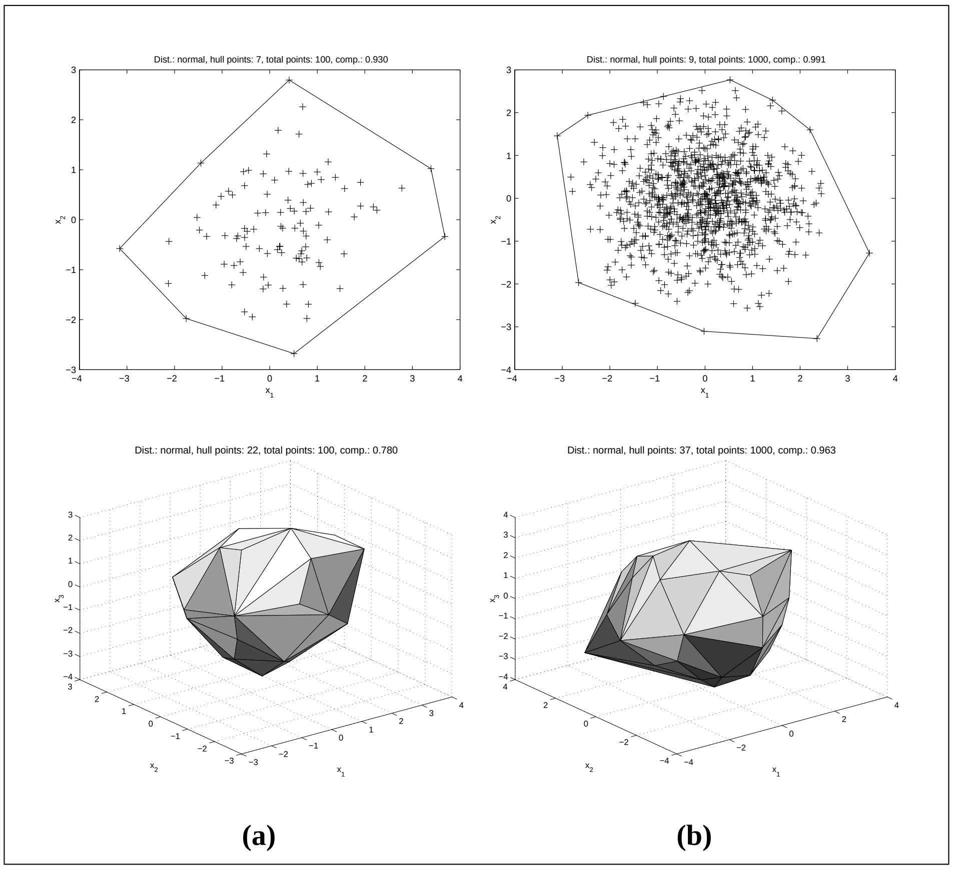

Figure 4.6: Standard Gaussian point generator. The data points for the four examples above data matrices D have been generated using a standard Gaussian.

work as a convex hull. The algorithm would start out with the two points with the highest kernel output and then use them as the seed for the next pair of points. It can either be the first points that are part of the convex hull or other points. The algorithm requires that the full Gram matrix is generated, but that should not be difficult as this algorithm could run virtually transparently along the SMO algorithm and keep a list of convex hull candidates on the list. The easiest approach could be this: For each point store the maximal kernel output of some other point. When the time comes to exchange the support vector set, a given fraction of the points with the highest kernel outputs are exchanged as well. We know from the first DSVM paper that the number of points on a convex hull

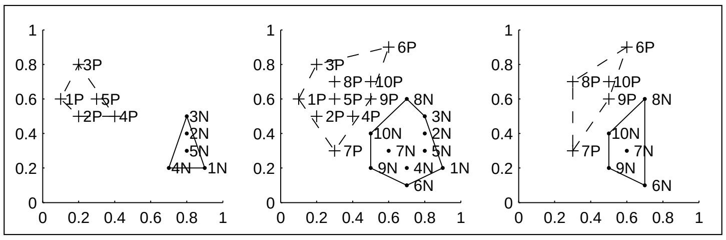

Figure 4.7: Separable-full exchange. The conceptual global data matrix GD is split onto two local data matrices D_1 and D_2. D_1 is to the left, D_2 is to the right, while the single consumer node PN_1 is in the middle. All points in D_1 and D_2 has been exchanged to the producer node PN_1.

is surprisingly few for the popular Gaussian and uniform distributions. Therefore, the percentage of pseudo convex hull points probably does not need to be high. A minimum would probably be the top 1 percent going up to about 10 percent. Future experiments will determine how well this scheme overlaps with Matlab's qhull scheme.

Non-separable: It is not so clear what happens when the global data set is not separable, because the producer nodes can have separable and non-separable classes, depending on the exact blend of data points on that node.

This exchange scheme is more a combination of using either the convex hull exchange scheme or the support vector scheme to identify and exchange the points. It is more robust than either of the other schemes used on a stand-alone basis.

Separable: This is the simplest case as both selection methods complement one-another. Important points will be identified using this scheme. If the two identification efforts are run separately, we could expect a high computation cost. However, it is possible the two algorithms can be combined to be used efficiently in this DSVM scheme.

Non-separable: It is possible to have local data sets in which support vector identification does not catch all points of importance for the consumer node, which is why the union of the support vector points and the convex hull points can be an improvement.

There is full flexibility in choosing an appropriate exchange schemes to run on each producer

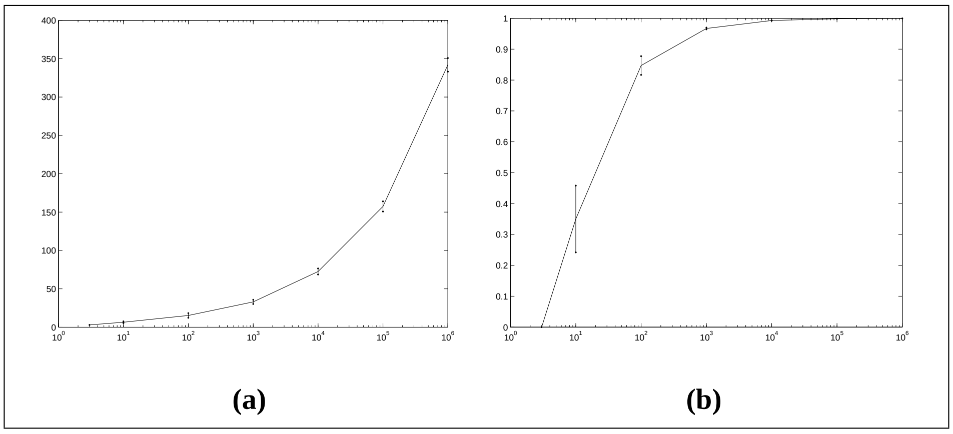

Figure 4.8: Compression ratios of convex hull for uniform distribution. Figure (a) show how many points that ended up on the convex hull as compared to how many points were distributed uniformly. Figure (b) show the compression ratio - i.e. the ratio of points on the convex hull divided by the total number of points. The observations on both figures have standard errors associated with them. It is computed by running each experiment ten times.

node.

Dimensionality puts a limiting factor on the usability of the convex hull approximation scheme.

Table 4.5: Convex Hull Points for Rising Dimensionality for 10,000 Points

Dimension	Uniform[a] distribution	Normal distribution
2	75.50 +/-2.55	13.60 +/-2.32
3	443.60 +/-12.61	50.70 +/-3.62
4	1289.80 +/-19.85	151.60 +/-9.24
5	2610.70 +/-30.10	336.70 +/-11.08
6	4139.00 +/-50.06	674.70 +/-21.96

[a]Number of points on convex hull.

Distribution is integral to determining the effectiveness of the convex hull exchange scheme. Table 4.6 illustrates the normal distribution rather than the uniform distribution was more suited for the convex hull exchange scheme. The explanation can be visualized by referring to Figure 4.5 and

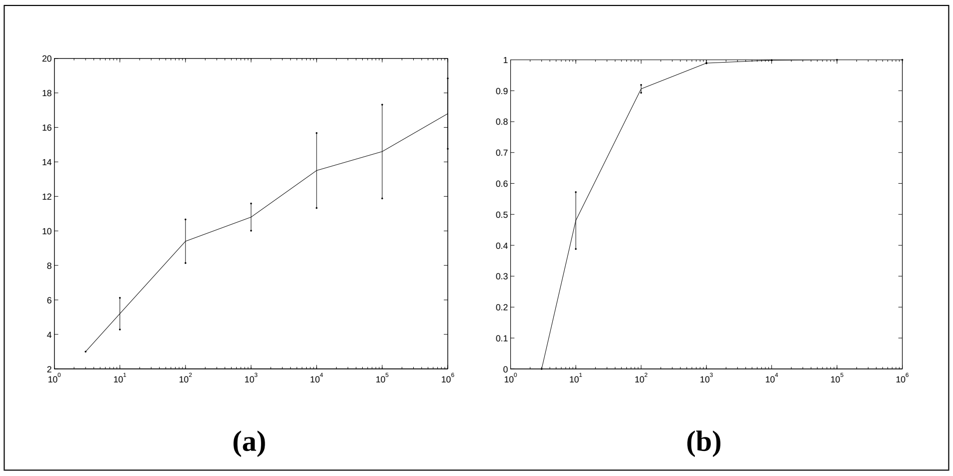

Figure 4.9: Compression ratios of convex hull for Gaussian distribution. Figure (a) shows how many points ended up on the convex hull as compared to how many points were distributed using a two-dimensional normal Gaussian and the covariance being the identity matrix. Figure (b) shows the compression ratio - i.e. the ratio of points on the convex hull divided by the total number of points. The observations in both figures have standard errors associated with them. It is computed by running each experiment 10 times.

Figure 4.6, as it is evident that the rare points of the normal distribution spans a more rugged convex hull than is the case for the uniform distribution.

The uniform compression ratio declines rapidly as shown in Table 4.6. It could be a potential weakness of the convex hull exchange scheme, but its effectiveness would have to be judged in a real situation when communication costs and other factors were considered as well as well. This would amount to a traditional constrained optimization problem.

As for the normal distribution, the compression ratio decreases at a more moderate rate than for the uniform distribution. In Table 4.6 we can see that even for six-dimensional data the compression ratio remains in the 90% range, which is very good.

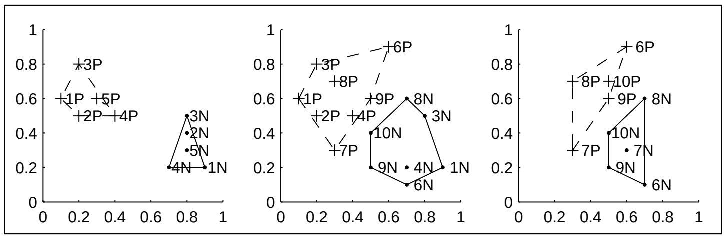
Figure 4.10: Separable-convex hull exchange. The points on the convex hull at each producer node have been identified and exchanged to the consumer node. This eliminates points $5P$, $2N$, and $5N$ from reaching the consumer node CN_1 if we just examine producer node PN_1 on the left. There are points from producer node PN_2 that also did not reach CN_1.

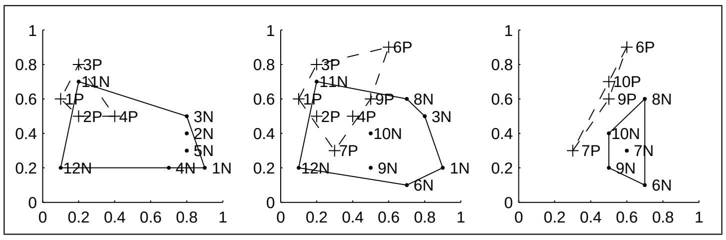
Figure 4.11: Nonseparable-convex hull exchange. The points on producer node PN_1 (left) are now non-separable. The points falling on the convex hulls on PN_1 (left)and PN_2 (right) have been exchanged to the consumer node.

4.8 η and δ Exchange for Classification, Regression and Clustering

A more general exchange scheme exists than the one discussed in the previous section. This new exchange scheme is rooted in the idea that the SVs are the most important datapoints for an SVM. Rather than identifying the convex hull of a classification problem, consider a method that works with SVs also for regression and clustering. The proposed method works for problems associated with a kernel-induced feature space with positive definite kernels. The method introduced in this section is that data points in feature space, $\Phi(x)$, which are close to the SVs of a trained SVM, are exchanged to other nodes. The logic is that these points are ”almost” SVs and should be prone to become an SV on another consumer node. If the exchange of only SVs (see Figure 4.12) is not

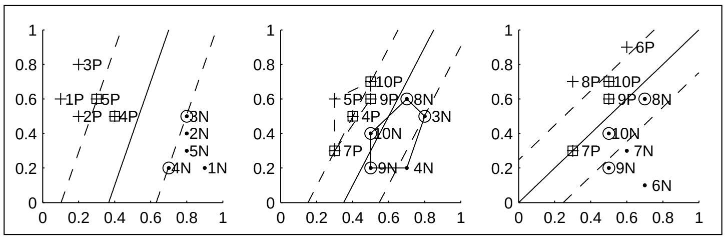
Figure 4.12: Separable-support vector exchange. The points serving as support vectors at each producer node have been exchanged to the consumer node. Note how points $1P$, $2P$, and $3P$ from class C_P were eliminated as important on PN_1 (left). From class C_N points $1N$, $2N$, and $5N$ were eliminated on PN_1 when using the SV exchange scheme.

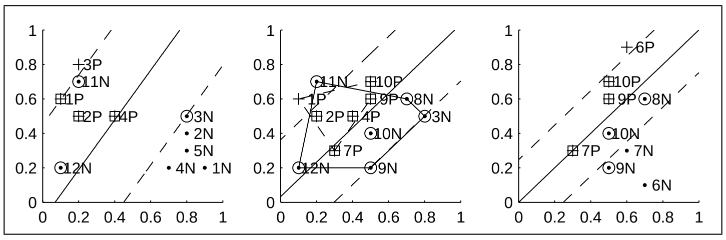
Figure 4.13: Nonseparable-support vector exchange. The points to be exchanged from the two producer nodes have been identified by the support vector exchange scheme. Notice how point $6P$ on D_2 (right) is ignored by this exchange scheme while it might have been an important point for the final model on the consumer node M_1 (middle).

enough, this η (see also (2.10) for η used in a different context) or δ scheme can be used. We use δ as a pre-defined percentage measure for how much data that is exchanged from a producer node. Before presenting the definitions, recall that the squared distance between two data points $\boldsymbol{x}, \boldsymbol{x}'$ in feature space is $\eta = ||\mathbf{\Phi}(x) - \mathbf{\Phi}(x')||^2 = k(x, x) + k(x', x') - 2k(x, x')$. The cases for regression and clustering are similar. In the clustering case, the bounded support vectors [6]are treated as belonging to SV. The following equations make use of the terms in Table 4.2:

$$EP_{\eta,t} = \{x \in D \setminus SV, x' \in SV : \bigcup \eta(x, x') \leq \ t\} \tag{4.3}$$

$$EP_{\eta,\delta} = \{x \in D \setminus SV, x' \in SV : \bigcup min_{\delta}(\eta(x, x'))\} \tag{4.4}$$

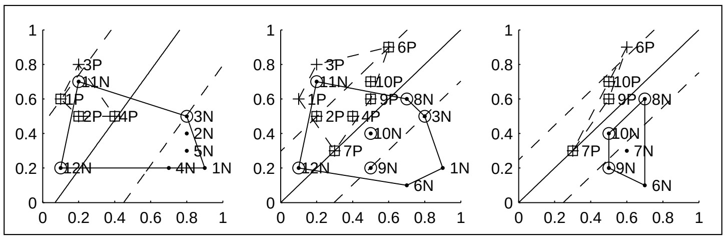

Figure 4.14: Nonseparable-convex hull and support vector exchange. The points on the convex hull as well as the support vectors are now exchanged to the consumer node. Point $6P$ from D_2 (right) fell on the convex hull of class C_P and was thus exchanged together with the union of the other convex hull points and SV points from C_P on D_2.

Table 4.6: Convex Hull Compression Ratios for Rising Dimensionality for 10,000 Points

Dimension	Compression[a] %	
	Uniform distribution	Normal distribution
2	0.992 +/-0.000	0.999 +/-0.000
3	0.956 +/-0.001	0.995 +/-0.000
4	0.871 +/-0.002	0.982 +/-0.001
5	0.739 +/-0.003	0.966 +/-0.001
6	0.586 +/-0.005	0.933 +/-0.002

[a]Example: A compression of 90% means that just 10% of the data is exchanged.

The first expression, (4.3), selects a complementary set of data points from the local data point set D by including points, within a certain maximum distance from any of the support vectors SV. The threshold parameter t that determines how many points are included in the supplementary exchange set $EP_{\eta,t}$. The second expression, (4.4), is designed to select a fixed number of the nearest data points for each support vector SV_i. Precisely, it is the parameter δ that determines how many data points are included. To avoid re-transmitting data points, both set specifications create the union, $\cup$, as an increasing overlap that will occur as δ and t increase. Optimal exchange schemes are domain specific, but we can note that (4.3) is an on-line algorithm as each $\boldsymbol{x}$ is tested fully upon calculation of $\eta(\boldsymbol{x}, \boldsymbol{x}')$. Due to the min operator, it is necessary to test all $\boldsymbol{x}$ in (4.4) before $EP_{\eta,\delta}$ can be determined. The on-line version, if time is an important constraint as each data point in $EP_{\eta,t}$, can

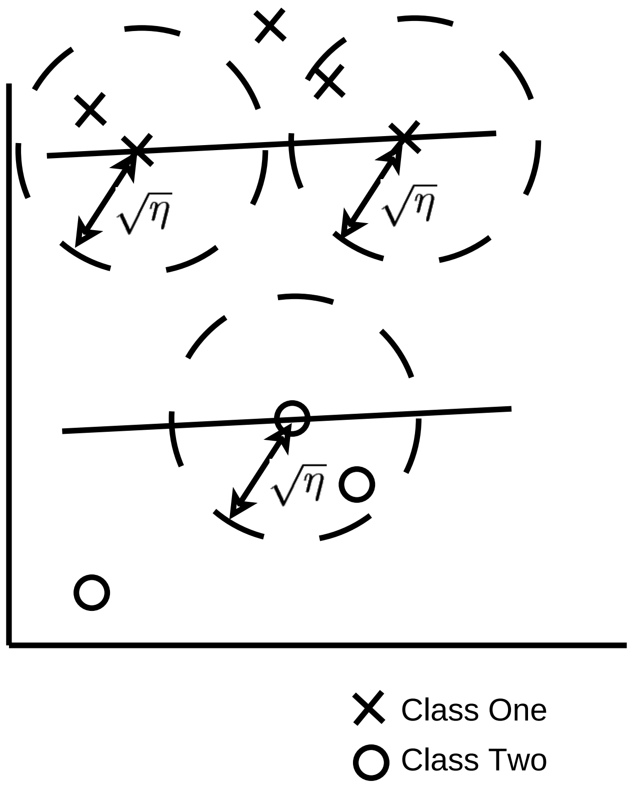

Figure 4.15: η exchange for classification. Data points within the perimeter of the circle are determined by the distance metric η in feature space included in the augmented exchange set of points. There is one point of each class that is not captured by this specific setting of a threshold.

be transmitted immediately. Should this not be the case, the version that exchanges an additional fixed number of points is better as it allows for tighter control of amount of data transmitted from the producer nodes to the consumer nodes. In the classification case, an additional sophistication could be incorporated by ensuring that prior class probabilities were enforced while selecting additional points for exchange.

4.9 Constraints Proxies in Multi-Node Setting

Various costs of the DSVM system are evaluated in this section, and proxies for the costs are constructed. The costs fall into two categories: system costs and machine learning costs. Within the system cost category, network costs, storage cost, CPU costs, and communication costs are examined. Communication costs are those that stem from exchanging a certain number of bytes over a network. This is particularly relevant for a resource constraint aware system such as the DSVM. The machine learning costs include the misclassification, regression, and clustering costs. In short, it can be labelled model quality cost. Finally, there can be a cost associated with time, which is the time lag between observing points at the nodes to the time it takes before the consolidated model is ready at the consumer nodes. We can call this *time cost*. The system-related costs and the model-related costs are discussed now, then are integrated using appropriate proxies rooted in the SVM algorithm. The two primary cost measures in the system area are network traffic and CPU power. Network traffic is the cost of the traffic that the DSVM nodes generate. This traffic falls into two subcategories. One is the obvious cost of exchanging large amounts of data, which need to be packaged in a network format and then transmitted to the end destination. The physical network topology can be quite complicated with lower-layer routing protocols ensuring final delivery of important data points to the consumer nodes. The other generator of network traffic is the DSVM structure information about, which nodes are up and running and which are not. This distributed picture of the DSVM system can be propagated in the network at various intervals. It is clear that instability of the distributed system on one hand and high accuracy in distributing the node information around in the DSVM both drive up network traffic. The topology information is not a cost-driver in terms of the amounts of data that is exchanged, but more in terms of the frequency at which the information is exchanged. This keep-alive information is expensive for distributed nodes that have limited uptime due to limited battery life. However, the actual exchange of data over the network is also expensive as distributed nodes have a non-trivial cost of sending and receiving data over the network. The other system cost is CPU power of training the models (producer and consumer nodes), producing the models (producer nodes), and gathering the data (producer nodes). It is possible to limit the CPU power consumption by creating training schemes that are less expensive to run than the traditional models. Similarly, it is possible to limit the cost of storage on the nodes by only retaining the points that are of high importance to the model. The storage, network traffic, and CPU power costs are interrelated in the DSVM system as the algorithm only depends on the subset of points that are of high importance to it, such as the points that constructs the weight vector in feature space, or the points that support the ϵ-tube in the regression case, or the support and boundary vectors of the clustering case. The cost of these three categories can be found by looking at some examples of nodes that would run in the DSVM system. Three examples of such nodes are the Motes by the Berkeley team, the mobile phone that runs the Symbian system, and traditional applet viewers that are connected to the Internet through a host computer. The first example is the mote unit, which communicates wirelessly over short distances. It is equipped with a battery, and the mote can shut down the CPU for periods of time to save energy. The phone is also equipped with a battery and is similar by that comparison. All three units have costs associated with storage, and they are quite different.

The DSVM system uses the sparse solutions produced by an SVM to reduce network traffic in distributed inference systems. Assessment of network traffic costs can be conducted by measuring how many data points flow among the nodes in the DSVM digraph. It is possible to measure the actual size in bytes, but we think a data point count would suffice to obtain a proxy for comparing different network traffic reducing approaches. However, in the chapter on energy awareness, we measure energy consumption per transmitted byte.

4.10 Using Information Retrieval to Understand Exchange Effectiveness

The precision and recall metrics—often used within information retrieval (IR) [4]—can be adapted to study what happens with the sets of support vectors, SV, that are used in the DSVM system. The two metrics are rooted in information retrieval. They serve as measurements of how well a set of documents match a query expressed by the user. There are a number of relevant documents in the collection and a number of irrelevant documents. With this collection, the retrieval machine will return a set of documents. The machine will probably return a mix of relevant and irrelevant documents. Therefore, two metrics have been constructed to measure how well the retrieval machine works. The first is called recall, R, and measures the ratio of how many relevant documents the machine retrieved. It is calculated as $R = \frac{\#relevant}{\#totalrelevant}$. Equally relevant is the precision metric, which is calculated as $P = \frac{\#relevant}{\#retrieved}$. As illustrated in the formulas, the two measures are inversely related. To capture this into a common measure, it has been suggested to work with the harmonic mean [83] of the two measures. It comes out like this:

$$HM = \frac{2RP}{R+P}. \tag{4.5}$$

There are several places in which IR measures can be used to analyze the behavior of the DSVM system. The producer node could be the starting point. It would be interesting to experiment on how well the exchanged SV sets match the SV set that would have been found if the consumer node was trained on a standalone basis using the full global data set. At the producer node, additional computations are performed to ensure the best data set is prepared for exchange.

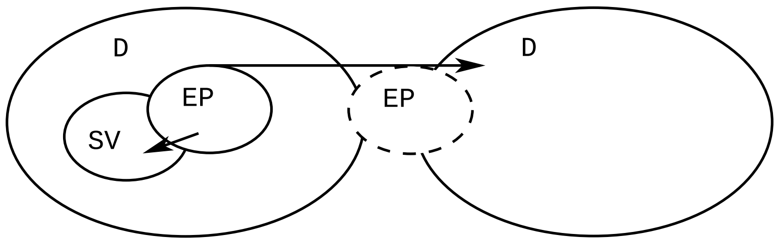

Figure 4.16: Exchange effectiveness using information retrieval. An SVM on the left producer node is being trained using local data D and at some point it will start exchanging data (the dashed set EP) to the consumer node on the right side. The dataset on the nodes are labelled D. The exchange set EP gradually becomes the set of support vectors SV during training. There are several ways, such as relaxed KKT conditions, that most likely results in a difference between EP and SV. It is this difference among the two sets that is captured by the harmonic mean HM .

The SVM is trained using relaxed KKT (see (2.3) and (2.4)) conditions to save CPU cycles. It is interesting to measure the IR metrics, (R, P, HM), profile of the support vector set as the training progresses. Though it is quite expensive, it would be an interesting piece of information to be able to review the relationships between R, P, HM, and the number of kernel evaluations $\#k$. A visual distinction could be made between inner loops on the working set and outer loops on the whole set of data. Furthermore, it would be worthwhile to collect data on the behavior of the convex hull approximations, as is discussed later in the section. There would be reason to believe that the smaller the node data set gets the more important the information from using the convex hull becomes. The reason is that the local SVMs will differ more and more from the global conceptual SVM, and therefore the convex hull could make the solution more robust by retrieving the points that would have been part of the solution if the separating hyperplane had been rotated around the convex hull. The same IR measures could be used for regression because the SVM algorithm also can be configured to produce sparse solutions. The role of C, as used in the classification setting, is somewhat replicated by the width of the $\varepsilon-$tube in the regression problem since the size of the SV set can be indirectly controlled by changing the width of the tube. The same experiment on monitoring precision and recall as a function of the kernel evaluations should be carried out in this setting. It would suffice to plot the harmonic mean of the precision and recall measures vs. the current kernel evaluations. It is likely that there are immediate applications to regression and classification problems. Cluster problems are perhaps more complicated. First, the per class approximation of the convex hull cannot be carried out due to the nonexistence of the class notion. It could be suspected that the local node clusters are sensitive to variations in the local data set.

4.11 Transductive Training of DSVM

Transductive training ideas have prevailed for some time in the SVM community, which state that some information from the unlabelled data set can be used to create a better model during the training phase of same. More specifically, the DSVM opens itself up to several possibilities for transductive training. The node level of the DSVM system can be examined, and the overall system level can be looked at. The definitions of the DSVM system might be important to this discussion, so they shall also be addressed. A DSVM system is a spatially and/or time-distributed machine learnable problem. The system consists of a complex cost function, which includes system parameters such as network traffic, memory cost (sometimes referred to as space complexity), and CPU time. The problem can be any collection of regression, classification, or clustering problems working in hierarchies or locally distributed with the DSVM system. The node level is probably the context in which the transductive idea is introduced most easily. If a node is both producer and consumer, it would have access to some information regarding the test points. It could be that 1,000 already classified points as well as 1,000 unclassified points were placed on the same node. Because the transductive information can be used, it is clear this should be done on this node as well. When the training model is set up, then the test information is entered in the model as well. This is a particularly simple scenario as the test points are known *a priori*. A more likely situation arises when the test points are unknown beforehand. One such case is if the model is used for simulation purposes. It can be difficult, if not impossible, to know where in the input space the trained model is to be used. The straightforward case is to assign a flat probability to every point in the input space. This discussion is mainly aimed at the situations in which the test distribution falls into one of two categories. One possibility is that the test points can be known as a probability distribution. An easy and pragmatic way of using this information would be to generate a test set based on this information and then use it in the transductive manner. The second possibility is that the test set is supplied beforehand as a final set of the form $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$: I.e. without the labels that would enable us to either train the model or test the model using these points. The single producer/consumer node setup has been discussed, and now the focus can shift to more realistic situations. If the producer and consumer node are separated by some link, L_{ij}, the traffic of points would have be from the consumer node to the producer node. This is an entirely new situation that has not been covered previously. The usual path of the training points is to be trained on the producer node and then exchange only the support vector points. Now the consumer node can send its points to the producer node before the training starts, then the producer node can make use if this information while working out the support vector set for the local model. Upon training the support vectors, the possible additions are exchanged with the consumer node. It can be argued that the consumer node generates network traffic by exchanging unfiltered points in this manner. Another approach would be to query the consumer node for points using some mechanism to express where in the input space new points would significantly alter the trained model at the producer node. The consumer node would then on a point-by-point basis supply, or not supply, the information requested. A twofold challenge presents itself. First, the producer node must be able to express where in the input space the points would be. Secondly, the consumer node must be able to parse this information and pass on the best candidate. Upon receiving the point, the producer node would re-train. A proxy for how much the retraining changed the model could be a dot product of the old Lagrange multiplier vector and the new Lagrange multiplier vector. I.e. a projection of the old vector to the new one.

We have constructed the framework and identified the main parameters for using the SVM in a distributed multi-node setting in this chapter. Furthermore, we have tried to use ideas from other disciplines such as information retrieval to find methods for monitoring the effectiveness of a given data-exchange scheme. The different combination of exchanging data and/or models was supplemented with the investigation of using convex hulls as augmented exchange data. In relation to this, we note that the chosen convex hull algorithm was mainly useful for vectors of relatively small dimensional spaces.

Chapter 5

System Analysis

The chapter discuss the considerations that must be taken when an implementation is to be done. The requirements of a distributed classification system are considered, and it it necessity to consider topics such as storage, augmented exchange schemes and even the sequence of exchange of the data points. First, the storage requirements on the nodes for a classification problem are clearly of interest when working with potentially small distributed devices. The analysis will address issues that drive the analysis of the DSVM system. To begin with, the different component types are identified and analyzed.

5.1 Node Types

The distributed classification system is a large software system. In order to develop it in a structured fashion, it is necessary to divide the system into components that each have a limited scope and functionality, but will yield a fully flexible distributed inference system when configured correctly.

The necessary components are identified below, along with a list of high-level responsibilities that each component can fulfill in the distributed inference system. All implementation details concerning issues such as programming languages, libraries, hardware etc. will not be discussed

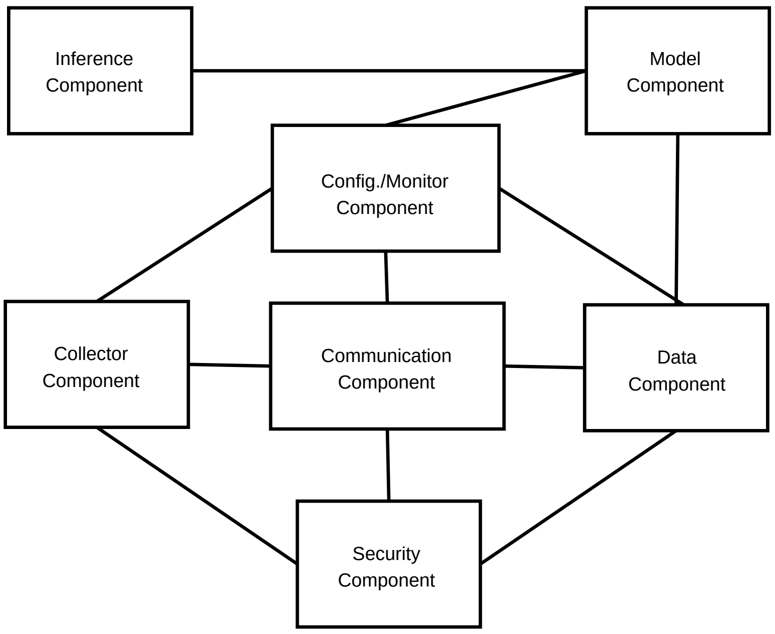

Figure 5.1: Component diagram of a complete DSVM system. The diagram identifies the main components and their primary associations.

below.

Collector Component: The component is an abstraction of any device that has the capability of collecting data. Examples of such devices could be mobile phones, sensors, web browsers etc.

Responsibilities:

- Collect data points from different sources, which could be a mobile phone, an existing data base or some flat file with data in it.
- Store data points temporary volatile buffer.
- Pass on data points to data component via communication component.

States:

- Configured: True if connected to one or more data components

- Collecting: True if collection is ongoing, such as loading and transferring a larger data source to a data node.

Data Component: Data that is ready for modelling at this time is stored on this node.

Responsibilities:

- Receive data from collector component(s).
- Store data and make it available for retrieval.

States:

- Configured: True if format of data is known.
- Loading: True if data is in being loaded from a collector component
- Loaded: True if data is loaded and ready for retrieval

The data this system runs on is a natural candidate for the inheritance hierarchy, falling naturally into input data and output data. The input data can be common across the CCR problems, with the only difference being the target variables. The target for classification is an integer, clustering problems have no target class, and regression problems have a real valued target. It is immediately clear that the inheritance branches out according to the problem type of three different subclasses. The output data plays a dual role depending on whether the system is training or testing. If the system is training, the output data determines the training error, a first indication of the extent to which the algorithm has been able to extract the underlying function from the data. If the model has been trained and new input data is presented, then output data will provide the answers sought.

Producer Component: The producer node is probably the most involved component as it will work closely with the other producer nodes depending on the model type employed. This role works closely with the consumer component. One example is when models are used to identify important data points and tag them for use at a different level in the system.

Responsibilities:

- Create a model of the data and tag data points important for the model.
- Query other model components for data points and use them in subsequent training.
- Make models available to consumer component when trained.

States:

- Configured: True if configured and connected to data components.
- Loading: True if data is being retrieved from data component(s).
- Training: True if model is being trained.
- Trained: True if model is ready.

Consumer Component: Responsibilities:

- Query a producer component for data that needs to be run by the model.
- Get the classification from the producer component.

States:

- Configured: True if configuration is set.
- Inferring: When querying of models is taking place with new data.
- Ready: When inference is done and results are ready for use.

Configuration and Monitor Component: This is a component to enable configuration and monitoring of the distributed inference system. The tasks can be divided into human interaction and machine interaction.

Responsibilities:

- Enable configuration of all other components.

- Enable manual monitoring of all other components.
- Enable automatic monitoring of all other components.
- Enable automatic re-configuration to adjust to changes in the system, such as higher and lower communication speeds and processor speeds.

States:

- Configured: True when the configuration of system is done.
- Monitoring: True when connected to and monitoring other components.

Communication Component: The communication component can be used as naming service to establish component awareness.

Responsibilities:

- Provide communication between any pair of components in the distributed inference system. The components can be of same type or different type.
- Use the security component to create a secure communication context. There is data level security, which can be a random perturbation of the data itself, or network level communication where some encryption scheme is employed.

States:

- Configured: True when configured.
- Connected: True when connected to components that are ready.

Security Component: There are many levels of security in the system and the roles of the security component are naturally many.

Responsibilities:

- Provide an appropriate security level to the system when requested by the communication component.

States:

- Configured: True when configured.
- Securing: True when connected to desired components.

- Producer node: capacity, processor speed.
- Consumer node: processor speed.
- Communication object: speed, reliability.

5.2 Persistence Management per Component:

The different components will have different needs for persistence management. Some will have a stronger need for persistence management than others.

Collector Component: The collector component's main responsibility is to be where the data is being generated and to collect it. It must either pass on the data at the same speed as it arrives at the component or save/buffer it for later transmission. If the collector component is a small distributed and disconnected device then the data is collected at the component and later retrieved or transmitted to the data component. The storage can be a relational database, a file, or flash memory.

Data Component: The data component connects downstream to the collector components and upstream to the model component. It is natural that the data component that gathers information from one or more collector components would store its retrieved information in a safe way such that it can be used at a later stage. Therefore, the data component would in some cases use some storage mechanism that is part of the whole distributed system and is less vulnerable to data loss. There can be multiple pairs of data components and model components. If just one data component fails to collect the data in that chain of models then the whole inference would fail. The raw data, the data coming from the collector component, is in most cases the important information.

Model Component: The model component that creates an inference model based on the data fed from the data component. It is clear that the model that has been trained and created can be restored from the data. In some cases it would be easy to do so. However, it might be that the model can not be restored even if retrained on the same data. In those cases is becomes important to store the model in such a way that it can be used again at a later time. If the inference component is actively using the model component to classify new data, then it becomes important that the model can be restored to its original trained state. Model types such as neural networks rarely exactly same due to the random initialization of the weights in the beginning of the training session.

Inference Component: The inference component is probably the component with the smallest need for persistence management. It clearly depends on a correct model component being in place. Fore performance reasons, some results results might be cached, but actually saving the results on persistent storage would be more than was asked for.

Presentation Component: The presentation component is the user's view of the results. There might be multiple views of the data, some of which might involve successive calls to the inference component to show new dimensions of the results. It would probably be necessary to store the presentation results, as it could be advantageous in terms of creating some independence between the presentation layer and the data layer. The presentation of the results could also be so intensive in terms of network communication and rendering of display objects such that is it something the needs to be minimized by using the persistent storage for retrieving the results multiple times without needing to issue queries backwards onto the storage system.

5.3 Data Identification

The main components for the DSVM system have been discussed in the previous sections. The identification of data points becomes more important in a statistical model setting than in a situation where the data is just collected and transmitted as-is. Therefore, this section will examine choices for distributed data identification useful for the SVM learning machines.

The points belong to the node of which they were created. Therefore, a natural way of identifying the points is to associate a primary key. This can be a date/time stamp or another number that serves as a primary key. We will call this for *explicit identification* because it distinguishes the points from one another once they are generated.

Another option for creating an identification scheme is to use the attribute values for generating a key. An obvious choice for this key generation scheme would be a checksum calculation. The advantage of generating the key using a well-known algorithm would be that the point could be exchanged in the system in raw form and would not have to have a primary key associated with it. The problems that can be anticipated are when some node N_i gets a point from another node N_j and the checksums match. Or if node N_j fails to resend its points when it reenters the DSVM system. Or there are many points on node N_j with the same attributes. The reason why this data based identification schemes work is because of the way the SVM works. It is a discriminative algorithm, and in its hard margin version it would not change even if one or one million points had the same attributes (and the same label). The reason is that in the hard-margin case it is only the margin that counts. Points that would match are removed in a preprocessing step before being allowed to enter the training algorithm.

It is also possible to create a pseudo checksum system to further compress the points. Again, recall that node N_i would probe node N_j for new points and then receive an array of checksums. Node N_i would then query only for those points for which there is no match on N_j. If two points of same class differ by only a small fraction of the attributes, there is still some probability that the point would alter the solution of the SVM significantly. This is in some contrast to the neural network, in which the extra points would play an important role, as this algorithm uses information about the density of the distribution of the points and can be adapted to provide posterior probabilities. For classification purposes the pseudo checksum can be used when two points are of same class, but the input vector varies insignificantly. One way to check this is to round the decimal points to a number of significant digits so that the checksum is performed on a less precise number of digits than the actual attribute value in the vector. This should ensure that similar points are not exchanged if one such point is already residing on the destination node. The approach to exchanging information is robust for the SVM kind of algorithm that seeks to find the maximal margin of some points.

The similarity mechanism represented in the SVM would also allow it to filter, which would filter out any points that were similar. Again, the strength of the SV set has been affirmed to work also as a preprocessing step.

In this chapter, a more abstract view has been taken on the DSVM system. It should allow implementation in different languages and on different platforms to follow a shared vocabulary. Analytically, we divided up the system into components and for each component the main responsibilities were pointed out. We used this analysis when the prototype was programmed in Java. Some components such as the security component was not implemented. However, many of the same components are found again when investigating the source code [32].

Chapter 6

Energy Awareness

Experiments are conducted with different settings on three selected systems, and we reach an interesting yet counterintuitive result regarding the energy efficiency of large sensor nodes vs. energy efficiency of small sensor nodes. Energy awareness in distributed nodes can be achieved by using and incorporating machine learning algorithms on each of the individual nodes. In this chapter we experiment energy consumption of an SVM as a tool for embedding such algorithms in distributed nodes.

The machine learning community provides the SVM algorithm [93], while energy awareness combined with sensor networks provides the domain in which we incorporate this algorithm. One approach is to incorporate self-configuration in the system: First, we can use the algorithm to perform a bootstrap test of the node's CPU speed, and then it can self-configure the SVM. Secondly, the node starts filtering or performs other intelligent tasks. This approach is possible on a JStamp processor as it can be configured to run at different CPU speeds, which could lead to different settings of the SVM algorithm.

We investigate the energy consumption for a constructed binary classification problem using our implementation of the SVM in the Java language [32]. The work can can be split into three main components. One is related to the distributed system itself while the second component corresponds to the choice of Java as the programming language. The final component consists of the chosen machine intelligence algorithm. In terms of the distributed aspects, we address this by identifying the idea that SVMs depend only in part on a subset of the data examples called the support vectors (SVs). This is an inherent characteristic of the support vector machine algorithm that applies across classification, regression, and cluster analysis. These three types of analysis problems can be addressed within the SVM framework. Java can potentially be an attractive platform for distributed computing since the terms ubiquitous, pervasive, and ambient computing are likely to fit well into Sun's ”Write Once, Run Anywhere” philosophy. In a double effort to assess Java's capabilities as well as make the successor to neural networks - the SVM - possible in distributed environments [70], we present the formulas for how to estimate the additional or marginal energy consumption when using a DSVM on three kinds of nodes: an IBM laptop, a Symbian OS based mobile phone, and an embedded Java chip.

This chapter is organized as follows: first we relate our work to previous research. Our concept of the DSVM is then introduced. The experiments section follows with insights into the code profiling of the DSVM, porting of the Java code from the J2SE API to the CLDC API, with the main experimentation focusing on approximating a formula for predicting the energy usage of the node given a set of parameters. Our experiments center on three node types:

These four nodes are further described in the experiments section in terms of operating system, type of processor, power consumption, and weight. Within the scope of these nodes we perform experiments and present some interesting results. The experimental results fall into three categories: (1) How many sensed points can the sensor nodes classify per second? (2) How much energy does each type of sensor node use to classify one point? (3) Guidelines for configuring the nodes to self adapt the algorithm.

We conclude the chapter by summarizing the main results and pointing toward the future of our work. An observation of particular interest is that our experimental setup shows about a factor three difference in the amount of energy used by the big node, medium node, and small node to

Figure 6.1: The big Java node: IBM A31

Figure 6.2: The medium Java node: Sony Ericsson p800

classify a point. The smallest node was the most energy consuming, which was not something we intuitively expected a priori. Please note this might be specific to our particular setup.

6.1 Experiments

The goal of the experiments is to gain a greater understanding of the energy consumption of the Java sensor nodes. We use a code profiler, software timers, and voltmeters to analyze the energy aspects of the code. Direct profiling of the code in terms of how time is spent when running the node DSVM provides insight into the time spent in each part of the program code. Moreover, the node analysis is centered on the marginal energy usage of running a binary classification problem. It should be

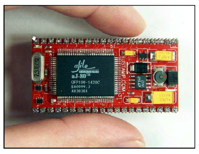
Figure 6.3: The small Java node: Systronix JStamp

Figure 6.4: The TinyOS node CPU: ATMEGA128L

noted that a large portion of machine learning and pattern recognition is a binary classification problem. Therefore, we have chosen to start with this problem. The base power consumption of the device can be defined as the energy used when performing basic tasks such as listening to the radio transmitter and capturing data. We measure the additional power consumption by loading a data set and monitor the device during training, i.e., when the Java Virtual Machine (JVM) runs at 100% speed. First, the code profiling is performed. This is a preliminary step leading up to—but not directly linked to—the energy consumption analysis. An additional experiment is performed to obtain an empirical analysis of how predictable the number of kernel evaluations is for a real world data set. Again, this is used mainly as a path toward the third and final series of experiments, which actually measures and profiles the energy consumption of the nodes. This concluding experiment provides our main contributions, which are a number of formulas for predicting the energy usage for different size Java-based nodes. Furthermore, the energy consumption formulas can be used in a bootstrap auto configuration mode by the nodes, if they are allowed to sample the speed of the node running the system.

6.2 Energy Consumption Profiling

Energy consumption is measured on three toy problems using four nodes of different hardware and software. Each of the three problems trains a binary SVM classifier with two, three, and, four sv. The experiment is then to retrieve the functional output of the trained SVM for different numbers of test points. This experiment is conducted on three node platforms: a big node, a medium node, and a small node. The goal of the experiment is to better energy consumption for various types of nodes because it will allow for an analysis of which node is most energy efficient.

The setup of the binary SVM classifier for this problem is to use the dotproduct kernel, $<\boldsymbol{x}_i \cdot \boldsymbol{x}_j>$. It can be an advantage to keep the dual formulation even though the kernel is linear as this allows for later substitution of other kernels and straight forward modification of the energy consumption formulas. In this example we use the dot product kernel, which is at least three times less expensive to calculate than the Gaussian kernel [73].

We set up the experiment for three nodes. The first is a standard laptop with a standard JVM, the second is a Sony Ericsson p800 mobile phone, the third is a small, native-Java processor called JStamp by Systronix [89], and the last is an ATMEGA128L 8-bit processor.

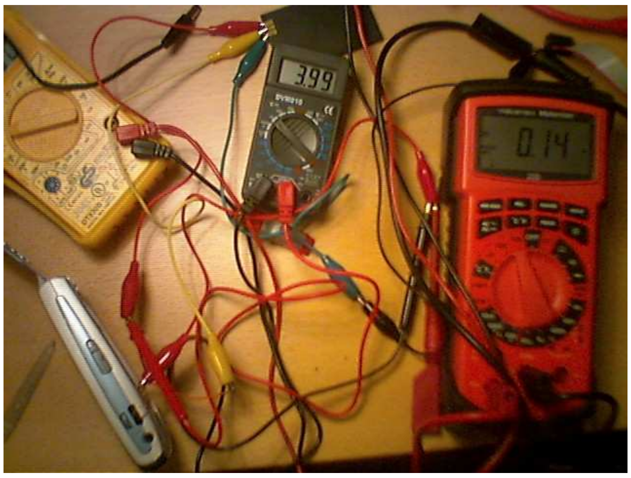

Figure 6.5: Experimental setup for the p800. Note the total system power consumption of $3.99\ V * 0.14\ A = 559\ mW$ for a running JVM. The measurement instrumentation is placed between the battery and the phone.

Table 6.1: Properties of the Nodes

	Big	Medium	Small	Tiny
OS	Win 2000	Symbian 7.0	JEM2	TinyOS
Processor	Intel P4, 1.4 GHz	32-bit RISC ARM9	aJ-80	ATMEGA128L
Environment	JVM	KVM	CLDC	TinyOS
API	J2SE 1.4.2	Sun PJAE 1.1.1a	CLDC 1.0	nesC
Weight	3.18 kg	148 g	10.2 g [a]	0.46 g [b]

[a]The weight of the print board and components are included.
[b]Weight ATMEGA128L chip without board etc.

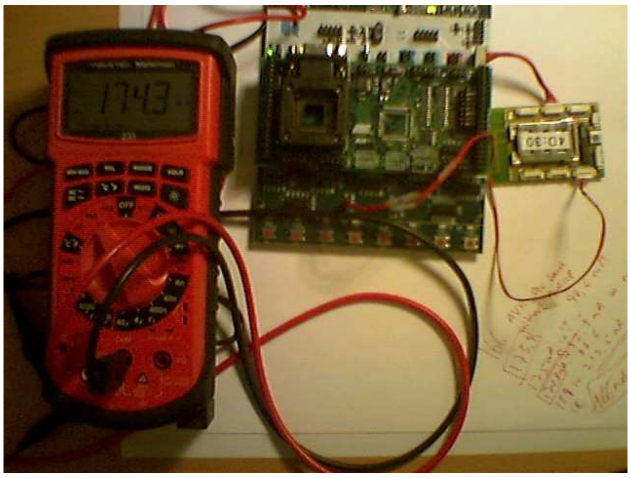

Figure 6.6: Experimental setup for the ATMEGA TinyOS Node. The total developer system power consumption including the running tinySVM program is $11.8\ V * 174.3\ mA = 2.1\ W$.

Three nodes are equipped with different JVMs and one is running TinyOS.

The big node has a traditional JVM, which is utilized by Java end-users. Implementation of the Java interpreter on the medium node is based on Sun's small KVM. On the small node, the Java interpreter is based on the native Java executing chip from aJile. Lastly, the Tiny node is running TinyOS, which is programmed using nesC language [40]. It should be noted that the JStamp processor has a lower energy consumption than the full developer station. The JStamp processor uses 200mW when running on a 3.3 V DC battery. This is close to the 260mW that we measured as the extra power used when running the JStamp versus not. The ATMEGA processor draws 100 mW according to the specifications, which makes the 97.9 mW quite accurate. However,

Table 6.2: Node Electricity and Power Properties

	Big	Medium	Small	Tiny
Voltage	16.3 V	3.99 V (battery)	15.22 V	11.8 V
SVM Loaded	1230 mA	10 mA	26.4 mA	166.0 mA
SVM Running	1950 mA	140 mA	43.5 mA	174.3 mA
Power Consumption[a]	11,736 mW	518.7 mW	260.3 mW	97.9 mW

[a]Power consumption is calculated by the difference in total system power with the SVM running minus the SVM in idle state. This gives the marginal power consumption.

our approach to estimating the marginal energy is rater crude as components on the development boards can interfere.

The experiment measures how much time it takes the three different systems to classify different numbers of p. The experiment is repeated three times while increasing the number of SVs in the SVM model. In the first run, the number of SVs is just two, then three in the second run, and finally four in the last run. After the series of experiments, we have time measurements that depend on two parameters: the number of SVs in the algorithm and the number of p to classify. If we can estimate the formula for the prediction time depending on the number sv and p then it is possible to use that formula in conjunction with power consumption to estimate the energy usage of the node, and thus how much of its battery life we use on a given task.

The classification portion of the DSVM program has been ported to nesC. This SVM is labelled *tinySVM*. An important difference between the big/medium/small systems and the tiny system is that the ATMEGA chip does have support for floating point arithmetic. Therefore it solves a simpler task than the other three systems in that all Java `double` variables have been interchanged with `uint32_t`. It could still provide some new insight to compare across the two systems.

It is possible to calculate the average number of p each of the systems can predict each second using time measurements. This is derived by dividing the sum of points with the total time for each of the systems in the tables: Table 6.3, Table 6.4, and Table 6.5 into Table 6.6.

Table 6.3: Time Measurements for SVM with 2 Support Vectors

p	Big ms	Medium ms	Small ms	Tiny ms
100,000	71	4,172	25,984	16,023
200,000	130	8,328	51,967	18,928
300,000	170	12,500	77,950	28,400
400,000	210	16,640	103,934	37,865
500,000	251	21,594	129,918	47,328
600,000	300	28,703	155,901	56,802
700,000	351	29,110	181,885	66,265
800,000	400	38,093	207,869	75,739
900,000	451	39,719	233,853	85,202
1,000,000	500	44,531	259,836	94,667

Table 6.4: Time Measurements for SVM with 3 Support Vectors

p	Big ms	Medium ms	Small ms	Tiny ms
100,000	100	7,672	38,136	13,479
200,000	171	12,282	76,273	26,959
300,000	240	18,297	114,410	40,448
400,000	270	24,406	152,547	53,927
500,000	351	30,515	190,683	67,407
600,000	410	36,604	228,820	80,887
700,000	491	42,687	266,957	94,365
800,000	551	49,797	305,093	107,855
900,000	621	55,594	342,827	121,335
1,000,000	701	62,047	380,919	134,814

Table 6.5: Time Measurements for SVM with 4 Support Vectors

p	Big ms	Medium ms	Small ms	Tiny ms
100,000	120	8,250	50,305	17,495
200,000	230	17,969	100,610	34,990
300,000	301	24,500	150,916	52,496
400,000	380	32,625	201,222	69,990
500,000	471	40,766	251,526	87,486
600,000	551	48,937	301,831	104,971
700,000	661	56,422	352,137	122,476
800,000	761	64,500	402,442	139,972
900,000	841	74,344	452,747	157,476
1,000,000	942	80,625	503,053	174,972

Table 6.6: Average Number of Classified Points Per ms

SV	Big ms	Medium ms	Small ms	Tiny ms
2	1,940.7	22.6	3.8	10.4
3	1,408.1	16.2	2.6	7.4
4	1,046.0	12.3	2.0	5.7

Table 6.7: Average Time for Classifying One Point

SV	Big	Medium	Small	Tiny
2	$0.515 \times 10^{-6}\ s$	$44.3 \times 10^{-6}\ s$	$263.2 \times 10^{-6}\ s$	$95.9 \times 10^{-6}\ s$
3	$0.710 \times 10^{-6}\ s$	$61.8 \times 10^{-6}\ s$	$384.6 \times 10^{-6}\ s$	$134.8 \times 10^{-6}\ s$
4	$0.956 \times 10^{-6}\ s$	$81.6 \times 10^{-6}\ s$	$500.0 \times 10^{-6}\ s$	$175.0 \times 10^{-6}\ s$

It is evident that the big system is much faster than the three smaller systems, as expected. The ATMEGA chip is faster than the JStamp.

Time equations for each of the systems can be calculated using ordinary least square regression (we discard the constant term) on each of the three observation sets in Table 6.7. That then yields the time it takes to classify a number of p for an SVM model with a given number of sv. The results of this regression are shown in (6.1), (6.2), (6.3), and (6.4):

$$t_{big}(sv, p) = 0.221 \times 10^{-3}\ ms \times sv \times p \tag{6.1}$$

$$t_{medium}(sv, p) = 19.15 \times 10^{-3}\ ms \times sv \times p \tag{6.2}$$

$$t_{small}(sv, p) = 118.4 \times 10^{-3}\ ms \times sv \times p \tag{6.3}$$

$$t_{tiny}(sv, p) = 39.6 \times 10^{-3}\ ms \times sv \times p \tag{6.4}$$

Marginal energy equations can be constructed by multiplying the time in (6.1), (6.2), (6.3), and (6.4) for each node with the marginal power consumption in Table 6.2, as measured earlier. The results are in (6.5), (6.6), (6.7), and (6.8).

$$\begin{aligned} e_{big}(sv,p) &= 11,736\ mW \times 0.221 \times 10^{-3}\ ms \times sv \times p \\ &= 2.59\ \mu J \times sv \times p \end{aligned} \tag{6.5}$$

$$\begin{aligned} e_{medium}(sv,p) &= 518.7\ mW \times 19.15 \times 10^{-3}\ ms \times sv \times p \\ &= 9.93\ \mu J \times sv \times p \end{aligned} \tag{6.6}$$

$$\begin{aligned} e_{small}(sv,p) &= 260.3\ mW \times 118.4 \times 10^{-3}\ ms \times sv \times p \\ &= 30.82\ \mu J \times sv \times p \end{aligned} \tag{6.7}$$

$$\begin{aligned} e_{tiny}(sv,p) &= 97.9\ mW \times 39.6 \times 10^{-3}\ ms \times sv \times p \\ &= 3.9\ \mu J \times sv \times p \end{aligned} \tag{6.8}$$

$$e(sv,p) = \begin{cases} 2.59\ \mu J \times sv \times p & \text{for big node} \\ 9.93\ \mu J \times sv \times p & \text{for medium node} \\ 30.82\ \mu J \times sv \times p & \text{for small node} \\ 3.9\ \mu J \times sv \times p & \text{for tiny node} \end{cases} \tag{6.9}$$

It may be useful to summarize this analysis with two examples of how these results can be applied.

Example 1: Estimation of marginal energy usage for classifying 500 testpoints on a JStamp node for an SVM based on three sv.

$$e(sv = 3, p = 500) = 30.82\ \mu J \times 3\ sv \times 500\ p = 46.23\ mJ \tag{6.10}$$

In this example, the JStamp system would use about 46 mJ to classify the 500 new points.

Example 2: Should a node classify on the node or pass on the data directly to another node for remote classification and then receive back the model? The answer depends on the cost of radio transmission of the full data set, the cost of the remote classification, and the cost of transmitting back the model to the node. For some configurations the optimal decision would be to classify locally, and in other instances it is better to send the data to a less energy consuming node.

6.3 Energy Usage in Local Classification versus Exchange

One hypothesis of the DSVM system is that it should be advantageous in some situations to classify new observations locally before making a decision if the radio should be activated. In this experiment we test the energy consumption of classifying one new point on the JStamp vs. performing an immediate exchange of the point. The radios used in the experiment *MaxStream 24XStream 2.4 GHz 9600 Baud Wireless Module*. First, the JStamp developer station is connected to one radio and the second radio is connected to the Big node6.1. Then the `ClassificationOutputDataPoint` is serialized in Java, and the `byte` array is written to the `javax.comm.SerialPort` class of the JStamp developer board.

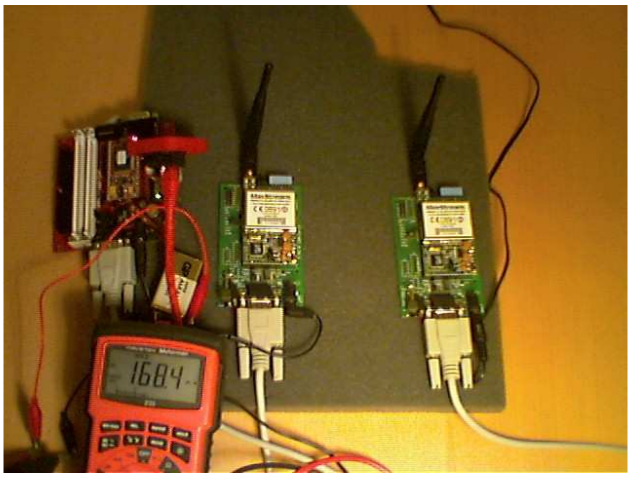

Figure 6.7: The wireless setup with MaxStream radios and JStamp. The JStamp connected MaxStream radio is transmitting while the ampere meter display a reading of 168.4 mA.

Table 6.8: Experiment Data for Local Classification Versus Exchange

Item	Measurement
Data point serialized	$102\ bytes$
Radio idle	$85.3\ mA \times 7.98\ V = 680.7\ mW$
Radio sending	$168.4\ mA \times 7.65\ V = 1,288.3\ mW$
Radio marginal	$1,288.3\ mW - 680.7\ mW = 607.6\ mW$
Data exchange	$107.5\ ms \backslash datapoint$
Local classification	$263.2 \times 10^{-6}\ s$
Number of SVs	2

The basic experimental results are presented in Table 6.8. It is interesting that the size of the serialized data point is quite large but that is the result of programming in an object oriented manner. The `ClassificationOutputDataPoint` contains several other objects [32] and each of those add size to the data object, which also contains its α multiplier. This is an advantage of the SVM that the datapoint and the α are so closely related. For the radio, the marginal energy consumption has been calculated by subtracting the idle energy from the sending energy. The difference is the marginal energy, which will be used to calculate the cost of sending a data point. It takes the modem $102ms$ to exchange the data point. This result was achieved by exchanging 100,000 datapoints and then taking the average. The associated `java.io.OutputStream` of the serial port was flushed between each point sent. We also note the time spent on classifying a novel data point on the JStamp from Table 6.7 containing the average time for classification with 2 SVs.

As a result of this experiment we would like to understand the energy cost ratio between classifying a data point locally versus just transmitting it over the wireless link in a serialized form. This ratio is calculated by dividing the cost of radio transmission with the cost of local classification.

$$r_{radio\_vs\_classification} = \frac{607.6\ mW \times 107.5 \times 10^{-3}\ s}{260.3\ mW \times 263.2 \times 10^{-6}\ s} = 953.4 \sim 10^3 \qquad (6.11)$$

To check the transferspeed we can note that the wireless modem is set up with 1 stop bit and no parity bit. With the start bit and the 8 bits of data then each of the 102 bytes are of length 10 bits. The achieved transfer rate is thus $\frac{1000\ ms}{107.5\ ms/point} \times 102\ bytes/point \times (8\ bit + 2\ bit) = 9488\ bit/s$ which is close to the 9600 baud specification.

6.4 Discussion of Energy Results

Our results fall into four categories:

1. Porting and analysis of the SVM algorithm in a distributed setting.
2. Energy profile of the system on three systems that run on J2SE, pJava, CLDC 1.0, and TinyOs/ATMEGA128L.
3. Demonstration that the embedded Java device can perform classification using an SVM based on Java and nesC.
4. Experimental demonstration suggesting that local classification is about 10^3 less energy consuming than radio exchange.

There are two issues of particular interest. One is that the big Java node is more energy efficient than the smaller Java nodes. In terms of the ATMEGA sensor node equipped with TinyOs, it is interesting to note that it is 7 times less energy consuming to classify on that node compared to the JStamp node. However, since the TinyOs program did not use floating points, then the direct comparison of the two system is not possible.

The classification versus radio exchange and the three energy equations in (6.9) provide the key results as it can be counterintuitive that both the medium (the p800) and the small node (JStamp) use approx. 3 and 10 times more energy than the big node (standard laptop) when classifying a new data instance.

It is foreseeable that radiocommunication cost will play an important role in future designs of distributed and wireless Java based sensor networks. We consider building a simulator that can be configured with the energy cost to assist designers of wireless Java-based sensor networks to strike the balance between computing locally on the node versus sending information to the a more powerful central node. Commercial availability of wireless RF modems for the popular JStamp developer station [89] is scheduled for fall 2004, which will further enable development of wireless Java based sensor networks.

Chapter 7

Conclusion

The fundamental logic of the DSVM system is that SVMs provide configurable solutions that inherently serve to address the constraints found in the four main constraint categories of computational, spatial, temporal, and communication costs. We have laid out the foundation for a framework that is a natural way to leverage important elements of SVMs and statistical learning theory. In the introduction, we set out to answer the following two research questions:

- Is the standard support vector machine useful for distributed machine learning?
- Can the support vector machine be applied specifically for distributed machine learning?

As an answer to the first question we say: perhaps. To the second research question, we conclude: yes it can. We consider that our main contribution. The answers are provided throughout the thesis, but a summarization is appropriate. The standard support vector machine required storage of a large kernel matrix and specialized software packages to solve the associated quadratic program. So the answer to the first research question is that the standard support vector machine presented in the paper by Boser et al. in 1992 [15] is not as useful as the support vector machine presented by John Platt labelled Sequential Minimal Optimization (SMO). Platt's extensions make it possible to balance memory requirements during training of the learning machine. The support vector machine is useful in distributed machine learning. The second, and closely related, research question is if the support vector machine can become useful in distributed machine learning. Again, the answer must be positive, as we have researched and presented many aspects that directly addressed the resource constraints that the support vector machine is subject to in distributed systems.

In the chapter on related research we touched on sensor network applications. The small operating system TinyOS presently dominates, but other sensor networks based on complementary technology such as Java are likely to emerge. One such example is a field programmable gate array (FPGA) open source Java virtual machine implemented by Schoeberl [78]. We also found that distributed data mining is emerging, which is supported by the 278 entries in the Distributed Data Mining Bibliography (DDMBIB) maintained by Liu and Kargupta.

The single-node SVM chapter introduced the main constraints that are relevant for distributed systems of potentially small nodes. We presented the concepts of general [18] and orthogonal support vectors [31], which further reduce the number of support vectors and thus the memory requirements on a node. The potentially high energy consumption associated with training of multiple SVMs while tuning SVM parameters was successfully addressed by the experiments on re-using the Lagrange multipliers, which is a key parameter in the SVM.

In the multi-node setting, we expanded the framework vocabulary by defining a producer node and a consumer node. We used this notion while discussing the applicability of SVMs for distributed novelty detection, regression, classification, and clustering. Our experiments on advanced exchange schemes related to support vectors, convex hulls, η/δ exchange, and the harmonic mean related to information retrieval added further insight into the inherent, significant filtering/compression capabilities of SVMs.

We devote a chapter to energy awareness experiments with the DSVM code. It yielded the result that the small Java node was less energy efficient than a bigger faster Java node. However, the small Java node is fast enough to classify a high number of unseen data per second on the artificial data used across the experiments. An especially interesting experiment positively affirmed the energy savings by local classification and filtering versus raw exchange of data. Furthermore, we tested a TinyOS node even though the results were not directly comparable, but it served partly to demonstrate that an SVM can function well across different operation systems and hardware platforms.

Our contributions is the framework laid forth, which might also be applicable to other machine learning tasks. One type of constrained machine learning is onboard cloud detection processing [84]. Another example of an application area is object tracking in sensor networks [16]. Yet others are defined by the semantic meaning of the data distribution itself such as the CarMining project [97]. On the system level, some of these networks can be characterized by a high number of nodes that are loosely coupled by a wireless connection protocol. On the other end of the spectrum we find satellites or ELF (extremely low frequency) communication objects such as submarines. ELF communication links could be an example of a communication constrained edge in a DSVM system.

The reference implementation has been done in Java, as this environment is flexible in choice of underlying hardware. However, this flexibility comes with a price, as noted in (6.9). It seems like an interesting research opportunity to push Java further toward the same low energy consumptions as the TinyOS sensor nodes have. It would require to construct a specialized small JVM that fits in memory footprints smaller that the current Java CLDC 1.1 specification of 192 kilobytes. The TinyOS environment could be the target for an upcoming implementation of the DSVM framework. It presents some interesting challenges such as the need for fixed point and integer arithmetic because the smallest devices would not normally support floating point operations. We will call the system for *tinyMiner* and the DSVM implementation component for *tinySVM*.

There are limitations to the research done in this thesis. The implementation in Java, Matlab, and TinyOS fully determines the results of the experimental sections. In the future, there might be other distributed SVM implementations, which can serve as benchmark applications, but none are present as of today. A candidate for such benchmark implementation is the Weka software data mining package [39], which can be re-programmed in a edition suited for smaller devices. In fact, we are looking into making a *WEKA-Egg* implementation of WEKA, which will be able to run on small Java nodes. In relation to pure machine learning, there would be several experiments that could be tested with the DSVM, but a starting point would be to split existing data sets evenly over a number of nodes and compare the final inference models to performance on the same non-partitioned set. A main outcome of the experiments is to get the energy profile of the consumer nodes and the producer nodes analyzed, such as was done in the energy awareness chapter. It is to be expected that the energy profile is more energy intensive than linear counterparts such as ordinary least square regression with a suitable discriminant function. However, those cases in which non-linearity in the problem domain demand kernels such as the Gaussian kernel this SVM-based framework would fit in.

In conclusion, we hope to have laid forth the foundations of a framework rigid enough to legitimatize further research with respect to using the SVM and other kernel related algorithms in constrained, distributed data mining problems.

Appendix A

Danish Abstract

I denne afhandling undersøger vi om en support vektor maskine (SVM) er brugbar til distribueret maskine læring. En SVM er en algoritme fra området maskine læring, som kan bruges til klassificering, regression og andre vigtige opgaver. Det nye i denne afhandling er at bruge SVM som gensidigt lærende enheder samtidigt med at problemets distribuerede kontekst respekteres. Vi konstruerer problemet således, at signifikante begrænsninger i det distribuerede indlæringssystem introduceres for at tilføje yderligere dimensioner til forskningsproblemet. I dette system, som vi kalder Distributed Support Vector Machine, er det ikke nok kun at se på SVM maskinens målfunktion. I mange situationer vil der skulle tages hensyn til afgrænsninger i relation til de distribuerede indlæringsenheder. Ideen, om at bruge SVM som samarbejdende indlæringsenheder i distribueret maskine læring, er ikke før testet. Denne afhandling præsenterer derfor de fundamentale rammer og retfærdiggør yderligere forskning på området.

Bibliography

[1] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, 2000.

[2] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 9–16. Morgan Kaufmann, San Francisco, CA, 2000.

[3] J. Aronis, V. Kolluri, F. Provost, and B. Buchanan. The world: Knowledge discovery from multiple distributed databases, 1997.

[4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[5] Arash Baratloo, Mehmet Karaul, Holger Karl, and Zvi M. Kedem. An infrastructure for network computing with Java applets. *Concurrency: Practice and Experience*, 10(11–13):1029–1041, 1998.

[6] A. Ben-Hur, D. Horn, H.T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[7] K. Bennet and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 11*, pages 368–374. MIT Press, 1998.

[8] R. Bhargava, H. Kargupta, and M. Powers. Energy Consumption in Data Analysis for Onboard and Distributed Applications. In *Proceedings of the 2003 International Conference on Machine Learning workshop on Machine Learning Technologies for Autonomous Space Applications*, 2003.

[9] IBM Senior Technical Staff Member Bill Bodin (reference is permitted pr. e-mail on July 13th, 2004). Private conversation. University of Texas, Austin, spring 2002, 2002.

[10] W. Binder, J. Hulaas, A. Villazon, and R. Vidal. Portable resource control in java: The j-seal2 approach. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01)*, 2001.

[11] Walter Binder and Volker Roth. Secure mobile agent systems using java: where are we heading? In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 115–119. ACM Press, 2002.

[12] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford, 2000.

[13] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[14] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14. Springer-Verlag, 2001.

[15] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

[16] R. Brooks, P. Ramanathan, and A. Sayeed. Distributed target classification and tracking in sensor networks. In *Proceedings of the IEEE*, pages 1163–1171, 2003.

[17] M. Brown, H. G. Lewis, and S. R. Gunn. Linear spectral mixture models and support vec-

tor machines for remote sensing. *IEEE Trans Geoscience and Remote Sensing, submitted*, 38(5):2346–2360, 2000.

[18] C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 71–77. Morgan Kaufmann, San Mateo, CA, 1996.

[19] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

[20] C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proceedings of the 17th International Conference on Machine Learing, (ICML2000, Stanford, CA, 2000)*, page 8, 2000.

[21] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems (NIPS*2000)*, volume 13, 2001.

[22] Philip K. Chan and Salvatore J. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *International Conference on Machine Learning*, pages 90–98, 1995. The paper introduces a meta-learning scheme that is denotes by en arbiter, which is a scheme for creating a tree that eventually comes out with a prediction. This might be interesting in terms of exchanging predictions in the classifier system.

[23] C. Chang and C. Lin. LIBSVM: a library for support vector machines (version 2.3), 2001.

[24] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 2002.

[25] Rita Yu Chen and Bill Yeager. Java mobile agents on project jxta peer-to-peer plat-

form. In *Proceedings of 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, 2003.

[26] Nello Christianini and John Shawe-Taylor. *Support Vector Machines and Other kernel-based Learning Methods*. Cambridge University Press, 2000.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[28] W. Davies and P. Edwards. Agent-Based Knowledge Discovery. In *AAAI Spring Symposium on Information Gathering*, 1995.

[29] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *International Conference onInternational Conference on Knowledge Discovery and Data Mining (KDD-2000)*, 2000.

[30] Bureau International des Poids et Measures. The international system of units (si), http://www.bipm.fr/en/si/.

[31] Tom Downs, Kevin E. Gates, and Annette Masters. Exact simplification of support vector solutions. *J. Mach. Learn. Res.*, 2:293–297, 2002.

[32] DSVM. Distributed support vector machine server and implementation, www.dsvm.org.

[33] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, second edition edition, 2000.

[34] Joydeep Ghosh (ed. Nong Ye). *Handbook of Data Mining*, chapter 10, pages 247–277. Lawrence Ealbaum Assoc., 2003.

[35] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd Interna-*

tional Conference on Information and Knowledge Management (CIKM'94), pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[36] G. Flake. Support vector machines for regression problems with sequencial minimal optimization, 1999.

[37] Gary William Flake and Steve Lawrence. Efficient svm regression training with smo. *Machine Learning*, 2001.

[38] Glenn Fung and Olvi L. Mangasarian. Proximal support vector machine classifiers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 77–86. ACM Press, 2001.

[39] S. Garner. Weka: The waikato environment for knowledge analysis, 1995.

[40] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11. ACM Press, 2003.

[41] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford University, Stanford, CA, USA, 1992.

[42] Soheil Ghiasi, Ankur Srivastava, Xiaojian Yang, and Majid Sarrafzadeh. Optimal energy aware clustering in sensor networks. *Sensors*, 2:258–269, 2002.

[43] Mark Girolami. Mercer kernel based clustering in feature space. *I.E.E.E. Transactions on Neural Networks*, 2001.

[44] D. L. Grecu and L. A. Becker. Coactive Learning for Distributed Data Mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 209–213, New York, NY, August 1998.

[45] Z. Haas, J. Deng, B. Liang, P. Papadimitatos, and S. Sajama. Wireless ad hoc networks, 2002.

[46] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Support vector learning for ordinal regression. In *Ninth Intl. Conf. on Artificial Neural Networks*, pages 97–102, 1999.

[47] Gayle W. Hill. Group versus individual performance: Are n + 1 heads better than one? *Psychological Bulletin*, 91(3):517–539, 1982.

[48] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41–52, June 2004.

[49] Tin Kam Ho. *Multiple Classifier Combination: Lessons and Next Steps*, volume 47 of *Series in Machine Perception and Artificial Intelligence*, chapter 7, pages 171–198. World Scientific, 2001.

[50] C. Hsu and C. Lin. A comparison of methods for multi-class support vector machines, 2001.

[51] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[52] E.M. Jordaan. Development of adaptive, online soft sensors. Master's thesis, Stan Ackermans Institute, Eindhoven, 1999. ISBN 90-5282-993-4.

[53] Eric Jul. *Object Mobility in a Distributed Object-Oriented System.* PhD thesis, Department of Computer Science, University of Washington, Seattle, December 1988.

[54] Eric Jul. Emerald paradigms for distributed computing. *ACM SIGOPS European Workshop*, 1992.

[55] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. *ACM Trans. Comput. Syst.*, 6(1):109–133, 1988.

[56] Niels Christian Juul. *Comprehensive, Concurrent, and Robust Garbage Collection in the Distributed, Object-Based System, Emerald*. PhD thesis, University of Copenhagen, Department of Computer Science, 1993. DIKU-rapport 93/1.

[57] Niels Christian Juul and Eric Jul. *Comprehensive and Robust Garbage Collection in a Distributed System*, volume 637. Lecture Notes in Computer Science, proceedings of iwmm'92 edition, 1992.

[58] H. Kargupta, B. Park, D. Hershbereger, and E. Johnson. Collective data mining: A new perspective toward distributed data mining, 1999.

[59] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 2000.

[60] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to platt's smo algorithm for svm classifier design. Technical report, National University of Singapore, 1999.

[61] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical Report TR-ISL-99-03, National University of Singapore, Bangalore, India, 1999.

[62] Martin Leopold, Mads Dydensborg, and Philippe Bonnet. Bluetooth and sensor networks: A reality check. In *1st ACM Conference on Sensor Networks*, 2003.

[63] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 491–502. ACM Press, 2003.

[64] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[65] P. Mitra, C. A. Murthy, and S. K. Pal. Data condensation in large databases by incremental learning with support vector machines. In *Proc. Intl. Conf. on Pattern Recognition (ICPR2000)*, 2000.

[66] Bernard Moulin and Brahim Chaib-Draa. An overview of distributed artificial intelligence. *Foundations of Distributed Artificial Intelligence*, pages 3–56, 1996. It is interesting that DAI. That distingtion between multiagent systems and distributed problem solving is interesting. DAI is not concerned with parallel or distributed processing. Bond and Gasser (1992) indicate: ”DAI is concerned with issues of coordination among concurrent processes at the problem-solving and representation levels. That is, they are not concerned with parallel processing for reasons of improved effiency *par se*. It might be worth looking at the observations as agents and the nodes as a sort of superagent that has more reasoning power. The notion of a ”black board” is introduced as a shared memory area.

[67] Andy Oram, editor. *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*. O’Reilly & Associates, 2001.

[68] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[69] R. U. Pedersen. Distributed support vector machine for classification, regression and clustering. In *COLT03 Kernel Impromptu Poster Session*, 2003.

[70] Rasmus U. Pedersen. Distributed support vector machine. In *Proceedings of International Conference on Intelligent Agents, Web Technology and Internet Commerce*, 2003.

[71] Rasmus U. Pedersen. Using support vector machines in distributed and constrained machine learning contexts. Data Mining in Resource Constrained Environments Workshop at Fourth SIAM International Conference on Data Mining, 2004.

[72] Cristiano Pereira, Sumit Gupta, Koushik Niyogi, Iosif Lazaridis, Sharad Mehrotra, and Rajesh Gupta. Energy efficient communication for reliability and quality aware sensor networks. TR 03-15, April 2003.

[73] John Platt. *Fast training of support vector machines using sequential minimal optimization in Advances in Kernel Methods — Support Vector Learning.* MIT Press, 1999.

[74] F. Poulet. Multi-way Distributed SVM algorithms. In *Parallel and Distributed computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03),* Cavtat-Dubrovnik, Croatia, September 2003.

[75] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks, 2002.

[76] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[77] rkosta and shakelford. Eclipse profiler plugin. Sourceforge: http://sourceforge.net/projects/eclipsecolorer, March 2002.

[78] Martin Schoeberl. Jop: A java optimized processor. In Workshop on Java Technologies for Real-Time and Embedded Systems, LNCS 2889, Catania, Italy, November 2003.

[79] B. Schölkopf, P. L. Bartlett, A. Smola, and R. Williamson. Shrinking the tube: a new support vector regression algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 330 – 336, Cambridge, MA, 1999. MIT Press.

[80] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms, 2000.

[81] B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection.

[82] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels.* MIT Press, 2002.

[83] W. M. Shaw(Jr.), R. Burgin, and P. Howell. Performance standards and evaluations in ir test collections: cluster-based retrieval models. *Information Processing and Management,* 33:1:1–14, 1997.

[84] Ashok Srivastava and Julienne Stroeve. Onboard detection of snow, ice, clouds and other geophysical processes using kernel methods. In *Proceedings of the ICML 2003 Workshop on Machine Learning Technologies for Autonomous Space Sciences,* 2003.

[85] Ingo Steinwart. On the optimal parameter choice for ν-support vector machines. Technical report, University Jena, 2002.

[86] Ingo Steinwart. Sparseness of support vector machines. *J. Mach. Learn. Res.,* 4:1071–1105, 2003.

[87] Salvatore J. Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Dave W. Fan, and Philip K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Knowledge Discovery and Data Mining,* pages 74–81, 1997.

[88] Alexander Strehl and Joydeep Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR),* 3:583–617, December 2002.

[89] Systronix. Jstamp web site, www.jstamp.com.

[90] Wolfgang Theilmann and Kurt Rothermel. Disseminating mobile agents for distributed information filtering. In *Proceedings of the First International Symposium on Agent Systems and*

Applications Third International Symposium on Mobile Agents, page 152. IEEE Computer Society, 1999.

[91] K. Tumer and J. Ghosh. *Robust Order Statistics Based Ensembles for Distributed Data Mining*. AAAI Press, 2000.

[92] Kagan Tumer and Joydeep Ghosh. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348, Feb. 1996.

[93] Vladimir Naumovich Vapnik. *The Nature of Statistical Learning Theory*. Springer, NY, 1995.

[94] Alfred D. M. Wan and Peter J. Braspenning. The Bifurcation of DAI and Adaptivism as Synthesis. In *Proceedings of the 1995 Dutch Conference on AI (NAIC)*, pages 253–262, 1995.

[95] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.

[96] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Mach. Learn.*, 38(3):257–286, 2000.

[97] R. Wirth, M. Borth, and J. Hipp. When Distribution is Part of the Semantics: A New Problem Class for Distributed Knowledge Discovery. In *Proceedings of PKDD-2001 Workshop on Ubiquitous Data Mining for Mobile and Distributed Environments*, pages 56–64, Freiburg, Germany, September 2001.

[98] Candace A. York. Ibm's advanced pvc technology laboratory. Web, June 2001. cyork@us.ibm.com.

[99] Feng Zhao and Leonidas J. Guibas. *Wireless Sensor Networks*. Morgan Kaufmann, 2004.

[100] S. Zhong and J. Ghosh. A unified framework for model-based clustering. *JMLR*, 4:1001–1037, 2003.

[101] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.