UNIVERSITY OF CRETE

Computer Science Department

P H D T H E S I S

Defended by

Kallirroe FLOURI

Distributed Algorithms for Support Vector Machine training in Wireless Sensor Networks

Thesis Advisor: Panagiotis TSAKALIDES

November, 2009
Heraklion

This dissertation is part of the 03ED69 research project, implemented within the framework of the "Reinforcement Program of Human Research Manpower" (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development - General Secretariat of Research and Technology and 75% from E.U. - European Social Fund).

Computer Science Department
University of Crete

Distributed Algorithms For Distributed Support Vector Machine Training

Dissertation submitted by
Kallirroe Flouri
as partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Author: _____

Supervisor: _____
Associate Professor
Panagiotis Tsakalides
Department of Computer Science,
University of Crete

Advisory Committee Member: _____
Professor
Apostolos Traganitis
Department of Computer Science,
University of Crete

Advisory Committee Member: _____
Assistant Professor
Maria G. Papadopouli
Department of Computer Science,
University of Crete

Examination Committee Member: _____
Associate Professor
Antonis Argyros
Department of Computer Science,
University of Crete

Examination Committee Member: _____
Professor
George A. Tsihrintzis
Department of Informatics
University of Piraeus

Examination Committee Member: _____
Associate Professor
Baltasar Beferull-Lozano
Department of Computer Science,
University of Valencia

Examination Committee Member: _____
Lecturer
Alin Achim
Department of Electrical and
Electronic Engineering,
University of Bristol

Chairman of the Department of Computer Science: _____
Professor
Dimitris Plexousakis
Department of Computer Science,
University of Crete

Heraklion, November 2009

Abstract

Advancements in low-power electronic devices integrated with wireless communication capabilities and sensors have opened up an exciting new field in computer science. Wireless sensor networks (WSN) can be developed at a relatively low-cost and can be deployed in a variety of different settings. The emergence of WSNs, has brought a significant interest towards decentralized detection, estimation and classification for use in monitoring, surveillance, location sensing, and distributed learning applications.

Processing sensor data locally requires considerably less energy than communicating it to a distant node, yielding an interesting communication-computation tradeoff. To reduce global communication requirements, one needs to perform signal processing to extract key information in a distributed fashion and without losing fidelity. In this dissertation, we focus on the distributed training of Support Vector Machine (SVM) classifiers by taking advantage of the sparseness representation of their decision boundary determined only by a small subset of the training samples, the so-called support vectors.

First, we present two incremental algorithms for distributed SVM training where the updates of the classifier are diffused sequentially in the network. We show that after a single complete pass through all the clusters, a good approximation of the optimal separating hyperplane is achieved. Then, we present two gossip-based algorithms, that are robust to unexpected failures of nodes and to changes in the network topology. In the first scheme, each sensor updates its hyperplane at every iteration by combining its support vectors with the support vectors communicated by the neighbors, resulting in a close-to-optimal efficient distributed scheme. In the second approach, the information exchanged between sensors describes uniquely and completely the convex hulls of the two classes. This approach guarantees convergence to the optimal classifier at the expense of increased complexity associated with the construction of the convex hulls.

Finally, by exploiting notions of convex optimization and duality theory, we derive a novel mathematical characterization for the sparse representation of the most important measurements that neighboring sensors should exchange in order to reach an agreement to the optimal linear classifier. We propose a function which ranks the training vectors in order of importance in the learning process. The amount of information to be exchanged is controlled by a user defined threshold, depending on the desired tradeoff between classification accuracy and power consumption. We prove that a threshold value exists for which the proposed algorithm converges to the optimal classifier. We explore the effect of the network topology to the convergence and we study the performance through simulation experiments.

Περίληψη

Η πρόοδος της μικροηλεκτρονικής και των υλικών επέτρεψε την κατασκευή πολύ μικρών αισθητήρων διευρύνοντας τους ορίζοντες για έρευνα στις περιοχές των τηλεπικοινωνιών και της πληροφορικής. Τα ασύρματα δίκτυα αισθητήρων είναι μια συνεχώς αναπτυσσόμενη, χαμηλού πλέον κόστους τεχνολογία, η οποία έχει αλλάξει τον τρόπο με τον οποίο ο άνθρωπος αλληλεπιδρά με το περιβάλλον. Οι απαιτήσεις αυτών των δικτύων έχουν οδηγήσει στην ανάγκη για ανάπτυξη κατανεμημένων αλγορίθμων για την ανίχνευση σημάτων, την εκτίμηση παραμέτρων, και την ταξινόμηση δεδομένων σε πλειάδα εφαρμογών όπως η περιβαλλοντική παρακολούθηση και ο προσδιορισμός της θέσης αντικειμένων.

Συνήθως, η τοπική επεξεργασία δεδομένων στους αισθητήρες απαιτεί χαμηλότερα ποσά ενέργειας από ό,τι θα καταναλώνονταν εάν όλοι οι αισθητήρες μετέδιδαν ασύρματα τα δεδομένα τους σε έναν κεντρικό κόμβο. Η ελαχιστοποίηση της κατανάλωσης ενέργειας σε ένα δίκτυο αισθητήρων είναι μείζονος σημασίας, επομένως απαιτείται τόσο η τοπική επεξεργασία δεδομένων όσο και η ανάπτυξη απλών αποδοτικών αλγορίθμων συνάθροισης και συμπίεσης, καθώς οι κόμβοι χαρακτηρίζονται από μικρή υπολογιστική δύναμη και περιορισμένη μνήμη. Στη παρούσα διδακτορική διατριβή μελετούμε και σχεδιάζουμε κατανεμημένους αλγόριθμους για την εκπαίδευση ταξινομητών Μηχανών Εδραίων Διανυσμάτων (SVMs), εκμεταλλευόμενοι την ιδιότητά τους ότι η επιφάνεια απόφασης κατασκευάζεται από ένα πολύ μικρό υποσύνολο των δεδομένων, τα λεγόμενα εδραία διανύσματα.

Αρχικά, παρουσιάζουμε δύο αυξητικούς αλγόριθμους για κατανεμημένη εκπαίδευση ενός SVM, στους οποίους η ανανέωση της εκτίμησης του υπερεπιπέδου απόφασης πραγματοποιείται διαδοχικά στους κόμβους του δικτύου. Δείχνουμε με εκτενείς προσομοιώσεις ότι μόνο με ένα πέρασμα από κάθε κόμβο, καταλήγουμε σε μια καλή εκτίμηση του βέλτιστου ταξινομητή. Έπειτα, παρουσιάζουμε δύο επαναληπτικούς αλγόριθμους που βασίζονται σε τεχνικές "Gossip", ώστε να αποφεύγονται προβλήματα που προκαλούνται από μη προβλεπόμενες αλλαγές στην τοπολογία του δικτύου. Ο πρώτος προτεινόμενος αλγόριθμος επιτρέπει σε κάθε επανάληψη την ανανέωση του ταξινομητή με την διάδοση μόνο των εδραίων διανυσμάτων μέσω γειτονικών κόμβων, συγκλίνοντας σε μια προσέγγιση του βέλτιστου ταξινομητή. Σύμφωνα με τον δεύτερο αλγόριθμο, οι γειτονικοί κόμβοι ανταλλάσσουν τα δεδομένα τα οποία χαρακτηρίζουν μοναδικά τα κυρτά περιγράμματα που δημιουργούνται σε κάθε κόμβο. Ο αλγόριθμος αυτός εγγυάται σύγκλιση στον βέλτιστο ταξινομητή, με κόστος την αυξημένη πολυπλοκότητα που απαιτείται για τον προσδιορισμό των κυρτών περιγραμμάτων.

Τέλος, θεωρούμε την εκπαίδευση ενός SVM ως ένα πρόβλημα βελτιστοποίησης υπό περιορισμούς και χρησιμοποιούμε στοιχεία από τη δυαδική θεωρία για να προτείνουμε ένα συστηματικό τρόπο για την βέλτιστη επιλογή των δεδομένων που πρέπει να ανταλλάσσουν οι κόμβοι του δικτύου, ώστε να επέλθει

κοινή σύγκλιση στη βέλτιστη λύση. Προτείνουμε μια συνάρτηση η οποία ταξινομεί τα δεδομένα κατα σειρά σπουδαιότητας στην διαδικασία εκμάθησης. Το πλήθος των δεδομένων που ανταλλάσουν οι γειτονικοί κόμβοι καθορίζεται από ένα κατώφλι, ανάλογα με την επιθυμητή εξισορρόπηση ανάμεσα στην ακρίβεια της ταξινόμησης και την καταναλισκόμενη ενέργεια. Αποδεικνύουμε ότι υπάρχει ένα κατώφλι, ώστε ο προτεινόμενος αλγόριθμος να εγγυάται σύγκλιση στο βέλτιστο ταξινομητή. Ερευνούμε την επιρροή της τοπολογίας του δικτύου στην ταχύτητα σύγκλισης του αλγόριθμου και μελετούμε την απόδοση του αλγορίθμου με εκτεταμένες προσομοιώσεις.

Acknowledgments

Once upon a time...
I had just finished my master's degree, but my journey in the "olive-tree country" was not yet fulfilled. A journey full of surprises, excitement, uncertainty, tears, joy and laughter. I was not alone, and I feel the urge to thank some people who walked beside me these four years...

First, I would like to thank the Institute of Computer Science of the Foundation for Research and Technology-Hellas, that provided the required facilities and equipment during my studies. I have spent more hours in the Institute, than in my house! Many thanks are owed to the Computer Science Department where my goal became true, and also the General Secretariat of Research and Technology of the Greek Ministry of Development, which provided the essential funding for this research.

I would like to thank my supervisor, professor Panagiotis Tsakalides who over the last four years continuously led and inspired me to new research horizons. Also many thanks are owed to the advisory committee members professors Apostolos Traganitis and Maria Papadopouli, and the examination committee members professors Antonis Argyros, George A. Tsihrintis, Baltasar Beferull-Lozano, and Dr. Alin Achim, for their comments and helpful advise. Special thanks to Baltasar Beferull-Lozano for a great collaboration during my phd studies.

Many friends and colleagues have supported and encouraged me during this journey through all these years. Some of them are still here, others have already chosen a different path. I would like to thank my dear friends George Dimitriou (also known as "G"), Manos Moschous and Giannis Moustakas (so-called "Jmous") -alphabetical order-, who where, are and will be there for me for good or bad, (hopefully) for a very long time. I thank all the colleagues of the Telecommunications and Networks Laboratory and especially thanks but also congratulations to my *roomates* in the *Cage*: Stefpap, Aggelakis, Chariton and Npapas, who put up with me all these years! I could not forget Maria, Despoina and Anastasia- friendships that begun during my phd studies but will be continued. I also owe a big thanks to Vassilis Lekakis - we shared many hours playing the *wii* with Elias. Elias Raftopoulos deserves more than anyone my unlimited love and gratitude as he is always near me during the last years. The best is yet to come in Zurich!

At this point, I want to deeply thank (again) Panagiotis Tsakalides. He was the one who motivated me through all these years. He showed faith in me, and gave me the opportunity to make my dreams come true. He was the leader in this long journey. "Tough ain't enough", give your best, no Matter the Cost or the constraints. Thank you for everything!

From the bottom of my heart, I would like to thank my family for the inexhaustible support and encouragement. Nothing would be possible without the help of my father, mother and sister. Thank you so much!
...and they lived happily ever after...

Contents

List of Figures

Chapter 1

Introduction

Contents

1.1 Wireless Sensor Networks: Description and challenges

Recent advances in the micro-sensor and semiconductor technology have opened a new field for computer science. The electronic miniaturization and the advances in the semiconductor manufacturing process enable for low-power and low-cost hardware. Small and smart devices equipped with a processing unit, storage capacity, and small radios for wireless communication provide new application opportunities. Augmented with different kinds of sensors, *e.g.*, for temperature, pressure, light, humidity, movement, etc., such sensor devices can be deployed to observe physical phenomena both accurately and reliably.

The dense deployment of hundreds or even thousands of sensor nodes that comprise a Wireless Sensor Network (WSN) facilitates a wide range of applications covering habitat monitoring, environmental monitoring, health care, structural health monitoring, security and surveillance. Sensors are usually very small in size, have micro-sensing capabilities, they can process, communicate and exchange data, and all these ideally with a low energy cost. Therefore, they can be placed close to the phenomenon one wants to study. Since they are deployed in unattended areas, human interaction is needed only for the set up of the network. Ideally, WSNs nodes are only once deployed but can be used for many experiments.

Any protocol used in a WSN should be power-aware and be designed by considering energy efficiency. In contrast to traditional networks, the forwarding of data packets is not address-based but data-centric. That is, sensor nodes are not addressed by a globally unique identifier but rather based on data attributes. For example, instead of requesting the temperature value of an individual node, an application may be more interested if there are any nodes which detect a temperature above a given threshold or what is the average temperature in a specific region. This paradigm shift also implies that a WSN will likely be tailored for the sensing application task. Application-specific data forwarding may reduce significantly the amount of information that need to be transmitted. By in-network

processing and data aggregation, redundant and useless data can already be filtered out along the forwarding path. For example, consider the scenario where a sink node is interested in the average temperature each node measures over a certain period of time. Rather than forwarding the data readings individually, they can be aggregated by intermediate nodes along the path without any loss of information by only transmitting the sum and the number of readings.

A sensor network can be of great benefit when used in areas where dense monitoring and analysis of complex phenomena over a large region is required for a long period of time. The design criteria and requirements of sensor networks differ from application to application. Some typical requirements are:

- Scalability: As sensor nodes get cheaper and cheaper, it is highly likely that sensor networks will consist of a huge number of nodes. Algorithms for sensor networks must therefore scale well with thousands and tens of thousands of sensor nodes.
- Adaptiveness: All protocols should be able to adapt to changes in the environment, *e.g.*, changes concerning the connectivity or changes concerning the sensing of physical phenomena.
- Resistance to failures: Due to the low-cost hardware or outside influences, sensor nodes are prone to failure. However, achieving the common application task should not be affected. In case of node failures, the network must be able to re-organize itself and if needed change assigned application tasks.
- Self-organization: Since a WSN is usually deployed in an unattended area, the network must operate without the need of manual configuration. For example, communication paths throughout the network should be established automatically. Also the cooperation between nodes must be organized in an unattended manner in order to achieve the global application task.
- Energy efficiency: As most sensor nodes are restricted concerning their energy capacity, all protocols and algorithms must be energy-efficient and save as much energy as possible. Since most energy is consumed during wireless communication, the radio must be turned off most of the time. But also the transmission of data should be energy-efficient in order to minimize the number of sent and received packets.
- Simplicity: Besides their energy capacity, sensor nodes are also limited in their processing and storage capabilities. Thus, algorithms should be as simple as possible in order to minimize their computational complexity and memory usage.

In the following, interesting real life applications of WSNs are presented for habitat monitoring, environmental monitoring, and military applications along with the vision of WSNs.

1.1.1 Applications of Wireless Sensor Networks

Recent advances in sensor network research allow for small and cheap sensor nodes which can obtain a lot of data about physical parameters, *e.g.*, temperature, humidity, lightning condition, pressure, noise level, carbon dioxide level, oxygen level, soil makeup, soil moisture and magnetic field. Sensing devices can also extract characteristics of objects such as speed, direction, and size, deduce the presence or absence of certain kinds of objects, and measure all kinds of values about machinery, *e.g.*, mechanical stress level or movement. This huge choice of options allows to use sensor networks in a number of scenarios, *e.g.*, habitat and environment monitoring, health care, military surveillance, industrial machinery

surveillance, home automation, as well as smart and interactive places [2]. The application of a sensor network usually determines the design of the sensor nodes and the design of the network itself. No general architecture for sensor networks exists at the moment. In this Section, some examples of habitat monitoring, environmental monitoring, military and health care applications are presented.

Habitat monitoring: The main objective of *habitat monitoring* is to track and observe wild life. In the past, habitat monitoring has been done by researchers hiding and observing the wild life using cameras and microphones. However, this technique is intrusive and uncomfortable, and long term observations are difficult and expensive. Usually, live data is not available. Sensor networks offer a better way for habitat monitoring. The sensor network technology is less intrusive than any other technique. Thus, the wild life is less affected, resulting in better research results. Also, long-term observations are possible and sensor networks can be designed in a way that live data is available on the Internet. Human interaction is usually needed only for setup of the sensor network and for removal of the sensors after the end of the observation. Hence, sensor networks help to reduce the costs of habitat monitoring research projects.

The Great Duck Island project [3] was one of the first applications of sensor networks in habitat monitoring research. The main objective of the research project was to monitor the micro climates (*e.g.*, temperature and humidity) in and around nesting burrows used by the Leach's Storm Petrel. The great advantage of this sensor network compared to standard habitat monitoring was its non-intrusive and non-disruptive nature. The project is named after a small island at the coast of Maine, Great Duck Island, where the research took place. At first, a network of 32 sensor nodes was deployed (see Figure 1.1). The sensor network platform consisted of processor radio boards commonly referred to as motes. They were manually placed in the nesting burrows by researchers. The sensor nodes periodically sent their sensor readings to a base station and got back to sleep mode. The base station used a satellite link to offer access to real-time data over the Internet. To get information about the micro climate in the nesting burrows, the sensor nodes collected data about temperature, humidity, barometric pressure and mid-range infrared. Between spring 2002 and November 2002, over 1 million sensor readings were logged from the sensor network. In June 2003, a larger sensor network, consisting of 56 nodes, was deployed, and it was extended in July 2003 by 49 additional sensor nodes and again augmented by 60 more sensor nodes and 25 weather station nodes in August 2003. Hence, the network consisted of more than 100 sensor nodes at the end of 2003. The network used multi-hop routing from the nodes to the base station. The software of the sensor nodes was based on the sensor network operating system TinyOS [4]. The sensor network of the Great Duck Island project was pre-configured and did not self configure, *e.g.*, each sensor node got assigned a unique network layer address during compilation of the code prior to deployment.

Environmental monitoring: While habitat monitoring deals with observation of animals and their surroundings, the task of *environmental monitoring* is to sense the state of the environment. Sensor networks in environment monitoring are extremely helpful in research and they allow exploration of areas that were not accessible up to date such as habitat monitoring, structural health monitoring, etc. With a sensor network consisting of a large number of nodes, it is possible to explore the macrocosmos in a much better way because it is easier to obtain data of natural phenomena at many different places. Hence, better prediction models can be build. Other applications of environmental monitoring include structural monitoring, which ensures the integrity of bridges, buildings, and other man-made structures. In the following, an environment monitoring research project is presented.

Figure 1.1: Placement of sensor nodes in the Great Duck Island project: sensor nodes were placed in the nesting burrows of storm petrels (1) and outside of the burrow (2). Sensor readings are relayed to a base station (3), which transmits them to a laptop in the research station (4), that sends it via satellite (5) to a lab in California. Image source: http://www.wired.com/wired/archive/11.12/network.html

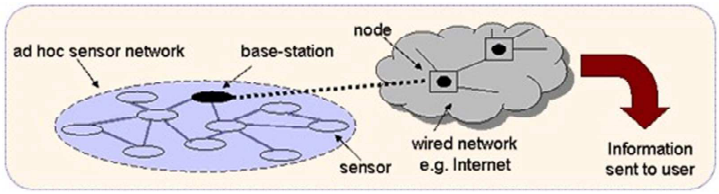


Figure 1.2: This diagram shows how this sensor web works. Image source: http://radio.weblogs. com/0105910/2004/05/31.html

Figure 1.3: A sensor node of GlacsWeb. Image source: http://radio.weblogs. com/0105910/2004/05/31.html

Glaciers are very important in climate research because climate changes can be identified by observing size and movement of the glaciers. GlacsWeb [5] is a sensor network for monitoring glaciers. A sample network was installed at Briksdalsbreen, Norway, in 2004 (see Figure 1.2). The aim of GlacsWeb is to understand glacier dynamics in response to climate change. Sensor nodes (see Figure 1.3) were deployed into the ice and on the till. A base station was installed on the surface of the glacier. The sensor network was designed to acquire data about weather conditions and the GPS position at the base station from the surface of the glacier. The sensor nodes used a simple duty cycle. Every sensor node sampled every 4 hours temperature, strain (due to stress from the ice), pressure (if immersed

Figure 1.4: MICA2 Mote. Image source: http://computer.howstuffworks.com/mote.htm

in water), orientation (in 3 dimensions), resistivity (to determine if the sensor is sitting in sediment till, water, or ice) and battery voltage. Thus, there were 6 sets of readings for each sensor node every day. The sensor nodes directly communicated with the base station once a day at a fixed time. All sensor nodes were queried during a 5 minutes interval each day. The base station recorded its location once a week using GPS. Obtaining the GPS position took 10 minutes. During those 10 minutes, remote administration of the base station was also possible. After this, the base station sent all its readings to a reference station PC via long range radio. The radio range of the sensor nodes was significantly reduced in ice to less than 40 m while it is 500 meters in air. The sensor nodes had a clock drift of 2 seconds a day. The base station synchronized them daily. The base station was equipped with a solar panel which failed when the panels were covered with snow. Future versions of the base station will also have a wind turbine.

Military applications: Sensor networks in *military applications* are often used for surveillance missions. The focus of surveillance missions is to collect or verify as much information as possible about the enemy's capabilities and about the positions of hostile targets. Sensor networks are used to replace soldiers because surveillance missions often involve high risks and require a high degree of stealthiness. Hence, the ability to deploy unmanned surveillance missions, by using wireless sensor networks, is of great practical importance for the military.

EnviroTrack is a middleware for tracking of objects. In [6] the design and implementation of a system for energy-efficient surveillance based on EnviroTrack is described. The main objective of the system is to track the positions of moving vehicles in an energy-efficient and stealthy manner. For the prototype, a network consisting of 70 sensor nodes was used. The sensor nodes are based on MICA2 Motes (see Figure 1.4). They are equipped with dual-axis magnetometers. Issues in the design of the network were long network lifetime, adjustable sensitivity, stealthiness, and effectiveness. To prolong the network lifetime, it is important to use as little energy as possible. The system allows for a trade-off between energy consumption of the sensor nodes and the accuracy of the tracking. To save energy, only a subset of nodes, the so-called sentries, are active at a given time. The sentries monitor events. When an event occurs, the sentry nodes awake the other sensor nodes of the network and form groups for collaborate tracking. Hence, in the absence of an event, most sensor nodes are in a sleep mode, using only very little energy. The sensor nodes can adjust the sensitivity of their sensors to adapt to different terrains and security requirements. To achieve stealthiness, zero communication exposure is desired in the absence of significant events. The sensor network can only be used in challenging military scenarios if the tracking is effective. Especially, the estimated location of a moving object must be precise enough

and the event must be communicated as fast and reliable as possible to the base station. As the collaborative detection and tracking process relies on the spatio-temporal correlation between the tracking reports sent by multiple sensor motes, it is important that the sensor nodes are timesynchronized and that the positions of the sensor nodes are known.

Health applications: In health care, especially in hospitals, a lot of sensors are used today to monitor the vital signs and states of patients 24 hours a day. Most of those sensors are wired, so patients are often severely restricted in their mobility. Hence, the use of wireless sensors would result in a great gain in life quality for the patients and it even would give them more security, because an automated system may react fast on emergency situations. Another issue in health care is the seamless transfer of patients between first aid personal, emergency rescuers and doctors at a hospital, so that a seamless monitoring and data transfer can take place. Sensor networks can solve a lot of problems in health care scenarios. The most important design criteria for applications in health care are security and reliability.

CodeBlue [7] is a system to enhance emergency medical care with the goal to have a seamless patient transfer between a disaster area and a hospital. Wearable vital sign sensors (body sensors) are used to track a patient's status and location. They also operated as active tags, which enhance first responders ability to assess patients on the scene, ensure seamless transfer of data among emergency personal, and enable an efficient allocation of hospital resources. Current body sensors are able to obtain heart rate, oxygen saturation, end-tidal CO2, and serum chemistries measurements. Figure 1.5 shows a typical body sensor node of the CodeBlue project. The network of CodeBlue does not rely on any infrastructure but is formed ad hoc. It is intended to span a large disaster area or a whole building, *e.g.*, a hospital. The system is designed to scale with a very dense network consisting of lots of nodes. The nodes of the network are heterogeneous and range from simple body sensors and PDAs to PCs. CodeBlue addresses data discovery and data naming, robust routing, prioritisation of critical data, security, and tracking of device locations. For data discovery, a publish/subscribe model is used. Body sensors publish data and devices used by nurses or doctors, *e.g.*, a PDA, can subscribe to that data stream. CodeBlue also uses data filtering and data aggregation. For example, a doctor may be interested in the full data stream of one patient and wants only to be notified if some other patients are in an unusual or critical state. The CodeBlue project uses a best-effort security model: if an external authority can be reached, strong guarantees are needed to access data. However, if no external authorities are available because of poor connectivity or infrastructure loss, weaker guarantees may be used. This situation may arise for example in a disaster area where it is not possible to spend time setting up an infrastructure, keying in passwords or exchanging keys.

1.1.2 Types of applications

Many of these applications share some basic characteristics. In most of them, there is a clear difference between sources of data and the actual nodes that sense data, sinks and nodes where the data should be delivered to. These sinks sometimes are part of the sensor network itself; sometimes they are clearly systems "outside" the network. Also, there are usually, but not always, more sources than sinks and the sink is oblivious or not interested in the identity of the sources; the data itself is much more important. The interaction patterns between sources and sinks show some typical patterns. The most relevant ones are:

Event detection: Sensor nodes should report to the sink(s) once they have detected the occurrence of a specified event. The simplest events can be detected locally by a single

Figure 1.5: A body sensor node of the CodeBlue Project which is build into a wristband. Image source: http://www.eecs.harvard.edu/ mdw/proj/codeblue/.

sensor node in isolation (*e.g.* a temperature threshold is exceeded); more complicated types of events require the collaboration of nearby or even remote sensors to decide whether a (composite) event has occurred (*e.g.* a temperature gradient becomes too steep). If several different events can occur, event classification might be an additional issue.

Periodic measurements: Sensors can be tasked with periodically reporting measured values. Often, these reports can be triggered by a detected event; the reporting period is application dependent.

Function approximation and edge detection: The way a physical value like temperature changes from one place to another can be regarded as a function of location. A WSN can be used to approximate this unknown function (to extract its spatial characteristics), using a limited number of samples taken at each individual sensor node. This approximate mapping should be made available at the sink. How and when to update this mapping depends on the application's needs, as do the approximation accuracy and the inherent trade-off against energy consumption. Similarly, a relevant problem can be to find areas or points of the same given value. An example is to find the isothermal points in a forest fire application to detect the border of the actual fire. This can be generalized to finding "edges" in such functions or to sending messages along the boundaries of patterns in both space and/or time [8].

Tracking: The source of an event can be mobile (*e.g.* an intruder in surveillance scenarios). The WSN can be used to report updates on the event source's position to the sink(s), potentially with estimates about speed and direction as well. To do so, typically sensor nodes have to cooperate before updates can be reported to the sink.

These interactions can be scoped both in time and in space (reporting events only within a given time span, only from certain areas, and so on). These requirements can also change dynamically overtime; sinks have to have a means to inform the sensors of their requirements at runtime. Moreover, these interactions can take place only for one specific request of a sink (so-called "one-shot queries"), or they could be long-lasting relationships

between many sensors and many sinks.

1.1.3 Vision of Future Sensor Networks

Sensor networks facilitate efficient solutions for applications that were not possible in the past. This is especially true for applications that require dense monitoring and analysis of complex phenomena covering a large region and lasting for a long time. Meteorologic research is one of the research fields that deals with complex phenomena. Even today, we still do not know a lot about the climate of earth. Hence, more sensor network applications in meteorology research can be expected for the near future. Sensor networks are also good to advance into areas that have limited accessibility. The oceans are one example. A better knowledge of the oceans will result in a better understanding of the climate because the oceans are an important factor, *e.g.* for the emergence of hurricanes. However, to explore the oceans with sensor networks, research into underwater communication and sensor design for underwater missions is necessary. In the long run, sensor networks may be of good use in space-related research. For example instead of sending one single sensor system like the Mars Rover to a distant planet, it would perhaps be possible in the future to deploy a sensor network consisting of thousands of nodes. As the system would consist of a lot of nodes, a total failure of the overall system would be not very likely. Thus, the risk of a failure of a mission would be smaller.

The vision of WSNs is to make human life easier by connecting the physical world. Future sensor networks are envisioned to be ubiquitous, large-scale and interconnected, and to evolve in the so-called World Wide Sensor Network (WWSNs). Currently, most of the WSNs are working as isolated islands. Without sharing sensor data across different domains, the most important features of ubiquitous computing (*e.g.*, context awareness) will not be easily achieved. For sharing among WWSN, the first nut to crack is to interconnect different WSNs which are spatially deployed in different locations with IP based Internet; the second one is to integrate them into a single WWSN over the Internet for publishing, sharing and searching of sensor data.

Moreover, an interesting research area where sensors and social networks can fruitfully interface, from sensors providing contextual information in context-aware and personalized social applications, to using social networks as "storage infrastructures" for sensor information has recently gained attention. Sensors provide information about various aspects of the real world. Online social networks on the other hand, provide insights into the communication links and patterns between people. They have enabled novel developments in communications as well as transforming the Web from a technical infrastructure to a social platform, the so-called Social Web. By combining Social Network and sensors, applications can provide an extension of social activities through sensors, as user activity is modelled not by voluntary user input, but can be automatically generated by sensors. Hence it enhances the idea of ubiquitous social networking, that can be observed on micro-blogging services such as Twitter, where some people tend to publish simple updates containing only their current location as GPS coordinate.

1.2 Motivation and Research Question

Although the application of a sensor network usually determines the design of the sensor nodes and the design of the network itself, there is one common requirement for all applications: energy efficiency. As most sensor nodes will be restricted concerning their energy capacity, all protocols and algorithms must be energy-efficient. Processing sensor

data locally requires considerably less energy than communicating it to a distant node, yielding an interesting communication/computation trade-off. To reduce global communication requirements, one needs to perform signal processing to extract key information in a distributed fashion and without losing fidelity.

One of the most important tasks to be performed in a wireless sensor network (WSN), is classification, that is, it is important to infer whether the samples measured by sensors in a WSN belong to a certain hypothesis (class) or not. It is well known that Support Vector Machines (SVMs) have been successfully used as classification tools in a variety of areas [9, 10, 11]. Training a SVM calls for solving a quadratic programming (QP) problem in a number of coefficients equal to the number of training examples. An appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane is specified via real-valued weights on the training samples. Training samples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Only training samples that lie close to the decision boundary between the two classes, the so-called support vectors, receive non-zero weights. In fact, since their design allows the number of support vectors to be small compared to the total number of training samples, they provide a compact representation of the data, to which new examples can be added as they become available. Therefore SVMs seem well suited to be trained in a distributed fashion.

Our goal is to be able to train a SVM in an efficient and distributed fashion so that: a) we can get good classification results on test data and b) our algorithms can be used easily in the context of WSN, where the training must take place across sensors. The research questions that we answer in this thesis are the following:

- Which sensor protocol is suitable for in-network information exchange? Ideally, in a sensor network we would like all sensors to have the same estimate of the classifier, so that each sensor would be trained to classify any new measurement.
- What kind of data should neighboring sensors exchange in order to get high classification accuracy but with low energy consumption? WSN nodes should exchange a sufficient amount of data in order to ensure or approximate optimality. On the other hand, the more data is exchanged, the more energy is consumed.

1.2.1 Contribution

This dissertation work comprises a study of distributed optimization techniques for in-network data processing so as to eliminate the need to transmit raw data to a central point. SVM gained our attention as it is considered a very popular classification tool in the literature, an interesting convex optimization problem, and a topic to be studied in a distributed aspect for applications to sensor networks. Taking advantage of the sparse representation that SVMs provide for the decision boundaries, we designed classes of incremental and gossip-based distributed consensus algorithms for training the classifier.

The first class of the algorithms are two energy efficient algorithms that involve a distributed incremental learning for the training of a SVM in a WSN both for stationary and non-stationary data. The key idea behind our proposed incremental algorithm was that as the number of support vectors is typically very small compared to the number of training samples, the data of previous clusters can be compressed to their corresponding estimated hyperplane (support vectors and offset) and forwarded to the next cluster. We showed that after a single complete pass through all the clusters, a good approximation of the optimal separating plane is achieved, that is, the separating hyperplane is very close to the one obtained using a centralized, energy-consuming algorithm, where all the sample data is used at once in a single training step at the base station.

With an eye to the non-stationary environments, we designed an alternative incremental algorithm. In many real world applications, the concept of interest (definition of classes to be separated) may be time-varying or space-varying. Consequently, these changes make the model built on old data inconsistent with the new data, hence regular updating of the model is necessary. This problem, known as concept drift, complicates the task of learning in SVM. A typical example of this phenomenon is weather prediction, where the rules may vary radically depending on the season. On the other hand, one may also observe changes in the training data, which have no correspondence to controllable parameters of the experiment [12]. For example, in engineering applications, the quality of a machine deteriorates over the course of its life-cycle. Therefore, there is a need to have a robust system that can adapt easily to these uncontrollable changes. In the case of distributed sequential training of a SVM in a WSN, this effect is even more accentuated. In order to address this problem, we modified the previous algorithm for non-stationary data. We showed that our proposed algorithms are much more efficient in terms of energy cost, since they reduce the energy spend up to 50% of the energy spend in the centralized case.

In all incremental techniques, the update of the estimate is diffused sequentially in the network and the convergence to the global estimate is reached at the final step of the algorithm. Hence, at each time slot only one node has the updated critical information and consequently the optimal estimate. In this case, the trained SVM classifier is constructed at the final step of the algorithm. However, nodes in a WSN, usually operate in environments that are prone to link and node failure. Hence, it is important to design algorithms that are robust to unexpected failures of nodes and consequently to changes in the topology. Thus, to maximize robustness, all nodes should ideally achieve convergence to the same optimal estimate.

To that goal, we designed gossip-based distributed consensus algorithms for training a SVM. Opposed to incremental algorithms, gossip-based approaches rely on communication with one-hop neighbors only, to develop iterative algorithms that eventually converge to the desired estimate. After some iterations, all sensors reach consensus to the optimal solution. Sensors keep refining their estimate concurrently at each time slot in order to reach finally convergence (consensus) to a common global estimate. We propose two gossip-based algorithms: the first provides a suboptimal solution but with the minimum energy consumption and the second guarantees convergence to the optimal solution while communicates more data, hence more energy is consumed.

Concluding the thesis, we provide a mathematical characterization for the sparse representation of the most important measurements that neighboring nodes should exchange in order to reach an agreement near the optimal SVM classifier. We introduce a selection function which ranks each training vector in order of importance. Therefore, the amount of information exchange can vary allowing for a desired trade-off between classification accuracy and power consumption. We investigate this trade-off for linearly and non linearly separable data sets and for high dimensionality measurements.

Concluding, the main contributions of this thesis are:

- The design of two incremental algorithms for distributed training of SVM in the context of a WSN for stationary and non stationary data.
- The design of two gossip-based distributed algorithms for the consensus of the network to an estimate of the optimal SVM classifier of linearly and non linearly separable data sets.
- The mathematical characterization of the required partial information that neighboring sensor nodes should exchange in order to achieve consensus in the network, while

minimizing the number of transmissions.

- The design of the corresponding gossip algorithm achieving global consensus with minimal inter-node network communication.

This thesis is structured as follows. Chapter 2 first provides a general introduction to the concept of convex optimization problems, investigating distributed optimization problems with emphasis on applications of sensor networks. Then, a class of distributed algorithms (gossip algorithms), motivated by the needs of ad hoc and wireless sensor networks, is presented. Finally, the problem of centralized SVM training is reviewed, both for linearly separable and linearly inseparable classes. Chapter 3 presents the proposed incremental distributed algorithm for SVM training for stationary and non-stationary data, along with a set of experimental results. Chapter 4 describes the proposed gossip-based distributed consensus approach for SVM training in the context of WSNs. Linearly and non linearly data sets are used in order to test performance in terms of classification accuracy and energy efficiency. Chapter 5 concludes our research with a mathematical analysis for the characterization for the sparse representation of the most important measurements that neighboring nodes should exchange in order to reach an agreement near the optimal SVM classifier. Experimental results are presented using a variety of data sets in several network topologies. Finally, in Chapters 6 and 7, we present the conclusions and future work directions, respectively.

Chapter 2

Background Theory

Contents

With the advent of wireless sensor networks, there has been a growing interest towards decentralized detection, estimation and classification algorithms for use in monitoring, surveillance, location sensing and distributed learning applications. Moreover, the development of visual sensor networking technology will require efficient distributed processing for automated event detection and classification. Indeed, many of the signal processing problems that occur in WSN applications can be viewed as distributed optimization problems. In one of the first related studies, Nowak et. al presented an incremental algorithm for the robust optimization of a cost function of interest, applied to the source localization, clustering, and density estimation problems [13]. More recent studies of distributed optimization have mainly focused on estimating simple functions of the data, analyzing issues such as convergence criteria and convergence rate [14, 15]. Moreover in [16], power consumption has been taken into account for the algorithmic design and the minimum required amount of communication power has been studied.

An important class of distributed algorithms employ the so-called gossip techniques. They seem well suited in the context of a WSN, since gossip techniques are robust to changes in the topology of the network in case of node failures: neighboring sensors can exchange data, and hence the information is diffused in the network. They are based on successive refinement of local estimates maintained at individual sensors. Gossip-based approaches rely on communication with one-hop neighbors only, to develop iterative algorithms that eventually converge to the desired estimate. After some iterations, all sensors reach *consensus* to the optimal solution. The notion of consensus averaging for the estimation of deterministic unknown parameters using linear data models was introduced in [1] whereby

each sensor updates its local estimate by appropriate weighting the estimates of its neighbors. A more elaborate approach entailing distributed computation of the sample average estimator with the aid of dual decomposition techniques was studied in [17]. For distributed estimation of a Gaussian random parameter in a scalar linear model, Baraniuk et al. applied the Jacobi iteration [18]. The same scalar linear model in a dynamical system was also considered in [19]. More recently, a consensus-based distributed expectation-maximization algorithm was proposed in [20] for density estimation and classification. Finally, in [21], a linear iterative strategy is developed that enables a subset of the nodes to calculate any given function of the node values.

Support Vector Machines constitute a modern classification tool, that has been successfully applied to a number of applications ranging from face recognition and text categorization to engine knock detection, bioinformatics and database marketing [22, 23, 24]. Training involves optimization of a convex cost function meaning that there are no false local minima to complicate the learning process. SVMs are the most well-known of a class of algorithms that use the idea of kernel substitution and which are broadly referred to as kernel methods. SVMs can also construct linear classification functions with good theoretical and practical generalization properties even in very high-dimensional attribute spaces. The major advantage of linear classifiers is their simplicity and low complexity.

In general, pattern classification algorithms assume that all the features are available centrally during the construction of the classifier and its subsequent use. But in many practical situations, data are recorded in different geographical locations by sensors, each observing features of local interest and having a partial view of the data.

The outline of this Chapter is the following. In Section 2.1 we briefly discuss why WSNs are different from traditional networks and present the design challenges of a WSN. To realize these requirements, innovative mechanisms for a communication network have to be found, as well as new architectures, and protocol concepts. In Section 2.2 we present some of the mechanisms that will form typical parts of WSNs. We describe some fundamental sensor protocols and we analytically describe the so-called *gossip algorithms*. Next, in Section 2.3, we provide the necessary background on convex optimization, the notion of duality, and the KKT conditions for primal-dual optimality. We then present an interesting distributed optimization approach of the well-known steepest decent algorithm, for the purpose of a WSN scenario, where the entire data set is not available, [13]. Finally, in Section 2.4 we view the problem of centralized Support Vector Machine (SVM) training as a convex optimization problem, both for linearly separable and linearly inseparable classes and we briefly describe some interesting applications that made SVMs so popular in WSNs. We also introduce the notion of Distributed SVM training in WSN applications, and we briefly present some interesting works on this topic.

2.1 Why Are Sensor Networks Different?

Sensors existed long before the emergence of new sensing technologies. The simplest sensors have been in operation for decades, *e.g.*, thermostats that adjust interior air-conditioning and heating. They were expensive, bulky and big storage units. Due to their size, human intervention was necessary for their deployment, or the replacement of their batteries.

The increasing miniaturization of radio frequency (RF) devices, micro-electromechanical systems (MEMS) and the advances in wireless technologies, have generated a great deal of research interest in the area of WSNs. WSNs nowadays, employ a large number of miniature autonomous devices known as sensor nodes to form the network without the aid of any established infrastructure. In a wireless sensor system,

the individual nodes are capable of sensing their environments, processing the information locally, or sending it to one or more collection points through a wireless link. Each node has a short-range transmission due to low RF transmit power. Short-range transmission minimizes the possibility of the transmitted signals being eavesdropped; also, it helps in prolonging the lifetime of the battery. In some sensor system applications, the nodes are hard to reach and it is impossible to replace their batteries. In other applications, the nodes must operate without battery replacement for a long time. Such conditions make the system power consumption a very crucial parameter.

2.1.1 Wireless Sensor Networks vs. Ad hoc Networks

WSNs use ad hoc topology because of its ease of deployment and decreased dependence on infrastructure. Although WSNs use an ad-hoc architecture, this architecture is different from that of a conventional wireless ad hoc networks. A WSN is comprised of thousands of sensors whose batteries are often irreplaceable. The data rate is low but with high redundancy. On the other hand, a conventional wireless ad hoc network is comprised of a smaller number of nodes with replaceable batteries. The data rate in this network is high but with low redundancy. The simplest example of an ad hoc network is perhaps a set of computers connected together via cables to form a small network, like a few laptops in a meeting room. In this example, the aspect of self-configuration is crucial - the network is expected to work without manual management or configuration.

Usually, however, the notion of a mobile ad hoc network (MANET) is associated with wireless communication and specifically wireless multihop communication; also, the name indicates the mobility of participating nodes as a typical ingredient. Examples for such networks are disaster relief operations, *e.g.,* firefighters communicating with each other, or networks in difficult locations like large construction sites, where the deployment of wireless infrastructure (access points etc.), let alone cables, is not a feasible option. In such networks, the individual nodes together form a network that relays packets between nodes to extend the reach of a single node, allowing the network to span larger geographical areas than would be possible with direct sender- receiver communication. The two basic challenges in a MANET are the reorganization of the network as nodes move about and handling the problems of the limited reach of wireless communication. Literature on MANETs that summarize these problems and their solutions abound, as these networks are still a very active field of research; popular books include [25, 26, 27]. These general problems are shared between MANETs and WSNs. Nonetheless, there are some principal differences between the two concepts, warranting a distinction between them and demanding separate research efforts for each one.

These general problems are shared between MANETs and WSNs. Nonetheless, there are some principal differences between the two concepts, warranting a distinction between them and regarding separate research efforts for each one.

Applications and equipment: MANETs are associated with somewhat different applications as well as different user equipment than WSNs: in a MANET, the terminal can be fairly powerful (a laptop or a PDA) with a comparably large battery. This equipment is needed because in typical MANET applications, there is usually a human in the loop: the MANET is used for voice communication between two distant peers, or it is used for access to a remote infrastructure like a Web server. Therefore, the equipment has to be powerful enough to support these applications.

Application specific: Owing to the large number of conceivable combinations of sensing, computing, and communication technology, many different application scenarios for WSNs become possible. It is unlikely that there will be a "one-size-fits-all" solution

for all these potentially very different possibilities. As one example, WSNs are conceivable with very different network densities, from very sparse to very dense deployments, which will require different or at least adaptive protocols. This diversity, although present, is not quite as large in MANETs.

Environment interaction: Since WSNs have to interact with the environment, their traffic characteristics can be expected to be very different from other, human-driven forms of networks. A typical consequence is that WSNs are likely to exhibit very low data rates over a large timescale, but can have very bursty traffic when something happens (a phenomenon known from real-time systems as "*event showers*" or "*alarm storms*"). Long periods (days) of inactivity can alternate with short periods (seconds or minutes) of very high activity in the network, pushing its capacity to the limits. MANETs, on the other hand, are used to support more conventional applications (Web, voice, and so on) with their comparably well understood traffic characteristics.

Scale: Potentially, WSNs have to scale to much larger numbers (thousands or perhaps hundreds of thousands) of entities than current ad hoc networks, requiring different, more scalable solutions. As a concrete case in point, endowing sensor nodes with a unique identifier is costly (either at production or at runtime) and might be an overhead that could be avoided. Hence, protocols that work without such identifiers might become important in WSNs, whereas it is fair to assume such identifiers exist in MANET nodes.

Energy: In both WSNs and MANETs, energy is a scarce resource. But WSNs have tighter requirements on network lifetime, and recharging or replacing WSN node batteries is much less an option than in MANETs. Owing to this, the impact of energy considerations on the entire system architecture is much deeper in WSNs than in MANETs.

Self configurability: Similar to ad hoc networks, WSNs will most likely be required to selfconfigure into connected networks, but the difference in traffic, energy trade-offs, and so forth, could require new solutions. Nevertheless, it is in this respect that MANETs and WSNs are probably most similar.

Dependability and QoS: The requirements regarding dependability and QoS are quite different. In a MANET, each individual node should be fairly reliable; in a WSN, an individual node is next to irrelevant. The quality of service issues in a MANET are dictated by traditional applications (low jitter for voice applications, for example); for WSNs, entirely new QoS concepts are required, which also take energy explicitly into account.

Data centric: Redundant deployment will make data-centric protocols attractive in WSNs. This concept is alien to MANETs. Unless applications like file sharing are used in MANETs, which do bear some resemblance to data centric approaches, data-centric protocols are irrelevant to MANETs, but these applications do not represent the typically envisioned use case.

Simplicity and resource scarceness: Since sensor nodes are simple and energy supply is scarce, the operating and networking software must be kept orders of magnitude simpler compared to today's desktop computers. This simplicity may also require breaking with conventional layering rules for networking software, since layering abstractions typically cost time and space. Also, resources like memory, which is relevant for comparably heavy-weight routing protocols as those used in MANETs, is not available in arbitrary quantities, requiring new, scalable, resource-efficient solutions.

Mobility: The mobility problem in MANETs is caused by nodes moving around, changing multihop routes in the network that have to be handled. In a WSN, this problem can also exist if the sensor nodes are mobile in the given application. There are two additional aspects of mobility to be considered in WSNs. First, the sensor network can be used to detect and observe a physical phenomenon (in the intrusion detection applications, for

example). This phenomenon is the cause of events that happen in the network (like raising of alarms) and can also cause some local processing, for example, determining whether there really is an intruder. What happens if this phenomenon moves about? Ideally, data that has been gathered at one place should be available at the next one. Also, in tracking applications, it is the explicit task of the network to ensure that some form of activity happens in nodes that surround the phenomenon under observation. Second, the sinks of information in the network (nodes where information should be delivered to) can be mobile as well. In principle, this is no different than node mobility in the general MANET sense, but can cause some difficulties for protocols that operate efficiently in fully static scenarios. Here, carefully observing trade-offs is necessary. Furthermore, in both MANET and WSNs, mobility can be correlated, *e.g.,* a group of nodes moving in a related, similar fashion. This correlation can be caused in a MANET by, for example, belonging to a group of people traveling together. In a WSN, the movement of nodes can be correlated because nodes are jointly carried by a storm, a river, or some other fluid.

In summary, there are commonalities, but the fact that: a) WSNs have to support very different applications; b) they have to interact with the physical environment; and c) they have to carefully mediate various trade-offs, justifies WSNs as a system concept distinct from traditional networks.

2.1.2 Challenges and design issues for WSNs

Although he design criteria and the requirements of sensor networks differ from application to application, nonetheless certain common traits appear, especially with respect to the characteristics and the required mechanisms of such systems. Realizing these characteristics with new mechanisms is the major challenge of the vision of wireless sensor networks. The following characteristics are shared among most of the application examples.

Type of service: The service type rendered by a conventional communication network is evident (moving bits from one place to another. For a WSN, moving bits is only a means to an end, but not the actual purpose. Rather, a WSN is expected to provide meaningful information and/or actions about a given task: "People want answers, not numbers" [28]. Additionally, concepts like scoping of interactions to specific geographic regions or to time intervals will become important. Hence, new paradigms of using such a network are required, along with new interfaces and new ways of thinking about the service of a network.

Quality of Service: Closely related to the type of a network's service is the quality of that service. Traditional quality of service requirements, *e.g.,* bounded delay or minimum bandwidth in multimedia, are irrelevant when applications are tolerant to latency [29], or the bandwidth of the transmitted data is very small in the first place. In some cases, only occasional delivery of a packet can be more than enough; in other cases, very high reliability requirements exist. In yet other cases, delay is important when actuators are to be controlled in a real time fashion by the sensor network. The packet delivery ratio is an insufficient metric; what is relevant is the amount and quality of information that can be extracted at given sinks about the observed objects or area. Therefore, adapted quality concepts like reliable detection of events or the approximation quality of a temperature map for example, are important.

Fault tolerance: Since nodes may run out of energy or might be damaged, or since the wireless communication between two nodes can be permanently interrupted, it is important that the WSN as a whole is able to tolerate such faults. To tolerate node failure, redundant deployment is necessary, using more nodes than would be strictly necessary if all nodes functioned correctly.

Lifetime: In many scenarios, nodes will have to rely on a limited supply of energy (using batteries). Replacing these energy sources in the field is usually not practicable, and simultaneously, a WSN must operate at least for a given mission time or as long as possible. Hence, the lifetime of a WSN becomes a very important figure of merit. Evidently, an energy-efficient way of operation of the WSN is necessary. As an alternative or supplement to energy supplies, a limited power source (via power sources like solar cells, for example) might also be available on a sensor node. Typically, these sources are not powerful enough to ensure continuous operation but can provide some recharging of batteries. Under such conditions, the lifetime of the network should ideally be infinite. The lifetime of a network also has direct trade-offs against quality of service: investing more energy can increase quality but decrease lifetime. Concepts to balance these trade-offs are required. The precise definition of lifetime depends on the application at hand. A simple option is to use the time until the first node fails (or runs out of energy) as the network lifetime. Other options include the time until the network is disconnected in two or more partitions, the time until 50% (or some other fixed ratio) of nodes have failed, or the time when for the first time a point in the observed region is no longer covered by at least a single sensor node (when using redundant deployment, it is possible and beneficial to have each point in space covered by several sensor nodes initially).

Scalability: Since a WSN might include a large number of nodes, the employed architectures and protocols must be able scale to these numbers.

Wide range of densities: In a WSN, the density of the network can vary considerably. Different applications will have very different node densities. Even within a given application, density can vary over time and space because nodes fail or move; the density also does not have to be homogeneous in the entire network (because of imperfect deployment, for example) and the network should adapt to such variations.

Programmability: Not only will it be necessary for the nodes to process information, but also they will have to react flexibly on changes in their tasks. These nodes should be programmable, and their programming must be changeable during operation when new tasks become important. A fixed way of information processing is insufficient.

Maintainability: As both the environment of a WSN and the WSN itself change (depleted batteries, failing nodes, new tasks), the system has to adapt. It has to monitor its own health and status to change operational parameters or to choose different trade-offs (*e.g.,* to provide lower quality when energy resources become scarce). In this sense, the network has to maintain itself; it could also be able to interact with external maintenance mechanisms to ensure its extended operation at a required quality [30].

2.2 Required Mechanisms

In the previous Section we discussed about the evolution of sensor nodes from bulky, big storage units to very small and cheap devices. The emergence of sensing technologies and the fact that the architecture and the applications of WSNs are different from those of conventional networks, enables separate research efforts and new mechanisms for designing a WSN.

To realize the requirements of WSNs, innovative mechanisms for a communication network have to be found, as well as new architectures, and protocol concepts. A particular challenge is the need to find mechanisms that are sufficiently specific to the idiosyncrasies of a given application to support the specific quality of service, lifetime, and maintainability requirements [31]. On the other hand, these mechanisms also have to generalize to a wider range of applications lest a complete "from scratch" development and implementation of a

WSN becomes necessary for every individual application. This would likely render WSNs as a technological concept economically infeasible. Some of the mechanisms that will form typical parts of WSNs are:

Multihop wireless communication: While wireless communication will be a core technique, a direct communication between a sender and a receiver is faced with limitations. In particular, communication over long distances is only possible using prohibitively high transmission power. The use of intermediate nodes as relays can reduce the total required power. Hence, for many forms of WSNs, so-called multihop communication will be a necessary ingredient.

Energy-efficient operation: To support long lifetimes, energy-efficient operation is a key technique. Options to look into include energy-efficient data transport between two nodes or, more importantly, the energy-efficient determination of a requested information. Also, nonhomogeneous energy consumption, the forming of "hotspots", is an issue.

Auto-configuration: A WSN will have to configure most of its operational parameters autonomously, independent of external configuration. The sheer number of nodes and simplified deployment will require that capability in most applications. As an example, nodes should be able to determine their geographical positions only using other nodes of the network. Also, the network should be able to tolerate failing nodes (because of a depleted battery, for example) or to integrate new nodes (because of incremental deployment after failure, for example).

Collaboration and in-network processing: In some applications, a single sensor is not able to decide whether an event has happened but several sensors have to collaborate to detect an event and only the joint data of many sensors provides enough information. Information is processed in the network itself in various forms to achieve this collaboration, as opposed to having every node transmit all data to an external network and process it "at the edge" of the network. An example is to determine the highest or the average temperature within an area and to report that value to a sink. To solve such tasks efficiently, readings from individual sensors can be aggregated as they propagate through the network, reducing the amount of data to be transmitted and hence improving the energy efficiency. How to perform such aggregation is an open question.

Data centric: Traditional communication networks are typically centered around the transfer of data between two specific devices, each equipped with (at least) one network address. The operation of such networks is thus *address-centric*. In a WSN, where nodes are typically deployed redundantly to protect against node failures or to compensate for the low quality of a single node's actual sensing equipment, the identity of the particular node supplying data becomes irrelevant. What is important are the answers and values themselves, not which node has provided them. Hence, switching from an address-centric paradigm to a *data-centric* paradigm in designing architecture and communication protocols is promising. An example for such a data-centric interaction would be to request the average temperature in a given location area, as opposed to requiring temperature readings from individual nodes. Such a data-centric paradigm can also be used to set conditions for alerts or events ("raise an alarm if temperature exceeds a threshold"). In this sense, the data-centric approach is closely related to query concepts known from databases; it also combines well with collaboration, in-network processing, and aggregation.

Locality: Rather a design guideline than a proper mechanism, the principle of locality will have to be embraced extensively to ensure, in particular, scalability. Nodes, which are very limited in resources like memory, should attempt to limit the state that they accumulate during protocol processing to only information about their direct neighbors. The hope is that this will allow the network to scale to large numbers of nodes without having to rely on powerful processing at each single node. How to combine the locality principle with efficient protocol designs is still an open research topic, however.

Exploit trade-offs: Similar to the locality principle, WSNs will have to rely to a large

degree on exploiting various inherent trade-offs between mutually contradictory goals, both during system/protocol design and at runtime. Examples for such trade-offs have been mentioned already: higher energy expenditure allows higher result accuracy, or a longer lifetime of the entire network trade-offs against lifetime of individual nodes. Another important trade-off is node density: depending on application, deployment, and node failures at runtime, the density of the network can change considerably. Therefore, the protocols will have to handle very different situations, possibly present at different places of a single network.

2.2.1 Sensor Protocols for Information Exchange

Protocols and architectures for communication among sensor nodes is a very crucial topic in WSNs. It has been given a lot of attention, since they might differ depending on the application and network architecture. One can find more details and analytic classification of sensor protocols in the literature [32, 33, 34, 35]. In this Section, we review two main protocols for information exchange between sensor nodes, flooding and routing. We then describe an alternative flooding approach, the so-called *gossip algorithms*.

Routing in WSNs: Routing in wireless networks has been an active research area for many years. Routing techniques rooted in computer data communications have been thoroughly explored for use in wireless networks, resulting in the emergence of many self-organizing, self-healing models in commercial implementations.

The reason for all this activity is that robust operation within changing propagation conditions and under energy and communication bandwidth constraints precludes the use of traditional IP-based protocols and creates a difficult challenge for dedicated WSN routing algorithms. The task of finding and maintaining routes in WSNs is nontrivial because energy restrictions and sudden changes in node status (including failure, jamming, or temporary obstructions) cause frequent and unpredictable changes. Building and propagating automatic routing through the network requires powerful node processors, large amounts of memory, and additional dedicated routers, as well as network downtime until alternative routing is established.

Determining routing tables is the task of the routing algorithm with the help of the routing protocol. In wired networks, these protocols are usually based on link state or distance vector algorithms (Dijkstra's or Bellman-Ford). In a wireless, possibly mobile, multihop network, different approaches are required. Routing protocols here should be distributed, have low overhead, be self-configuring, and be able to cope with frequently changing network topologies. Building and maintaining routing tables with alternate routing (for responding to changing propagation conditions) while using low cost, low power processors proves to be a formidable challenge, which is amplified when the size and number of hops increase.

Many new and sophisticated algorithms have been proposed to resolve these issues. The resulting routing schemes take into consideration the inherent features of WSNs along with application and architecture requirements. To minimize energy consumption, routing techniques employ some interesting techniques special to WSNs, such as data aggregation and in-network processing, clustering, different node role assignment, and data-centric methods.

These routing techniques seek balance between simple solutions with limited robustness and sophisticated solutions. Even in sophisticated solutions, there is still the risk that in large networks or when messages are short, the routing overhead will consume valuable resources such as bandwidth and power and sometimes cause packet collisions. Worst case, these factors combine to finally degrade network robustness, throughput, and end-to-end delay.

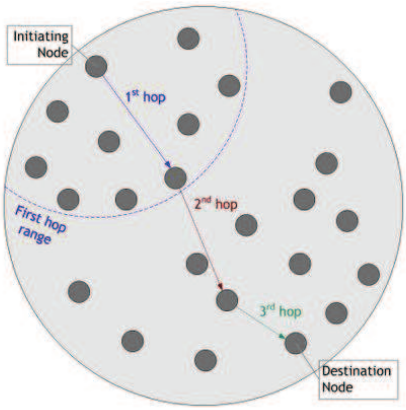


Figure 2.1: Signal propagation in networks based on routing, http://www.industrial-embedded.com/articles/id/?4098.

One basic routing attribute related to the dynamic nature of an RF environment has yet to be solved: One moment after the routing table is created, it is already obsolete because the RF conditions have changed.

Flooding the network: The simplest forwarding rule is to flood the network: Send an incoming packet to all neighbors. As long as source and destination node are in the same connected component of the network, the message is sure to arrive at the destination. To avoid message exchange endlessly, a node should only forward those messages that has not yet seen. In flooding, instead of using a specific route for sending a message from one node to another, the message is sent to all the nodes in the network, including those to whom it was not intended.

The attractiveness of the flooding technology lies in its high reliability and utter simplicity. There is no need for sophisticated routing techniques since there is no routing. No routing means no network management, no need for self-discovery, no need for self-repair, and, because the message is the payload, no overhead for conveying routing tables or routing information.

Flooding technology has additional advantages related to propagation. Signals arriving at each node through several propagation paths benefit from the inherent space diversity, thus maximizing the network robustness of handling obstructions, interferences, and resistance to multipath fading, with practically no single point of failure. In other words, blocking one path or even a limited number of paths is usually of no consequence.

Figures 2.1 and 2.2 exemplify the different propagation patterns for the two mesh technologies in the same 24-node, three-hop network. The first hop signal propagation is blue, the second is purple, and the third is green. In Figure 2.1, the effect of a signal obstruction or interference in the route on the left precludes the signal from arriving at its destination. Sophisticated routing-based schemes identify the problem and try to reroute the signal. If no alternative route helps, a new routing is recalculated, leading to side effects such as latency and possible service interruption until the new route it completed, verified, and propagated. In the flooding-based scheme in Figure 2.2, a signal obstruction will most likely not affect the operation at all because of the numerous redundant paths.

An alternative to forwarding data to all neighbors is to forward it to an arbitrary one. Such gossiping results in the messages randomly traversing the network in the hope of eventually finding the destination node.

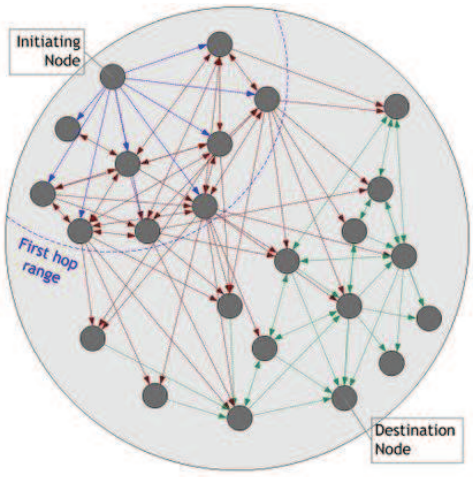


Figure 2.2: Signal propagation in networks based on flooding, http://www.industrial-embedded.com/articles/id/?4098.

Gossiping in WSNs: Gossiping [36] is an alternative to the classic flooding approach that uses randomization to conserve energy. The goal is to spread updates to all nodes as fast as possible while minimizing the message overhead. The question is to select neighbors for gossiping the rumor at hand (how often, which neighbors, etc.). Gossip algorithms essentially consist of information propagation through nodes randomly selecting neighbors to transmit to in each round. If a gossiping node receives data from a given neighbor, it can forward data back to that neighbor if it randomly selects that neighbor. There has been considerable work on constructing gossip algorithms to compute aggregate functions in networks [37, 38]. In comparison to other approaches, gossip-based approaches are more limited in that the functions considered in most cases are limited to averages, sums and extremal values. The advantage however is fault tolerance as well as simplicity in implementation; the computational operations that nodes have to perform are restricted to very simple ones. Gossip algorithms are reviewed in Section 2.2.2.

2.2.2 Gossip Algorithms

In this Section, we review distributed asynchronous algorithms, also known as gossip algorithms, for computation and information exchange in an arbitrarily connected network of nodes. Nodes in such networks operate under limited computational, communication and energy resources. These constraints naturally give rise to "gossip" algorithms: schemes which distribute the computational burden and in which a node communicates with a randomly chosen neighbor.

Gossip algorithms are distributed message-passing schemes designed to disseminate and process information over wireless sensor and ad-hoc networks. They have received significant interest because the problem of computing a global function of data distributively over a network, using only localized message-passing, is fundamental for numerous applications.

Ad-hoc networks, such as sensor networks, peer-to-peer networks and mobile networks are not deliberately designed with an "infrastructure". Sensor networks, for example, are formed by randomly deployed sensors in a geographic area in order to sense or monitor environment, surveillance or order applications. In such networks, nodes need to collect,

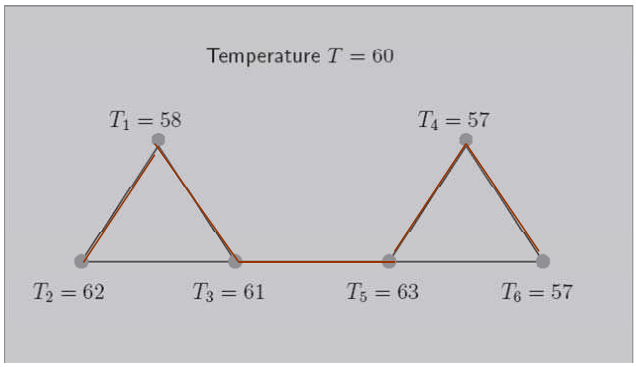


Figure 2.3: The red line depicts the infrastructure for computing in a distributed manner the average temperature by exchanging the temperature values in an orderly fashion.

process and communicate information over a wireless channel. Nodes in such networks do not have access to addressing or routing information. They also have limited energy and computation resources, therefore nodes may hibernate or leave the network or die. Since the global topology of the network is not available to the nodes, they only have access to local information or information of the neighboring sensors. Therefore, algorithms deployed in such networks need to be completely distributed, robust against node failure and changes in topology. These constraints have motivated the design of gossip algorithms: schemes which distribute the computational burden and in which a node communicates with a randomly chosen neighbor.

The simplest setup is the following: n nodes are placed on a graph whose edges correspond to reliable communication links. Each node is initially given a scalar (which could correspond to some sensor measurement like temperature) and we are interested in solving the distributed averaging problem: namely, to find a distributed message-passing algorithm by which all nodes can compute the average of all n scalars. A scheme that computes the average can easily be modified to compute any linear function of the measurements as well as more general functions. Furthermore, the scalars can be replaced with vectors and generalized to address problems like distributed filtering and optimization as well as distributed detection in sensor networks [39, 40, 41].

A toy example described to motivate the averaging problem is sensing the temperature of some small region of space using a sensor network, [42]. Sensors are deployed to measure the temperature T of a source. Sensor i, measures $T_i = T+n_i$, where the n_i are independent, identically distributed (i.i.d), zero mean Gaussian sensor noise variables. Consider the case of 6 sensors *i.e.*, $i = 1, \ldots, 6$. The unbiased, minimum mean squared error (MMSE) estimate is the average $\hat{T} = \sum_{i=1}^{6} T_i/6$. Thus, to combat minor fluctuations in the ambient temperature and the noise in sensor readings, the nodes need to average their readings. The average temperature can be calculated by interchanging the values in an orderly fashion for a specific infrastructure, Figure 2.3. In case of a node failure, the infrastructure and hence the algorithm needs to be re-computed. Since the algorithm needs to be robust for this kind of applications, sensors can "gossip" with their neighbors without having knowledge of the topology of the network. Therefore, a node contacts one of its neighbors and forms a pair, Figure 2.4. Paired nodes average their current estimated and after some communication the estimate of each node converges to the average.

Distributed averaging can be done in many ways. One straightforward method is flood-

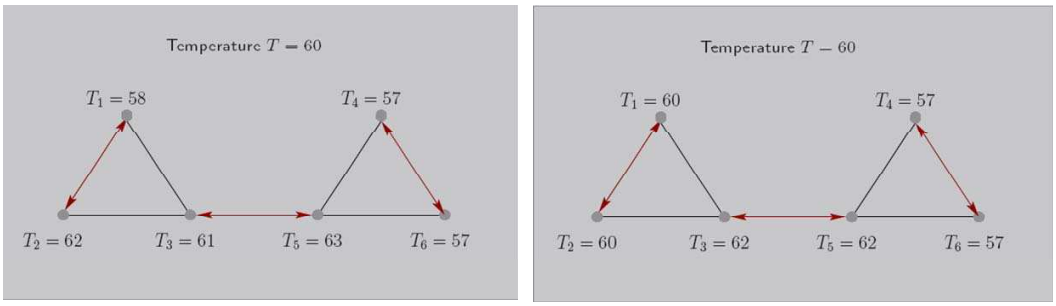


Figure 2.4: Averaging using gossip algorithms.

ing. Each node maintains a table of the initial node values of all the nodes, initialized with its own node value only. At each step, the nodes exchange information from their own tables and the tables of their neighbors. After a number of steps equal to the diameter of the network, every node knows all the initial values of all the nodes, so the average (or any other function of the initial node values) can be computed.

Another interesting approach described in [1], considers distributed linear iterations. More specifically, consider a network (connected graph) $G = (N, E)$ consisting of a set of nodes $N = \{1, \ldots, n\}$ and a set of edges E, where each edge $\{i, j\} \in E$ is an unordered pair of distinct nodes. The set of neighbors of node i is denoted $N_i = \{j | \{i, j\} \in E\}$. Each node i holds an initial scalar value $x_i(0) \in \mathbb{R}$, and $\mathbf{x}(0) = (x_1(0) \ldots, x_n(0))$ denotes the vector of the initial node values on the network. The network gives the allowed communication between nodes: two nodes can communicate with each other if and only if they are neighbors. The goal is to compute the average of the initial values, $(1/n) \sum_{i=1}^{n} x_i(0)$, via a distributed algorithm, in which nodes only communicate with their neighbors. The distributed linear iterations have the form

$$x_i(t+1) = W_{ii} x_i(t) + \sum_{j \in N_i} W_{ij} x_j(t), \qquad i = 1, \ldots, n \tag{2.1}$$

where $t = 0, 1, 2, \ldots$ is the discrete time index, and W_{ij} is the weight on x_j at node i. Setting $W_{ij} = 0$ for $j \notin N_i$, this iteration can be written in vector form as

$$\mathbf{x}(t+1) = \mathbf{W}\mathbf{x}(t). \tag{2.2}$$

The constraint on the sparsity pattern of the matrix $\mathbf{W}$ can be expressed as $\mathbf{W} \in S$, where

$$S = \{\mathbf{W} \in \mathbb{R}^{nxn} | W_{ij} = 0 \ \text{ if } \{i, j\} \notin E\}. \tag{2.3}$$

The linear iteration (2.2) implies that $\mathbf{x}(t) = \mathbf{W}^t x(0)$ for $t = 0, 1, 2 \ldots$. Weight matrix $\mathbf{W}$ is chosen so that for any initial value $\mathbf{x}(0)$, $\mathbf{x}(t)$ converges to the average vector

$$\bar{x} = (\mathbf{1}^T \mathbf{x}(0)/n)\mathbf{1} = (\mathbf{1}\mathbf{1}^T/n)\mathbf{x}(0), \tag{2.4}$$

i.e.,

$$\lim_{t \to \infty} \mathbf{x}(t) = \lim_{t \to \infty} \mathbf{W}^T x(0) = (\mathbf{1}\mathbf{1}^T/n)\mathbf{x}(0). \tag{2.5}$$

Here $\mathbf{1}$ denotes the vector with all coefficients one. This is equivalent to the matrix equation

$$\lim_{t \to \infty} W^t = \lim_{t \to} \frac{\mathbf{1}\mathbf{1}^T}{n}. \tag{2.6}$$

The asymptotic convergence factor is defined as

$$r_{asym}(\mathbf{W}) = \sup_{\mathbf{x}(0)\neq\bar{x}} \lim_{t\to\infty} \left(\frac{\| \mathbf{x}(t) - \bar{x} \|_2}{\| \mathbf{x}(0) - \bar{x} \|_2} \right)^{1/t}, \tag{2.7}$$

and the associated convergence time

$$\tau_{asym} = \frac{1}{\log(1/r_{asym})}, \tag{2.8}$$

which gives the (asymptotic) number of steps for the error to decrease by the factor $1/e$. It is obvious that as the difference of the value of each node from the average value increases, so does the convergence time of the linear iterations, Figure 2.5.

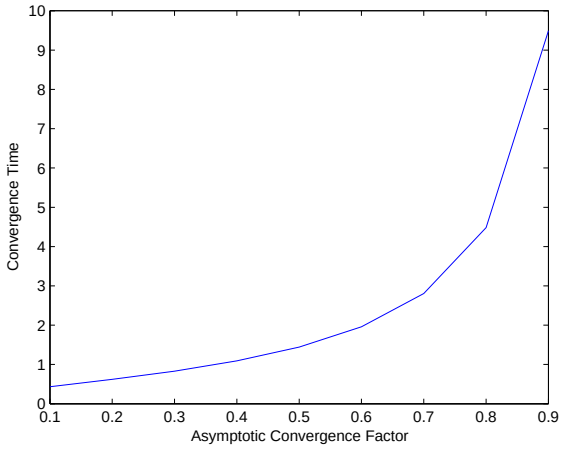


Figure 2.5: The convergence time of the distributed linear iterations as a function of the asymptotic convergence factor.

2.2.2.1 Convergence Conditions

The distributed linear iteration (2.1) converges to the average, *i.e.* equation (2.6) holds, for any initial vector $\mathbf{x}(0) \in \mathbb{R}$ if and only if

$$\lim_{t\to\infty} \mathbf{W}^t = \frac{\mathbf{1}\mathbf{1}^T}{n}. \tag{2.9}$$

The necessary and sufficient conditions for this matrix equation to hold are the following,

$$\mathbf{1}\mathbf{W} = \mathbf{1}^T, \tag{2.10}$$

$$\mathbf{W}\mathbf{1} = \mathbf{1}, \tag{2.11}$$

$$\rho(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}) < \mathbf{1}, \tag{2.12}$$

where $\rho(\cdot)$ denotes the spectral radius of a matrix. Moreover,

$$r_{asym} = \rho(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}), \tag{2.13}$$

The above theorem is proved in [1]. Here, we provide some interpretations.

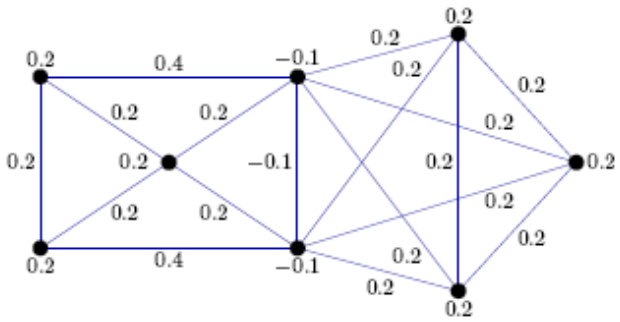


Figure 2.6: A small graph with 8 nodes and 17 edges. Each edge and node is labeled with the optimal symmetric weights, [1].

- Equation (2.10) states that $\mathbf{1}$ is a left eigenvector of W associated with the eigenvalue one. This condition implies that $\mathbf{1}\mathbf{x}(t+1) = \mathbf{1}^T\mathbf{x}(t)$ for all t, *i.e.,* the sum (and therefore the average) of the vector of node values is preserved at each step.
- Equation (2.11) states that $\mathbf{1}$ is also a right eigenvector of $\mathbf{W}$ associated with the eigenvalue one. This condition means that $\mathbf{1}$ (or any vector with constant entries) is a fixed point of the linear iteration (2.1).
- Together with the first two conditions, condition (2.12) means that one is a simple eigenvalue of $\mathbf{W}$, and that all other eigenvalues are strictly less than one in magnitude.
- If the elements of $\mathbf{W}$ are nonnegative, then (2.11) and (2.12) state that $\mathbf{W}$ is doubly stochastic, and (2.13) states that the associated Markov chain is irreducible and aperiodic.

Concluding, the averaging time of a gossip algorithm depends on the spectral radius of the weight matrix W and consequently on the second largest eigenvalue of this matrix. Hence, the smaller the eigenvalue, the faster the averaging problem.

Boyd also presented in [1] an example of a network that consist of 8 nodes and 17 edges, shown in Figure 2.6. The histogram of the eigenvalues of the weight matrix is depicted in Figure 2.7. The second largest eigenvalue is 0.6 and the largest eigenvalue is of course 1, as we demanded in conditions (2.10), (2.11).

Now assume that each sensor measures the temperature of the environment. Due to minor fluctuations in the temperature and the noise in sensor readings, the nodes do not have exactly the same data measured. So, they need to average their data. Data are generated using gaussian distribution with mean value 30 and standard deviation 5. The mean value of the measurements of each node is 29.5 and it is depicted with the straight line in Figure 2.9. The blue dots depict the value of one sensor at each iteration. One can easily notice that after only a few iterations, the value of one sensor converges to the average value of the initial value of the temperature as it was measured from all sensors.

Moreover, Boyd in [1] proposed some heuristics based on the Laplacian in order to choose $\mathbf{W}$ that guarantees convergence of the distributed linear averaging iterations. Figure 2.8 depicts the histogram of the eigenvalues of another matrix with constant edge weights. The second largest eigenvalue is 0.65, which is greater than eigenvalue 0.6, of the first weight matrix, thus we expect a slower convergence. This is depicted comparing Figures 2.9 and 2.10 and it is confirmed by equation (2.8) since $\tau_{asym}(0.65) = 2.36 > 1.95 = \tau_{asym}(0.6)$.

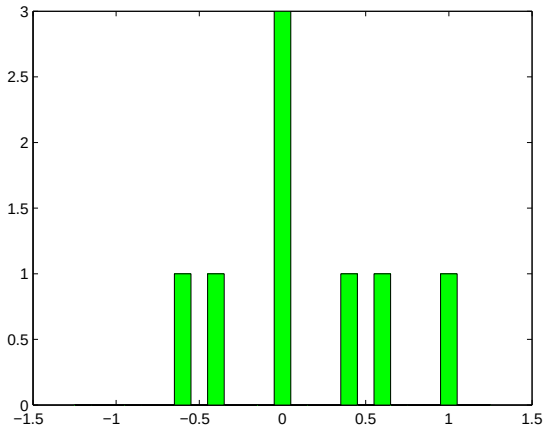


Figure 2.7: The histogram of the eigenvalues of the weight matrix of the above graph.

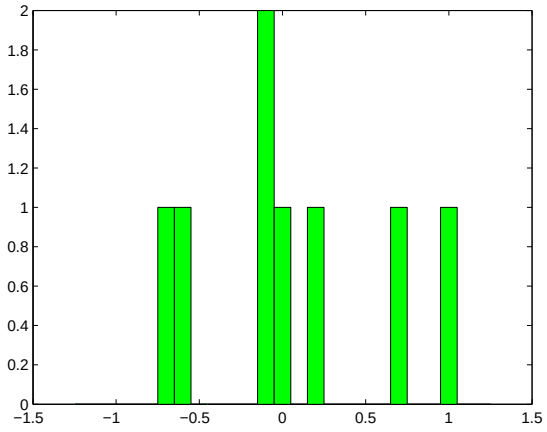


Figure 2.8: The histogram of the eigenvalues of matrix W with constant edge weights.

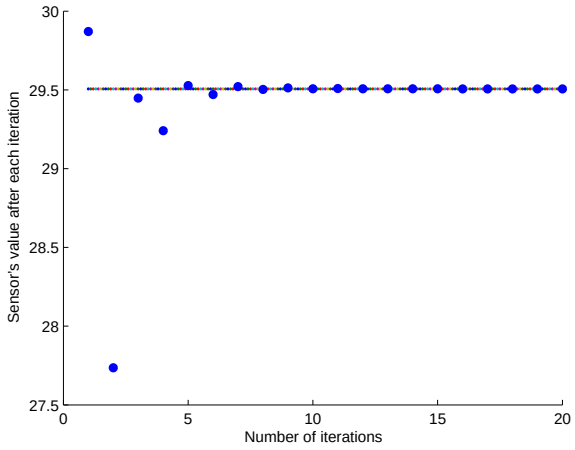


Figure 2.9: The convergence of one sensor after 20 distributed linear iterations.

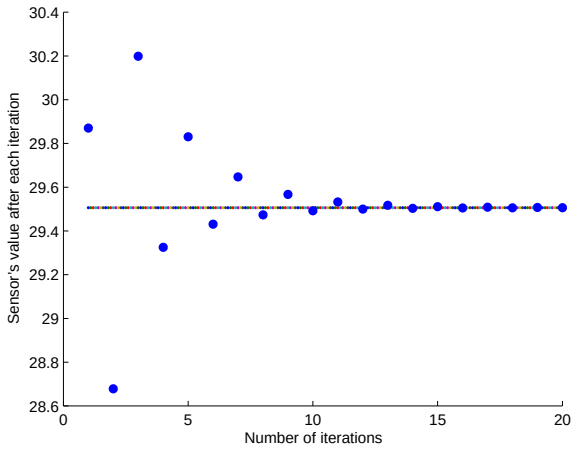


Figure 2.10: The convergence of one sensor of the graph with constant edge weights after 20 distributed linear iterations.

2.3 Basic Concepts of Optimization problems

The concept of optimization is basic to much of what we do in our daily lives: a desire to do better or be the best in one field or another. Engineers, try to produce the best possible result with the available resources. In a highly competitive modern world it is no longer sufficient to design a system whose performance on the required task is just satisfactory. It is essential to design the best system. Therefore, many problems that an engineer has to solve are expressed as optimization problems.

Figure 2.11: Locate the top of the hill while blindfolded, http://www.vrand.com/education.html.

Consider an example in Figure 2.11. One hiker bets that he can locate the top of the hill while blindfolded. The other one agrees but asks the first one to also stay inside the fences. Translating this situation into optimization problem formulation, one can see that the objective is to find the highest point on the hill. Therefore, objective function is the height achieved by the first hiker with respect to his original position. The design variables are longitude and latitude - the coordinates, defining the position of the hiker. The constraints are that the hiker has to stay inside the fences. Note here, that in general, the hiker may start the search from outside the fences.

This simple problem can be expressed as the following optimization problem:

$$\begin{aligned} \max_{\mathbf{x}} \quad & Y = f(\mathbf{x}) \\ \text{subject to} \quad & f_1(\mathbf{x}) \leq 0, \\ & f_2(\mathbf{x}) \leq 0, \end{aligned} \tag{2.14}$$

where f_1, f_2 are the constraints for each fence and $\mathbf{x} = [x_1\ x_2]^T$. The optimization process is illustrated in Figure 2.12. This optimization problem can be divided into the following steps. Find a search direction that will improve the objective while staying inside the fences; Search in this direction until no more improvement can be made by going in this direction; The process can be repeated, until no search direction can be found that improves the objective.

The optimization problem formulation and the optimization process presented above are very general and can be applied to any design problem in any field. For example, let $G = (V, E)$ be an undirected network. Consider a set of nodes K, where each node $k \in K$ is determined by a source-terminal pair (s_k, t_k) of vertices, and a demand d_k, which is to be routed over the network from s_k to t_k. Let u_{ij} represent an upper bound on the capacity of edge $\{i, j\}$. The problem is to decide how much demand is routed over each edge, without violating capacity constraints. The (linear) costs c_{ij} are determined by the actual capacity

Figure 2.12: Optimization process, http://www.vrand.com/education.html.

usage, and should be minimized. Variables f_{ij}^k indicate the demand of node $k \in K$ that is routed from i to j over edge $\{i,j\}$. The formulation of this problem is the following:

$$\begin{aligned}
\min \quad & \sum_{\{i,j\}\in E}\sum_{k\in K} c_{ij}(f_{ij}^k + f_{ji}^k) \\
\text{subject to} \quad & \sum_{j:\{i,j\}\in E} f_{ij}^k - \sum_{j:\{i,j\}\in E} f_{ji}^k = \begin{cases} d^k, & \text{if } i = s_k \\ -d_k, & \text{if } i = t_k \\ 0, & \text{otherwise.} \end{cases} && \forall k \in K,\ \forall i \in V, \\
& \sum_{k\in K} (f_{ij}^k + f_{ji}^k), && \forall \{i,j\} \in E, \\
& f_{ij}^k,\ f_{ji}^k \geq 0, && \forall k \in K,\ \forall \{i,j\} \in E.
\end{aligned} \tag{2.15}$$

The first constraint is the standard flow conservation constraints for each node. The capacity on an edge is undirected because installed capacity can be set-up for usage by traffic in both directions. Thus, the sum of forward and backward flow on an edge should not exceed its capacity. This is reflected by the next constraint. The cost function may be used to set preference on shortest paths (in length or number of connections).

2.3.1 Convex Optimization

By recognizing and formulating a problem as a convex optimization problem, one can solve it efficiently, using interior-point or other special methods [43]. These solution methods are reliable enough to be embedded in a computer-aided design or analysis tool, or even a real-time reactive or automatic control system. Even more importantly, there are also theoretical and conceptual advantages since the associated dual problem often has an interesting interpretation and sometimes leads to a distributed method for solving it.

Convex optimization is a well-developed area in both the theoretical and practical aspects, especially during the last two decades when a number of fundamental and practical results have been obtained. A convex optimization problem (or convex program) is one of the form:

$$\begin{aligned}
\min_{\mathbf{x}} \quad & f_0(\mathbf{x}) \\
\text{subject to} \quad & f_i(\mathbf{x}) \leq 0, \quad 1 \leq i \leq m, \\
& h_i(\mathbf{x}) = 0, \quad 1 \leq i \leq l,
\end{aligned} \tag{2.16}$$

where $\mathbf{x} \in R^n$ is the optimization variable, f_0 is the convex objective function, $f_1, \ldots, f_m$ are m convex inequality constraint functions, and $h_1, \ldots, h_l$ are l linear equality constraint functions. A point is *feasible* if it satisfies all the constraints $f_i(\mathbf{x}) \leq 0$ and $h_i(\mathbf{x}) = 0$. The

problem (2.16) is said to be feasible if there exists at least one feasible point and infeasible otherwise.

The basic idea in Lagrangian duality is to take the constraints in (2.16) into account by augmenting the objective function with a weighted sum of the constraint functions. The Lagrangian $L : R^n \times R^m \times R^l \longrightarrow R$ associated with the problem (2.16) is defined as

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{l} \nu_i h_i(\mathbf{x}), \tag{2.17}$$

where λ_i is the Lagrange multiplier associated with the i-th inequality constraint, while ν_i is the Lagrange multiplier associated with the i-th equality constraint. The vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ are called the *dual variables* or *Lagrange multiplier* vectors associated with the problem (2.16). Similarly, the original objective function $f_0(\mathbf{x})$ is referred to as the *primal objective*, whereas the *dual objective* is defined as the minimum value of the Lagrangian over $\mathbf{x}$

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}). \tag{2.18}$$

Note that the infimum in (2.18) is with respect to all $\mathbf{x}$ (not necessarily feasible points), and that the dual variables $\boldsymbol{\lambda}, \boldsymbol{\nu}$ are dual feasible if $\boldsymbol{\lambda} \geq 0$.

For each pair $\boldsymbol{\lambda}, \boldsymbol{\nu}$ with $\boldsymbol{\lambda} \geq 0$, the Lagrange dual function gives a lower bound on the optimal value p^* of the optimization problem (2.16), which depends on the parameters $\boldsymbol{\lambda}, \boldsymbol{\nu}$. A natural question is: What is the best lower bound that can be obtained from the Lagrange dual function? This leads to the optimization problem

$$\begin{aligned} \max_{\boldsymbol{\lambda}, \boldsymbol{\nu}} \quad & g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geq 0. \end{aligned} \tag{2.19}$$

The difference between the optimal primal value $p*$ and the optimal dual objective d^* is called the *duality gap*, which is always nonnegative (weak duality). A central result in convex analysis [43], [44] is that when the problem is convex, under some mild conditions, the duality gap reduces to zero at the optimal (*i.e.*, strong duality holds). Hence, the primal problem (2.16) can be equivalently solved by using the dual problem formulation (2.19).

The so-called *Karush-Kuhn-Tucker* (KKT) conditions constitute an important analysis tool when dealing with convex optimization problems with differentiable objective and constraint functions. Specifically, if f_i are convex and h_i are affine in problem (2.16), and x^*, λ^*, ν^* are any points that satisfy the KKT conditions:

$$f_i(x^*) \leq 0, \quad i = 1, \ldots, m \tag{2.20}$$

$$h_i(x^*) = 0, \quad i = 1, \ldots, l \tag{2.21}$$

$$\lambda_i^* \geq 0, \quad i = 1, \ldots, m \tag{2.22}$$

$$\lambda_i^* \cdot f_i(x^*) = 0, \quad i = 1, \ldots, m, \quad \text{(complementary slackness)} \tag{2.23}$$

$$\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{l} \nu_i^* \nabla h_i(x^*) = 0, \tag{2.24}$$

then x^* and (λ^*, ν^*) are primal and dual optimal, with zero duality gap.

Summarizing, convexity can be viewed as the watershed between easy and hard optimization problems. This is in part because a local optimum of convex optimization is also globally optimal, the duality gap is zero under certain constraint qualifications, and the KKT conditions are both necessary and sufficient for primal-dual optimality.

2.3.2 Distributed Optimization

In many envisioned applications of wireless sensor networks, the ultimate objective is not merely the collection of raw data, but rather the estimation of certain environmental parameters or functions of interest (e.g., source localization, spatial distributions). Following standard methodology all measurements should be transmitted to a central point for processing, in order to derive an estimate of a parameter or a function. However, this transmission may place a significant drain on communication and energy resources. Bertsekas and Nedic introduced in [45] a distributed algorithm for in-network data processing, aiming at reducing the amount of energy and bandwidth used for communication.

The estimation problems they consider are addressed through the optimization of a cost function (e.g., maximum likelihood, minimum mean squared error, or maximum a posteriori) involving data from all sensors. The distributed algorithms are based on an incremental optimization process. Specifically, a parameter estimate is circulated through the network, and along the way each node makes a small gradient descent-like adjustment to the estimate based only on its local data. Moreover, Bertsekas and Nedic in [46] present an extended convergence analysis of the proposed incremental subgradient method.

Applying results from the theory of incremental subgradient optimization, Nowak and Rabbat show in [47], that for a broad class of estimation problems (robust estimation, energy-based source localization and clustering estimation), the distributed algorithms converge to within an e-ball around the globally optimal value. In the following Section, we present these two approaches, as an example of a distributed optimization technique for a WSN application, that is based on a centralized well-studied optimization algorithm.

2.3.2.1 Incremental Subgradient Optimization

The basic theory, methods and convergence behavior of incremental subgradient optimization were presented by Nedic and Bertsekas in [45], [46]. As an illustration of the basic idea, consider a sensor network comprised of n nodes randomly distributed uniformly over a region, each of which collects m measurements. Many estimation criteria possess the following important form:

$$f(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} f_i(\theta), \tag{2.25}$$

where θ is the parameter of function to be estimated, and $f(\theta)$ the cost function, which can be expressed as a sum of n local functions $\{f_i(\theta)\}_{i=1}^{n}$ in which $f_i(\theta)$ only depends on the data measured at sensor i.

So we formulate the incremental optimization problem as an estimation problem of a set of parameters, which describe the global phenomena being sensed by the network. Denote by Θ this set of parameters which describe the global phenomena being sensed by the network and by $x_{i,j}$ the j-th measurement taken at the i-th sensor. One can write:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \cdot \sum_{i=1}^{n} f_i(\{x_{i,j}\}_{j=1}^{m}, \theta). \tag{2.26}$$

The functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are convex and Θ is nonempty, closed and convex subset of $\mathbb{R}^d$. To simplify the notation we will write $f_i(\theta)$ instead of $f_i(\{x_{i,j}\}_{j=1}^{m}, \theta)$; that is, the function $f_i(\theta)$ depends on the data at the i-th sensor as well as the global parameter θ.

Gradient and subgradient descent methods are popular techniques for iteratively solving optimization problems of this nature [48]. Nowak in [47] introduces the concept of a

subgradient by first recalling an important property of the gradient of a convex differentiable function. For a convex differentiable function, $f : \Theta \rightarrow \mathbb{R}$, the following inequality for the gradient of f at a point θ_0 holds for all $\theta \in \Theta$:

$$f(\theta) \geq f(\theta_0) + (\theta - \theta_0)^T \cdot \nabla f(\theta_0). \tag{2.27}$$

In general, for a convex function f, a *subgradient* of f at θ_0 (observing that f may not be differentiable at θ_0) is any direction g such that

$$f(\theta) \geq f(\theta_0) + (\theta - \theta_0)^T \cdot g \tag{2.28}$$

and the *subdifferential* of f at θ_0, denoted by $\partial f(\theta_0)$, is the set of all subgradients of f at θ_0. Note that if f is differentiable at θ_0, then $\partial f(\theta_0) \equiv \{\nabla f(\theta_0)\}$. The update equation for a *centralized subgradient descent approach* to solving (2.26) is

$$\hat{\theta}^{(k+1)} = \hat{\theta^{(k)}} - \alpha \cdot \sum_{i=1}^{n} g_{i,k}, \tag{2.29}$$

where $g_{i,k} \in \partial f_i(\hat{\theta}^{(k)})$, α is a positive step size, and k is the iteration number. The subgradient method was originally developed by Shor in the Soviet Union in the 1970s. Basic references on subgradient methods and their convergence properties include Shor's book [49], and Bertsekas' book [48] combined with an convergence analysis of the subgradient method for optimization problems.

2.3.2.2 Incremental Subgradient Method in Sensor Networks

Bertsekas' decentralized incremental approach for solving (2.26), was employed by Nowak who applied this method in an optimization problem considering a network of n sensors in which each sensor collects m measurements. The estimation problem Nowak considers is addressed through the optimization of a cost function involving data from all sensor nodes as shown in (2.25). The decentralized approach is based on a parameter estimate that is circulated through the network. Nodes make sequentially updates of the estimate based on sensor's local data.

In the following, a decentralized incremental approach is used for solving (2.26) in which each update iteration (2.29) is divided into a cycle of n subiterations, and each subiteration focuses on optimizing a single component $f_i(\theta)$. If $\hat{\theta}(k)$ is the vector obtained after k cycles, then

$$\hat{\theta}^{(k)} = \psi_n^{(k)}, \tag{2.30}$$

where $\psi_n^{(k)}$ is the result of n subiterations of the form

$$\psi_i^{(k)} = \psi_{i-1}^{(k)} - \alpha \cdot g_{i,k}, \tag{2.31}$$

where $i = 1, 2, ...n$, $g_{i,k} \in \partial f_i(\psi_{i-1}^{(k)})$ and $\psi_0^{(k)} = \psi_n^{(k-1)}$. For the purposes of analyzing the rate of convergence, we can assume that the algorithm is initialized to an arbitrary starting point $\hat{\theta}^{(0)} \in \Theta$.

As mentioned before, Bertsekas and Nedic in [46] presented some convergence results for the incremental subgradient method for various stepsize rules (constant stepsize, diminishing stepsize and dynamic stepsize) and gave proofs of the convergence rate estimates. Their results are based on two assumptions. First, they assume that an optimal solution,

θ^*, exists. Additionally, they assume that there is a scalar, $\zeta > 0$, such that all subgradients of the functions $f_i(\theta)$ are upper bounded:

$$\| g_{i,k} \| \leq \zeta. \tag{2.32}$$

They also assume that the distance between the starting point $\hat{\theta}^{(0)}$ and an optimal solution θ^* is bounded:

$$\| \hat{\theta}^{(0)} - \theta^* \| \leq c_0. \tag{2.33}$$

Under these assumptions and for a constant positive stepsize α, setting

$$e = \alpha \cdot \zeta^2, \tag{2.34}$$

guarantees convergence to a solution, which brings the objective function within an e-ball of the optimal value $f(\theta^*)$ after at most K cycles:

$$K \leq \frac{c_0 \cdot \zeta^2}{e^2}. \tag{2.35}$$

In addition to a theoretical analysis of distributed estimation algorithms of this sort, Nowak et. al also investigate this application in three problems:

- Robust estimation: Robust estimates are often derived from criteria other than the sum-of-squared errors including Huber loss function.
- Cluster and density estimation: In the discovery process, one may have very little prior information about characteristics of the environment and distribution of the data. Clustering and density estimation are standard first-steps in data exploration and analysis and usually lead to non-quadratic optimizations.
- Source localization: Source localization algorithms are often based on a squared-error criterion (e.g., Gaussian noise model), but the location parameter of interest is usually nonlinearly related to the data (received signal strength is inversely proportional to the distance from source to sensor) leading to a nonlinear estimation problem.

All three problems can be tackled using distributed algorithms, and simulation experiments in these applications demonstrate the potential gains obtainable in practical settings. In the following Section, we present simulations experiments for an energy based source localization problem based on Nowak's distributed algorithm and we present simulation results, investigating the performance of the decentralized incremental algorithm in a variety of scenarios.

2.3.2.3 Application in Energy-Based Source Localization

Estimating the location of an acoustic source is an important problem in both environmental and military applications. In this application described in [47], an acoustic source is positioned at an unknown location in the sensor field. Since the source emits isotropically a signal, the problem is to estimate the source's location using the distributed incremental method described above, using received signal energy measurements taken at each sensor.

Assume that n sensors are uniformly distributed over an area 50×50, and that each sensor knows its own location, $r_i = (r_{i,x}, r_{i,y})$, $i = 1, ..., n$, relative to a fixed reference point. Consider an isotropic energy propagation model for the j-th received signal strength measurement at node i supposing that the source is positioned at point $\theta^* = (\theta^*_x, \theta^*_y)$:

$$x_{i,j} = \frac{A}{\| \theta^* - r_i \|^\beta} + w_{i,j}, \tag{2.36}$$

where $A > 0$ is a constant and

$$\| \theta^* - r_i \| > 1 \tag{2.37}$$

for $i = 1, 2, ..., n$. The exponent $\beta \geq 1$ describes the attenuation characteristics of the medium through which the acoustic signal propagates while $w_{i,j}$ are i.i.d. samples of a zero-mean Gaussian noise process with variance σ^2. A maximum likelihood estimate for the source's location is found by solving

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})^2. \tag{2.38}$$

The cost function of this optimization problem is denoted by $f(\theta)$.

$$f(\theta) = \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})^2. \tag{2.39}$$

It is obvious that the cost function can be expressed as the sum of n local functions $\{f_i(\theta)\}_{i=1}^{n}$, where

$$f_i(\theta) = \frac{1}{m} \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})^2. \tag{2.40}$$

The non linear least squares problem (2.38) clearly fits into the general incremental subgradient framework described in Section 2.3.2.1. In order to implement the iterative distributed algorithm described in Section 2.3.2.2, the gradient of $f_i(\theta)$ is computed:

$$\nabla f_i(\theta) = \frac{2\beta A}{m \| \theta - r_i \|^{\beta+2}} \cdot \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})(\theta - r_i). \tag{2.41}$$

The distributed optimization algorithm can be implemented in the two components of $\theta = (\theta_x, \theta_y)$ separately and rewrite (2.41) as:

$$\frac{\partial f_i(\theta)}{\partial \theta_x} = \frac{2\beta A}{m \| \theta - r_i \|^{\beta+2}} \cdot \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})(\theta_x - r_{i,x}), \tag{2.42}$$

$$\frac{\partial f_i(\theta)}{\partial \theta_y} = \frac{2\beta A}{m \| \theta - r_i \|^{\beta+2}} \cdot \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}})(\theta_y - r_{i,y}), \tag{2.43}$$

or

$$\frac{\partial f_i(\theta)}{\partial \theta_x} = k_i \cdot (\theta_x - r_{i,x}), \tag{2.44}$$

$$\frac{\partial f_i(\theta)}{\partial \theta_y} = k_i \cdot (\theta_y - r_{i,y}), \tag{2.45}$$

where

$$k_i = \frac{2\beta A}{m \| \theta - r_i \|^{\beta+2}} \cdot \sum_{j=1}^{m} (x_{i,j} - \frac{A}{\| \theta - r_i \|^{\beta}}). \tag{2.46}$$

Equation (2.31) can be updated using the expressions (2.44) and (2.45):

$$g_{i,x} = \frac{\partial f_i(\theta)}{\partial \theta_x}, \tag{2.47}$$

$$g_{i,y} = \frac{\partial f_i(\theta)}{\partial \theta_y}. \tag{2.48}$$

Therefore, the update equation of a distributed optimization algorithm for source location for each component can be expressed as:

$$\psi_{i,x} = \psi_{i-1,x} - \alpha \cdot g_{i,x}, \tag{2.49}$$

$$\psi_{i,y} = \psi_{i-1,y} - \alpha \cdot g_{i,y}, \tag{2.50}$$

or

$$\psi_{i,x} = (1 - \alpha \cdot k_i) \cdot \psi_{i-1,x} + \alpha \cdot k_i \cdot r_{i,x}, \tag{2.51}$$

$$\psi_{i,y} = (1 - \alpha \cdot k_i) \cdot \psi_{i-1,y} + \alpha \cdot k_i \cdot r_{i,y}. \tag{2.52}$$

Having started at an arbitrary point $\theta^{(0)}$, after k cycles the algorithm is expected to converge to a solution $\hat{\theta}^{(k)} = (\psi_{i,x}^{(k)}, \psi_{i,y}^{(k)})$ that is close enough to the optimal value, which is the real location of the source θ^*. The magnitude is bounded of the gradient by first observing that

$$\| \nabla f_i(\theta) \| \leq \frac{2\beta A \parallel \theta - r_i \parallel}{m \parallel \theta - r_i \parallel^{\beta+2}} \cdot \sum_{j=1}^{m} \mid x_{i,j} - \frac{A}{\parallel \theta - r_i \parallel^{\beta}} \mid . \tag{2.53}$$

Taking under consideration that $\parallel \theta - r_i \parallel > 1$, one can rewrite (2.53) as:

$$\| \nabla f_i(\theta) \| \leq \frac{2\beta A}{m} \cdot \sum_{j=1}^{m} \mid x_{i,j} - \frac{A}{\parallel \theta - r_i \parallel^{\beta}} \mid \tag{2.54}$$

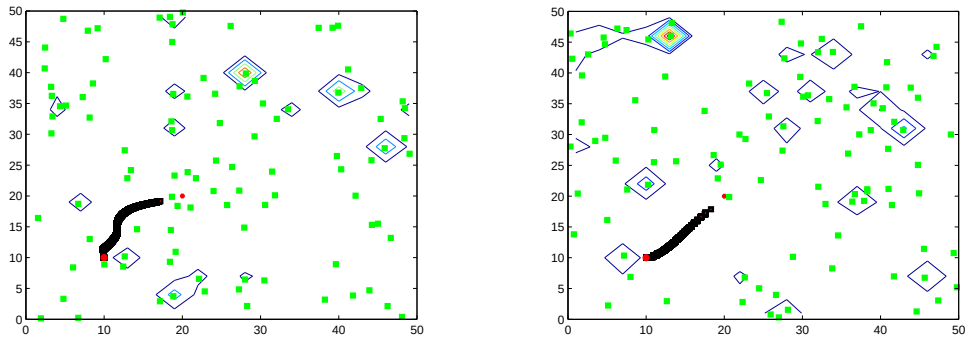
and for a constant c, one can set the expression:

$$\| \nabla f_i(\theta) \| = 2 \cdot \beta \cdot A \cdot c \tag{2.55}$$

So, for $i = 1, 2, ...n$ there is a scalar $\zeta = 2 \cdot \beta \cdot A \cdot c$ such that $\parallel g_{i,k} \parallel \leq \zeta$ for all subgradients of the functions $f_i(\theta)$ and $\theta \in \Theta$. Now, the optimal value of the step size α can be computed from the expression (2.34) and then the algorithm converges to a solution with certainty, which brings the objective function within an e-ball of the optimal value after at most K cycles.

Now, consider a network consisting of $n = 100$ sensors uniformly distributed in a 50×50 square. Point (20,20) is chosen to be the source location. The exponent, which describes the attenuation characteristics of the medium through which the acoustic signal propagates, is equal to 1 (i.e., $\beta = 1$). The source emits a signal with strength $A = 100$ and each sensor makes 10 measurements at a signal to noise ratio (SNR) of 3 dB. The algorithm is initialized at the starting point $\hat{\theta}^{(0)} = (10, 10)$. The optimal value of the step size is calculated using the theoretical analysis of Bertsekas [46] for the specific scenario. The value of α for a specific deployment of these sensors is $\alpha = 2.8409e - 008$, and the number of cycles needed for a guaranteed convergence to a solution is $K = 165.934.718$. Consider that the convergence is accomplished when the solution is within an e-ball with radius e=3 of the optimal solution $\hat{\theta} = (20, 20)$. For such a small step size, the number of cycles is expected to be big.

Figure 2.13 depicts a path produced by the decentralized incremental subgradient algorithm and displayed on top of contours of the log-likelihood function. The values of the parameters are exactly the same as the ones of the previous simulation (that is $m = 10, n = 100, b = 1, SNR = 3$) but for a step size $\alpha = 0.001$. It can be seen in Figure 2.13(a) that the algorithm for a signal with strength $A = 30$ converges to the solution $\hat{\theta} = (17.1976, 19.1751)$ after $K = 114$ cycles, but in figure 2.13(b) for another deployment of sensors for a signal with strength $A = 100$, the algorithm converges faster, within $K = 23$ cycles, to the solution $\hat{\theta} = (17.2029, 18.9235)$.



(a) (b)

Figure 2.13: An example path taken by the decentralized incremental subgradient algorithm displayed on top of contours of the log-likelihood function. The true source location is at the point (20,20). The green squares correspond to the 100 sensors that are uniformly distributed in the area $[0, 50] \times [0, 50]$. Each sensor makes 10 measurements. (a) The signal strength is $A = 30$, the signal to noise ratio is 3 dB and the attenuation characteristics of the medium are defined by the parameter $\beta = 1$. (b) The signal strength is $A = 100$, the signal to noise ratio is 3 dB and the attenuation characteristics of the medium are defined by the parameter $\beta = 1$.

The convergence of the algorithm is investigated for a specific deployment of $n = 10$ sensors, making $m = 5$ measurements each but for different values of the signal strength A. Figure 2.14 shows how many cycles are required so that the algorithm converges to a solution within an e-ball (e=3) of the optimal solution $\hat{\theta} = (20, 20)$. The signal to noise ratio is equal to 3, and $\beta = 1$ and the step size $\alpha = 0.001$. Moreover, the convergence of the algorithm is investigated for the same deployment but with different values of the parameter β. Comparing Figure 2.15(a) to Figure 2.15(b) one can easily notice that the algorithm converges faster for a stronger signal ($A = 100$) than for a weaker signal ($A = 3$). Figure 2.15(a) depicts six cases where the algorithm converges in $K < 10^4$ cycles while in Figure 2.15(b) only in one case the algorithm converges in $K < 4.5 \cdot 10^4$ cycles. Finally, Figure 2.16 depicts the number of cycles needed for a convergence with the same parameters that were used in Figure 2.15(a) but for different deployment of sensors. Observing the two figures, parameter β does not affect the time of convergence. On the other hand, the convergence depends greatly on the deployment of the sensors in the area.

2.4 Support Vector Machines

Support Vector Machines (SVMs) is a learning system based on recent advances in statistical learning theory. Training an SVM involves optimization of a convex cost function. SVMs deliver state-of-the-art performance in real-world applications such as text categorization, hand-written character recognition, image classification, biosequences analysis, etc., and are now established as one of the standard tools for machine learning and data mining.

As a principle approach to machine learning, and particular to various types of machine,

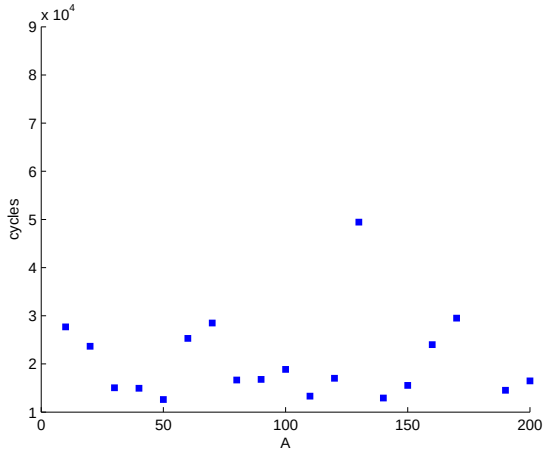
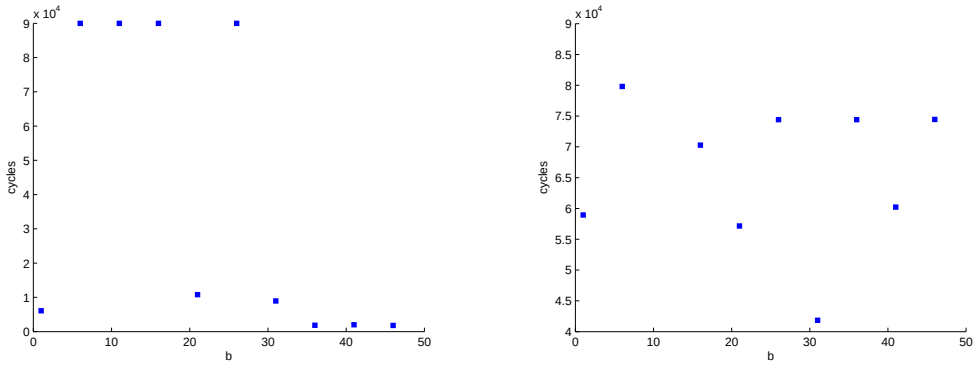


Figure 2.14: The number of cycles required so that the algorithm converges to a solution within an e-ball (e=3) of the optimal solution $\hat{\theta} = (20, 20)$ vs. signal strength A. The algorithm does not reach a convergence point.



(a) (b)

Figure 2.15: The number of cycles required so that the algorithm converges to a solution within an e-ball (e=3) of the optimal solution $\hat{\theta} = (20, 20)$ vs. the exponent which describes the attenuation characteristics of the medium β. The algorithm does not reach a convergence point. (a) The signal strength is $A = 100$ and the signal to noise ratio is 3 dB. (b) The signal strength is $A = 3$ and the signal to noise ratio is 3 dB.

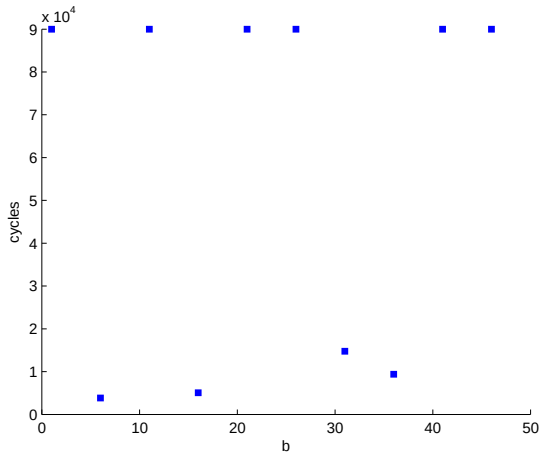


Figure 2.16: The number of cycles required for another deployment of sensors so that the algorithm converges to a solution within an e-ball (e=3) of the optimal solution $\hat{\theta} = (20, 20)$ vs. signals' strength A. When the number of cycles is $9 \cdot 10^4$, the algorithm does not reach a convergence point.

i.e., classification, regression, or novelty detection tasks, SVMs exhibit a good generalization to new data. The motivation for SVMs, actually came from statistical learning theory [50]. Unlike other approaches to machine learning, such as neural networks, SVM is a learning approach, using a training set with unknown statistics in order to make inferences and decision rules with small loss for any new data. Relatively very few parameters require tuning (kernel function and soft margin), therefore they are very easy to use in any application. In contrast to neural networks, SVM is a quadratic problem that involves optimization of a convex cost function, hence, there are no false local minima and no problems finding the global minima.

The idea behind the support vector machines is to look at the radial basis function (RBF) network as a mapping machine, through the kernels, into a high dimensional feature space. Then a hyperplane linear classifier is applied in this transformed space utilizing those patterns vectors that are closest to the decision boundary. These are called support vectors corresponding to a set of data centers in the input space. The hyperplane in this feature space (or the nonlinear decision surface in the original space) will be optimized in giving the largest tolerance margin. The algorithm computes all the unknown parameters automatically including the number of these centers. In the last decade, significant advances have been made in support vector machine research [24], both theoretically using statistical learning theories [50, 51], and algorithmically based on optimization techniques [52]. Since this is a relatively new design methodology for pattern classification, we give a substantially detailed review in this section, and then present a case study on right ventricle shape data.

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. in [53] 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, this work was extended to a *soft margin* approach, [54]. 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 multi-classifier 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, 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, such as regression and clustering. The regression algorithm extension has been refined by Alex Smola and Bernhard Scholkopf and pioneered by Vapnik [50]. 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.

Next, we formulate the learning task or training of a binary SVM classifier as a convex optimization problem, assuming that all the computation is performed in a centralized manner at a certain fusion center. First, we examine the case of two linearly separable data sets, and then we describe an alternative formulation of an linear SVM for linearly inseparable data sets, also called as $\nu - SVM$. Both cases can be described using several equivalent formulations, each of them having a concrete geometrical interpretation.

2.4.1 SVM for Linearly Separable Data Sets

This problem is a wonderful example of a mathematical programming concept of duality. The solutions of the primal and the dual problems are identical (*i.e*, strong duality holds), and both have interesting geometrical interpretations.

2.4.1.1 Primal Problem

Consider a binary classification task with data vectors $\mathbf{x}_i,\ i = 1, \ldots, n$ from class {+1} and $\mathbf{y}_j,\ j = 1, \ldots, m$ from class {-1}. Assume that these data sets are linearly separable. Intuitively, the plane that best separates the data sets, is the one further from both classes. With this choice of hyperplane, small changes in the data will not yield misclassification errors. Thus, intuitively, one is interested in constructing a hyperplane that maximizes the minimum distance from the plane to each set. A plane supports a class if all points in that class are on one side of that plane. For the points in class {+1} and class {-1}, the goal is to find a vector $\mathbf{w}$ and an offset b such that $\mathbf{w} \cdot \mathbf{x}_i + b \geq 1$ and $\mathbf{w} \cdot \mathbf{y}_j + b \leq 1$, respectively. Maximizing the slab $\frac{2}{\|\mathbf{w}\|}$, is equivalent to minimizing $\frac{\|\mathbf{w}\|}{2}$ in the following quadratic programming problem:

$$\begin{array}{lll} \min\limits_{\mathbf{w}} & \frac{\|\mathbf{w}\|^2}{2} & \\ \text{subject to} & \mathbf{w}^T \cdot \mathbf{x}_i + b \geq \quad 1, & i = 1, \ldots, n \\ & \mathbf{w}^T \cdot \mathbf{y}_j + b \leq -1, & j = 1, \ldots, m. \end{array} \tag{2.56}$$

This problem turns out to be a convex optimization problem since the objective and the constraint functions are convex [43]. It is very interesting at this point to examine also the dual problem and interpret the results geometrically.

2.4.1.2 Dual Problem

Using the Lagrangian duality described in Section 2.19, it is possible to obtain an equivalent alternative formulation given by:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\theta},\boldsymbol{\gamma}} & \| \sum_{i=1}^{n} \theta_i \mathbf{x}_i - \sum_{j=1}^{m} \gamma_j \mathbf{y}_j \|^2 \\ \text{subject to} & \sum_{i=1}^{n} \theta_i = 1,\ \sum_{j=1}^{m} \gamma_j = 1, \\ & \theta_i \geq 0,\ \gamma_j \geq 0. \end{array} \tag{2.57}$$

where $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_n]$, $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_m]$. In the framework of optimization theory, the convex optimization problems (2.56) and (2.57) are said to be dual of each other.

Observing (2.57), one can easily notice that it can be interpreted as the problem of finding the minimum distance between two convex hulls[1]: the convex hull that contains the data $\{\mathbf{x}_i\}$ of one class and the convex hull that contains the data $\{\mathbf{y}_j\}$ of the other class (cf. Figure 2.17(a)). The optimal discriminant (classifier) is defined by vector $\mathbf{w}^*$ and offset b^* as follows:

$$\mathbf{w}^* = \sum_{i=1}^{n^*} \theta_i^* \mathbf{x}_i - \sum_{j=1}^{m^*} \gamma_j^* \mathbf{y}_j, \tag{2.58}$$

$$b^* = 1 - \mathbf{w}^{*T} \mathbf{x}_i, \text{ or } b^* = 1 - \mathbf{w}^{*T} \mathbf{y}_j, \tag{2.59}$$

The resulting separating hyperplane is expressed by means of a linear combination of the so-called *support vectors*, *i.e.*, those $\mathbf{x}_i's$ and $\mathbf{y}_j's$ corresponding to non-zero θ_i^* and γ_j^*, respectively, (cf. Figure 2.17(b)). Note that $n^* + m^* << n + m$, *i.e,* the support vectors are a small subset of the original data and hence they can be considered to constitute a sparse representation of the measurements.

2.4.2 ν-SVM for Linearly Inseparable Data Sets

In real word applications, it is common to encounter the case of linearly non separable classes. It is possible to solve this kind of problems with a linear SVM only by relaxing the data constraints. In the following, we present the primal and the dual formulation of a SVM in this case, and we also describe the $\nu - SVM$ formulation that leads also to a very interesting geometrical interpretation.

2.4.2.1 Primal Problem

The width of the classifier's margin, which determines the number of weak and wrong classifications in the training set, is defined as the distance between the pair of parallel support planes described by $\mathbf{w}^T \cdot \mathbf{x}_i + b = 1$, $\mathbf{w}^T \cdot \mathbf{y}_j + b = -1$. The training vectors now can be any of the following:

- vectors $\mathbf{x}_i$, $\mathbf{y}_j$ that are outside the support planes and are correctly classified, *i.e.*, $\mathbf{w}^T \cdot \mathbf{x}_i + b \geq 1$, $\mathbf{w}^T \cdot \mathbf{y}_j + b \leq -1$,
- vectors $\mathbf{x}_i$, $\mathbf{y}_j$ falling inside the slab of the support planes but are correctly classified, *i.e.*, $0 \leq \mathbf{w}^T \cdot \mathbf{x}_i + b \leq 1$, $-1 \leq \mathbf{w}^T \cdot \mathbf{y}_j + b \leq 0$,
- vectors $\mathbf{x}_i$, $\mathbf{y}_j$ that are misclassified, *i.e.*, $\mathbf{w}^T \cdot \mathbf{x}_i + b \leq 0$, $\mathbf{w}^T \cdot \mathbf{y}_j + b \geq 0$.

[1] A convex hull of X is the set of points of the form $\sum_{i=1}^{n} t_i \mathbf{x}_i$, where the numbers t_i are non-negative and sum to 1, n is an arbitrary natural number and the points $\mathbf{x}_i$ are in X.

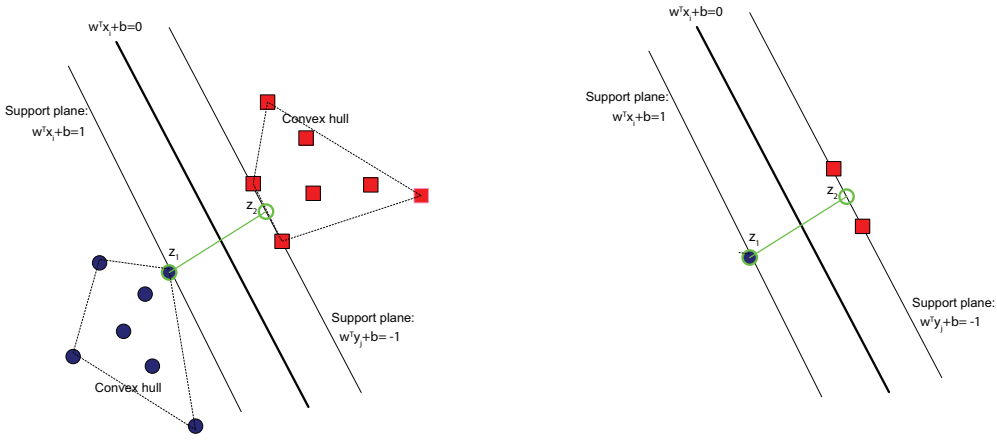


Figure 2.17: a) The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes, and intersects it half-way between the two classes. b) The optimal hyperplane is constructed only from the three support vectors. The other measurements give no information about the hyperplane.

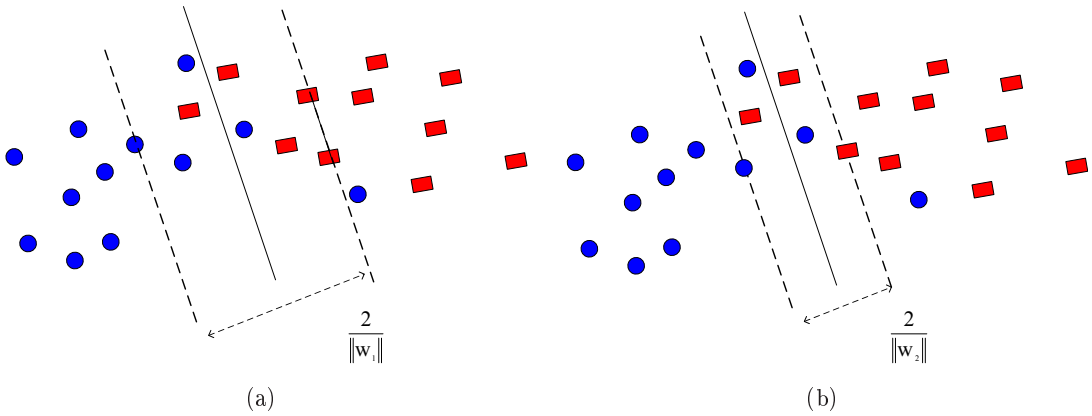


Figure 2.18: An example of two non separable classes and the resulting SVM linear classifier (full line) with the associated margin for the values a) $C_1 = 0.1$ and the margin $\frac{2}{\|\mathbf{w_1}\|}$ and b) $C_2 = 100$ and the margin $\frac{2}{\|\mathbf{w_2}\|} < \frac{2}{\|\mathbf{w_1}\|}$.

All three cases can be expressed in one inequality by introducing the so called *slack variables* u_i, and v_j for each class respectively, so that $\mathbf{w}^T \cdot \mathbf{x}_i + b \geq 1 - u_i$, $\mathbf{w}^T \cdot \mathbf{y}_j + b \leq -1 + v_j$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$. The first category of the data are vectors corresponding to $u_i = 0$, the second to $0 < u_i \leq 1$ and the third to $u_i > 1$. Similarly for v_j. The goal now is to make the margin as large as possible but at the same time to keep the number of points with $u_i > 0$, $v_j > 0$ as small as possible. In mathematical terms, this is equivalent to the following optimization problem:

$$\begin{array}{ll} \min\limits_{\mathbf{w},\mathbf{u},\mathbf{v}} & \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C\Big(\sum_{i=1}^{n} u_i + \sum_{j=1}^{m} v_j\Big) \\ \text{subject to} & \mathbf{w}^T \cdot \mathbf{x}_i + b \geq 1 - u_i,\ i = 1, \ldots, n \\ & \mathbf{w}^T \cdot \mathbf{y}_j + b \leq -1 + v_j,\ j = 1, \ldots, m, \\ & \mathbf{u} \succeq 0,\ \mathbf{v} \succeq 0. \text{ (component-wise } \geq 0) \end{array} \tag{2.60}$$

The parameter C is a positive constant which gives the relative weight of the number of misclassified points, compared to the width of the slab. Figures 2.18(a) and 2.18(b) depict an example of the trade-off between the width of the margin and the number of misclassified data. Notice also that for $\mathbf{u} = \mathbf{v} = \mathbf{0}$, problem (2.60) coincides with optimization problem (2.56) for the case of linearly separable data sets.

However, since the margin is such an important entity in the design of a SVM, an alternative formulation is the also known as $\nu - SVM$ method. The margin now is defined by the support planes:

$$\begin{aligned} \mathbf{w}^T\mathbf{x} + b &= \rho, \\ \mathbf{w}^T\mathbf{y} + b &= -\rho, \end{aligned}$$

where $\rho \geq 0$ is a free variable to be optimized. Under this new setting, the primal problem (2.56) is now given by:

$$\begin{array}{ll} \min\limits_{\mathbf{w},\mathbf{u},\mathbf{v},\rho} & \frac{\|\mathbf{w}\|^2}{2} - \nu\rho + \frac{1}{n+m}\Big(\sum_{i=1}^{n} u_i + \sum_{j=1}^{m} v_i\Big) \\ \text{subject to} & \mathbf{w}^T \cdot \mathbf{x}_i + b \geq \rho - u_i,\ i = 1, \ldots, n, \\ & \mathbf{w}^T \cdot \mathbf{y}_j + b \leq -\rho + v_j,\ j = 1, \ldots, m, \\ & \mathbf{u} \succeq 0,\ \mathbf{v} \succeq 0,\ \text{(component-wise } \geq 0) \\ & \rho \geq 0. \end{array} \tag{2.61}$$

Notice that for $u_i = v_j = 0$, the constraints in (2.61) state that the margin separating the two classes is equal to $\frac{2\rho}{\|\mathbf{w}\|}$. The larger the parameter ρ, the wider the margin and the higher the number of vectors within the margin, for a specific $\mathbf{w}$. The parameter ν controls the influence of the second term in the cost function [55], and its value lies in the range $[0, 1]$. Dividing the cost function by $\frac{\nu^2}{2}$ and the set of the constraints by ν, the solution is not effected, so the optimization problem now becomes:

$$\begin{array}{ll} \min\limits_{\mathbf{w}',\mathbf{u}',\mathbf{v}',\rho'} & \frac{\|\mathbf{w}'\|^2}{2} - 2\rho' + \delta\Big(\sum_{i=1}^{n} u'_i + \sum_{j=1}^{m} v'_j\Big) \\ \text{subject to} & \mathbf{w}'^T \cdot \mathbf{x}_i + b' \geq \rho' - u'_i,\ i = 1, \ldots, n, \\ & \mathbf{w}'^T \cdot \mathbf{y}_j + b' \leq -\rho' + v'_j,\ j = 1, \ldots, m, \\ & \mathbf{u}' \succeq 0,\ \mathbf{v}' \succeq 0,\ \text{(component-wise } \geq 0) \\ & \rho' \geq 0, \end{array} \tag{2.62}$$

where $\delta = \frac{2}{\nu(n+m)}$, $\mathbf{w}' = \frac{\mathbf{w}}{\nu}$, $b' = \frac{b}{\nu}$, $\rho' = \frac{\rho}{\nu}$, $u_i' = \frac{u_i}{\nu}$, $v_j' = \frac{v_j}{\nu}$. For simplicity, we use the same notation:

$$\begin{array}{ll} \min\limits_{\mathbf{w},\mathbf{u},\mathbf{v},\rho} & \frac{\|\mathbf{w}\|^2}{2} - 2\rho + \delta\Big(\sum_{i=1}^{n} u_i + \sum_{j=1}^{m} v_i\Big) \\ \text{subject to} & \mathbf{w}^T \cdot \mathbf{x}_i + b \geq \rho - u_i,\ i = 1, \ldots, n, \\ & \mathbf{w}^T \cdot \mathbf{y}_j + b \leq -\rho + v_j,\ j = 1, \ldots, m, \\ & \mathbf{u} > 0,\ \mathbf{v} > 0, \\ & \rho \geq 0. \end{array} \tag{2.63}$$

This formulation leads to a very interesting geometrical dual interpretation of the problem, analyzed in the following Section.

2.4.2.2 Dual Problem

Using optimality theory, one can write (2.63) with the equivalent dual formulation

$$\begin{array}{ll} \min\limits_{\boldsymbol{\theta},\boldsymbol{\gamma}} & \| \sum_{i=1}^{n} \theta_i \mathbf{x}_i - \sum_{j=1}^{m} \gamma_j \mathbf{y}_j \|^2 \\ \text{subject to} & \sum_{i=1}^{n} \theta_i = 1,\ \sum_{j=1}^{m} \gamma_j = 1, \\ & 0 \leq \theta_i \leq \delta,\ i = 1, \ldots, n, \\ & 0 \leq \gamma_j \leq \delta,\ j = 1, \ldots, m. \end{array} \tag{2.64}$$

The optimal classifier is defined by the vector $\mathbf{w}$ and the offset b as in Equations (2.58) and (2.59) respectively. Again, the discriminant is defined by the support vectors, *i.e.,* the vectors corresponding to the non zero Lagrange multipliers $\boldsymbol{\theta}^*$, $\boldsymbol{\gamma}^*$. The support vectors in the case of linearly inseparable sets, are vectors lying on the support planes and the vectors inside the margin (misclassified vectors).

This optimization problem is almost the same with the one defining the nearest point between two convex hulls in the separable class case, cf. Equation (2.57), with a small, yet significant, difference. The Lagrange multipliers are bounded by the parameter δ. This parameter defines the size of the *reduced convex hulls*, [55]. In general, the reduced convex hull of a vector space X, is denoted as $R(X, \delta)$ and is defined as the convex set:

$$R(X, \delta) = \Big\{ \begin{array}{l} \mathbf{c} : \mathbf{c} = \sum_{i=1}^{n} \lambda_i \mathbf{x}_i : \mathbf{x}_i \in X, \\ \sum_{i=1}^{n} \lambda_i = 1,\ 0 \leq \lambda_i \leq \delta. \Big\} \end{array} \tag{2.65}$$

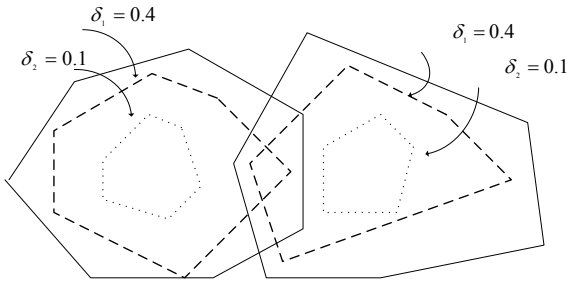
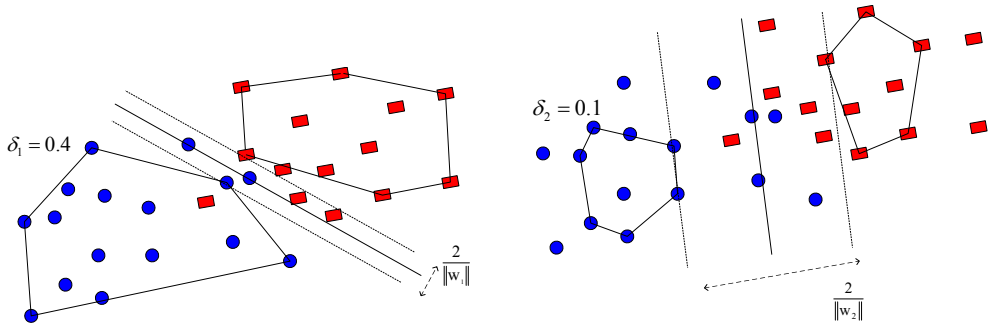


Figure 2.19: The reduced convex hulls (full lines) and the resulting reduced convex hulls (dotted lines) corresponding to $\delta_1 = 0.4$ and $\delta_2 = 0.1$.

Figure 2.19 shows the respective convex hulls for the case of two intersecting data classes. The solid lines indicate the convex hulls, and the dotted lines the reduced convex



(a) (b)

Figure 2.20: An example of two non separable classes. The wider the margin, the more misclassified data. a) For $\delta_1 = 0.4$ and margin equal to $\frac{2}{\|\mathbf{w}_1\|}$, there is only one misclassified data. b) For a wider margin $\frac{2}{\|\mathbf{w}_2\|} > \frac{2}{\|\mathbf{w}_1\|}$ and $\delta_2 = 0.1$ there are three misclassified data.

hulls for $\delta = 0.4$ and $\delta = 0.1$, respectively. The smaller the value of δ, the smaller the size of the reduced convex hull. The smaller the size of the reduced convex hulls, the thicker the margin and hence the larger the number of misclassified vectors. This is illustrated in Figure 2.20.

Now, it is quite clear that (2.60) can be interpreted in geometrical terms as finding the minimum distance between two reduced convex hulls. In the separable case, cf. Equation (2.57), the constraints imply that $0 \leq \theta_i \leq 1$, $0 \leq \gamma_j \leq 1$, which in its geometric interpretation means that the full convex hulls are searched for the nearest points. In contrast, in the linearly inseparable class case, a lower upper bound ($\delta \leq 1$) is imposed for the Lagrange multipliers. Therefore, the search for the nearest points is limited within the respective reduced convex hulls.

Concluding, in both cases, the optimization problem can be interpreted as finding the minimum distance between two convex hulls, in the case of separable sets, and the minimum distance between two reduced convex hulls, in the case of non linearly separable sets. The discriminant for linearly separable sets is constructed by the support vectors corresponding to vectors lying on the support planes, while in the case of linearly inseparable sets by the support vectors corresponding to vectors lying on the support planes and the ones within the margin area.

2.4.3 Distributed SVM Training

Processing sensor data locally requires considerably less energy than communicating it to a distant node, yielding an interesting communication/computation trade-off. To reduce global communication requirements, one needs to perform signal processing to extract key information in a distributed fashion and without losing fidelity.

In general, pattern classification algorithms assume that all the features are available centrally during the construction of the classifier and its subsequent use. But in many practical situations, data are recorded in different geographical locations by sensors, each observing features of local interest and having a partial view of the data. In a WSN application, where sensors can be randomly scattered in the area of interest, traditional

(centralized) SVM training requires the transmission of the data of each sensor to a fusion center. Hence, distributed learning may be used to keep the memory and energy consumption of the learning algorithm at a manageable level as well as to make predictions at a time when the whole data is not yet available.

An appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane is specified via real-valued weights on the training samples. Training samples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Only training samples that lie close to the decision boundary between the two classes, the so-called *support vectors*, receive non-zero weights. Therefore SVMs seem well suited to be trained incrementally. In fact, since their design allows the number of support vectors to be small compared to the total number of training samples, they provide a compact representation of the data, to which new examples can be added as they become available.

Various incremental algorithms have been proposed [56, 57, 58, 59] for training a SVM. The key idea in incremental algorithms is to preserve only the current estimation of the decision boundary at each incremental step along with the next batch of data (or part of it). A disadvantage of these techniques is that they may give only an approximate solution and may require many passes through the whole data set to reach a reasonable level of convergence. In principle, all working methods used to train SVMs, especially shrinking [60], use only a small part of the samples for optimization in each step. This is because in all these methods, none of the samples are discarded during the training and thus all of them have to be considered in each working set selection step. As a consequence, both the memory and the power required are too high to be used in WSNs.

More recently, the research community has focused on algorithms that are based on random communication among sensors, without the aid of any established infrastructure. The idea behind these approaches for distributed SVM training, is that sensors exchange partial information according to a specific form, *e.g.,* support vectors, or vectors that lie on the convex hulls. Independently of our work, Vandenberghe et al. proposed a distributed parallel SVM training mechanism based on the same idea of exchanging support vectors among multiple servers in a strongly connected network [61]. In [62], Navia-Vazquez developed a distributed semiparametric SVM, which aims at further reducing the total information passed between nodes. Finally in [63], an SVM scheme is applied to distributed image classification in a sensor network. In the following Chapters, we present our contribution on distributed algorithms for SVM training.

Chapter 3

Incremental-based Distributed SVM training

Contents

As the research field of mobile computing and communication advances, so does the idea and the need of a distributed, ad-hoc wireless network of hundreds to thousands of microsensors, which can be randomly scattered in the area of interest. Network microsensors enable a variety of new applications such as environmental monitoring, warehouse inventory tracking, location sensing, patient and structural health monitoring. Moreover, in the near future, the development of visual sensor networking technology employing content-rich vision-based sensors will require efficient distributed processing for automated event detection and classification. Hence, the ability to incrementally learn from batches of data with minimal communication requirements is important for real-world applications. Distributed learning may be used to keep the memory and energy consumption of the learning algorithm at a manageable level as well as to make predictions at a time when the whole data is not yet available. This kind of incremental algorithms formulate the exact solution at step $i+1$ in terms of the solution at step i and the new set of available data samples.

Various incremental algorithms have been recently proposed [56, 57, 58, 59] for training a SVM. The key idea in all of them is to preserve only the current estimation of the decision boundary at each incremental step along with the next batch of data (or part of it). Taking advantage of the compact representation of the data set that SVM provide, we design two energy-efficient distributed learning algorithms in the context of a WSN. In Section 3.1 we present the proposed distributed algorithms, while in Section 3.2, we present a set of simulation experiments in order to assess the performance of our proposed approaches comparing them to the performance of a representative centralized SVM algorithm. Finally, in Section 3.3 we provide some conclusions of this work.

3.1 Distributed training of a SVM in a WSN

Let us consider a deployment of m sensors taking measurements in a certain area. Our goal is to be able to train a SVM in an efficient and distributed fashion so that: a) we can

get good classification results on test data and b) our algorithms can be used easily in the context of a WSN, where the training must take place across sensors.

Notice that under the traditional centralized approach, the measurements should be sent first to a base station, where all the processing takes place and a decision boundary that separates the two classes is found. However, direct communication between each sensor and the base station (end-user) in a WSN is both cumbersome (due to the usually large number of sample vectors involved) and highly energy inefficient for a variety of reasons. First, the base station may be far away from the sensing area, and thus direct communication of raw sensor data to the base station can be quite energy costly. In addition, as the number of sensors in a network grows larger and larger, it becomes difficult to manage the vast amount of data collected from the sensors. Also, with increased node density in one location, multiple sensors may view the same event giving rise to sample vectors that are similar to each other, and thus, may be redundant in terms of being useful to determining the separating plane.

On the other hand, as shown in previous work (e.g. [64]), in various practical problems related to WSN, it is possible to design energy-efficient clustering network protocols that greatly reduce the power dissipation. In such protocols, sensors are organized into local spatial clusters. Each cluster has a clusterhead, a sensor which receives data from all other sensors in the cluster, performs data fusion, and transmits the results to the base station. This greatly reduces the amount of data sent to the base station and thus achieves an improved energy efficiency. With this motivation, we propose two novel distributed algorithms in order to train incrementally a SVM in a WSN scenario using a energy-efficient clustering protocol, [65, 66].

3.1.1 Distributed Fixed-Partition SVM training

Typical fixed-partition techniques divide the training samples in batches clusters of sample vectors of fixed size [58]. These kind of algorithms seem appropriate for training incrementally a SVM using *only* partial information at each incremental step [57]. For the WSN scenario, we propose to use a Distributed Fixed-Partition algorithm (DFP-SVM) where the final estimation of the separating hyperplane is obtained incrementally through a sequence of steps, each step taking place at a given cluster.

The key motivation behind this incremental algorithm is that as the number of support vectors is typically very small compared to the number of training samples, the data of previous clusters can be compressed to their corresponding estimated hyperplane (support vectors and offset). Thus, instead of transmitting to the next clusterhead all the measurements stored in the previous one, only the current estimation of the hyperplane is transmitted, which reduces the energy spent. More specifically, suppose there are K clusterheads in the sensor deployment. For each $i = 1, 2, \ldots, K$, the estimation $SVM_i = \{\mathbf{w}_i, b_i\}$ at clusterhead i is obtained combining the previous estimation SVM_{i-1} calculated at cluster $i-1$ and all the sample vectors measured by the sensors belonging to clusterhead i; after this estimation is obtained, the i-th clusterhead transmits SVM_i to the $(i+1)$-th clusterhead, Figure 3.1.

As we show in our experimental results of Section 3.2, after only a complete pass through all the clusters, a good approximation of the optimal separating plane is obtained, that is, the separating hyperplane is very similar to the one obtained using a centralized energy-inefficient algorithm, where all the sample data is used at once in a single training step at the base station.

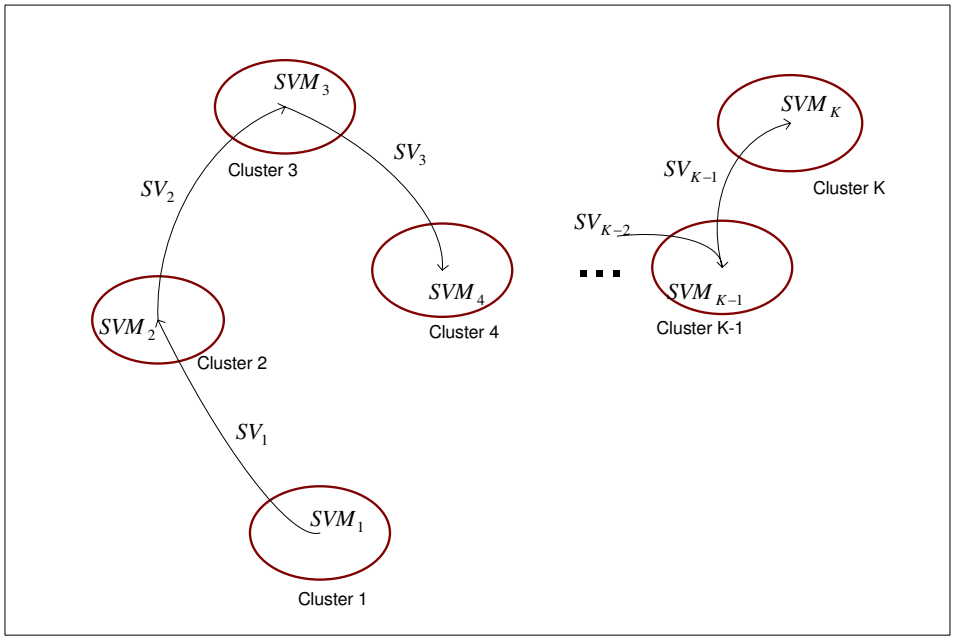


Figure 3.1: Scheme of distributed training of a SVM: For each cluster, the estimation SVM_i at clusterhead i is obtained combining the support vectors (SV_{i-1}) of the previous estimation SVM_{i-1} calculated at cluster $i-1$ and all the sample vectors measured by the sensors belonging to cluster i.

3.1.2 Weighted DFP-SVM training

In many real world applications, the concept of interest (definition of classes to be separated) may be time-varying or space-varying; similarly, the underlying data distribution may change as well. Often these changes make the model built on old data inconsistent with the new data, hence regular updating of the model is necessary. This problem, known as *concept drift*, complicates the task of learning in SVM. A typical example of this phenomenon is weather prediction, where the rules may vary radically depending on the season.

On the other hand, one may also observe changes in the training data, which have no correspondence to controllable parameters of the experiment [12]. For example, in engineering applications, the quality of a machine deteriorates over the course of its lifecycle. Therefore, there is a need to have a robust system that can adapt easily to these uncontrollable changes.

In the case of distributed sequential training of a SVM in a WSN, this effect is even more accentuated: As the data is presented in several batches, changes in the target concept may occur between different batches of data. We are interested in possible concept drifts in WSN applications. For instance, consider a number of sensors distributed in a building, taking measurements of temperature, humidity and light in order to determine which rooms in the building are shinny or not. It is clear that the data distribution changes over time depending on the time of the day. Another example is vehicle tracking for surveillance or monitoring of a hostile environment. In this case, sensors should track all kinds of vehicles that pass through the area and probably have different characteristics such as weight, size, and shape.

We modify our previously proposed algorithm DFP-SVM in order to make it more

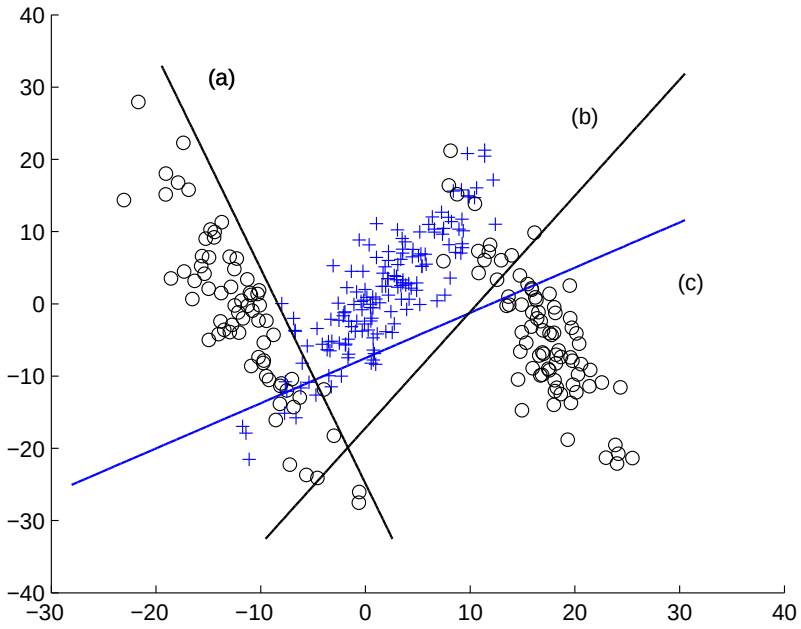


Figure 3.2: Discriminant planes of the training set resulting from: (a) the first incremental step of the DFP-SVM, (b) the second incremental step of the DFP-SVM, and (c) the centralized algorithm.

suitable for WSN applications where there exist concept drifts. Our approach consists of adapting Ruping's algorithm [56] to the WSN context. We call this algorithm as the Weighted Distributed Fixed-Partition SVM training (WDFP-SVM).

As an illustrative example, consider the 300 samples in Figure 3.2 taken by 300 sensors distributed in a field. Let us assume that the data is divided into two batches, such that the first cluster contains the sample vectors with $x < 0$. Training the SVM on this first cluster of data leads to the decision boundary denoted by line (a) in Figure 3.2. If we perform the distributed training for the SVM according to the DFP-SVM algorithm with two incremental steps, the resulting decision boundary (line (b)) largely ignores the old support vectors and it practically corresponds to the decision boundary that would have been learned if only the second cluster of samples had been used ignoring the first cluster. Although in general, this is a desired property of the SVM algorithm (because it means that the SVM is somehow robust against outliers), in the case illustrated in Figure 3.2, it can be seen that most of the outliers are the old support vectors, which causes an important misclassification error.

To address this problem, one needs to make the error on the old support vectors (representing the old learning set), more costly than the error on the new samples. This can be easily achieved by training the SVM with respect to a new loss function [56]. Let $(\mathbf{x}_i, y_i)_{i \in S}$ be the old support vectors from both classes and $(\mathbf{x}_i, y_i)_{i \in I}$ be the new sample vectors. The alternative cost function that should be used instead of (2.60) is:

$$\Phi(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C(\sum_{i \in I} \xi_i + L \sum_{i \in S} \xi_i), \tag{3.1}$$

where the parameter L increases the cost for the old support vectors. An appropriate heuristic choice for the parameter L is to let it be equal to the number of training samples in the previous cluster divided by the number of support vectors. This arises from the idea of approximating the average error of an arbitrary decision function (over all samples) by the average error calculated only over the support vectors. In this way, every support vector influences a constant fraction of all sample vectors.

3.2 Results and Discussion

In this Section, we present a set of simulation experiments covering both the cases with and without concept drift in order to assess the performance of the two proposed distributed SVM algorithms. We evaluate our incremental algorithms comparing them to the traditional centralized SVM training algorithm [60]. At the same time, we also demonstrate that the energy consumption decreases when the SVM is trained incrementally as compared to the centralized case.

3.2.1 Performance of the DFP-SVM algorithm

In the centralized algorithm proposed in [60], there is only an evolving subset of sample data used, making it necessary to address all the constraints associated with large data sets. In our WSN scenario, all the sample data is sent to the base station for processing so that none of the samples are discarded during the training and thus all samples are considered in each working set selection step.

We consider a sensor network composed of $n = 300$ nodes uniformly distributed in the field, where each of the sensors collects sample vectors from two classes. In our experiments, we generate the sample data of the two classes using two Gaussian distributions with two different mean values. Figure 3.3 illustrates the training set of $l = 300$ sample vectors generated by two Gaussian distributions with means $\vec{\mu_1} = [2, 2]$ and $\vec{\mu_2} = [22, 2]$ respectively. The corresponding representation ellipses are thus centered at $(2, 2)$ and $(22, 2)$, with eigenvalue ratios $\lambda_1 = \lambda_2 = \frac{35}{25}$ and rotation angles $\theta_1 = \theta_2 = 20^0$, respectively. The whole training set is partitioned into 12 clusters, each one with a fixed size of 25 sample vectors. Figure 3.3 illustrates our results. Plane (a) is the decision boundary found with the centralized algorithm. The planes denoted by (b) and (c), are the decision boundaries constructed after training the SVM using the DFP-SVM and WDFP-SVM algorithms, respectively. Both distributed algorithms give a good approximation of the decision boundary constructed with the centralized algorithm (plane (a)), in particular, the plane constructed using the WDFP-SVM algorithm (line (c)) coincides exactly with the one obtained using the centralized algorithm.

We also simulated 500 Monte Carlo runs in order to test the performance of these two distributed algorithms on another test data set drawn from the same distributions. Figure 3.4 represents the average error rates (%) for our two proposed algorithms as a function of the consecutive incremental steps. At each step, only the hyperplane parameters are used together with the sample vectors of the next cluster of nodes, and it is shown that with only one pass across the clusters, both distributed algorithms converge to the same average error rate obtained with the centralized algorithm, which requires more energy.

On the other hand, Figure 3.5 simulates a scenario with a concept drift. The first cluster of data consists of measurements of two Gaussians with mean vectors $\vec{\mu_1} = [2, 2]$, $\vec{\mu_2} = [-12, -2]$, eigenvalue ratios $\lambda_1 = \frac{2}{1}$, $\lambda_2 = \frac{35}{25}$ and rotation angles $\theta_1 = 90^0$, $\theta_2 = 20^0$, respectively. The decision boundaries for this subset of training vectors obtained using

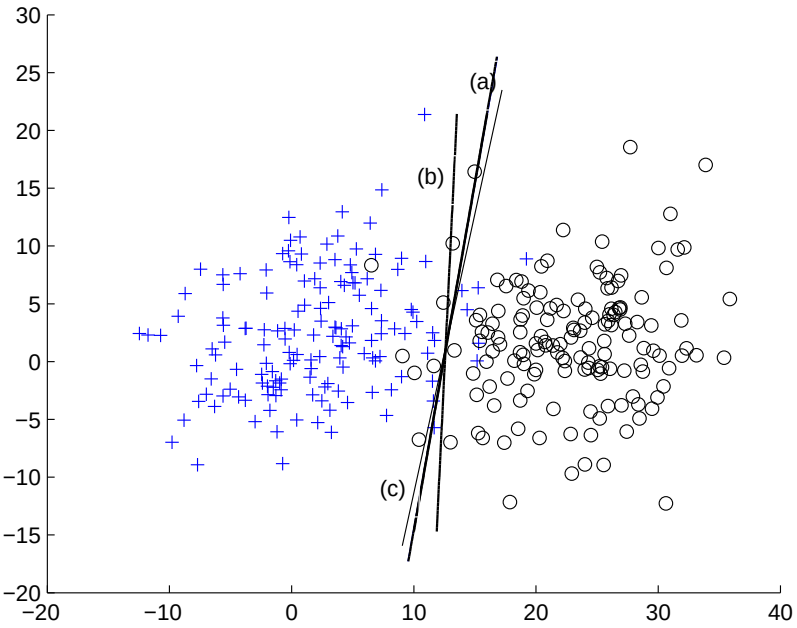


Figure 3.3: Discriminant planes obtained using the centralized algorithm (line (a)) and the two proposed distributed algorithms DFP-SVM (line (b)) and WDFP-SVM (line (c)).

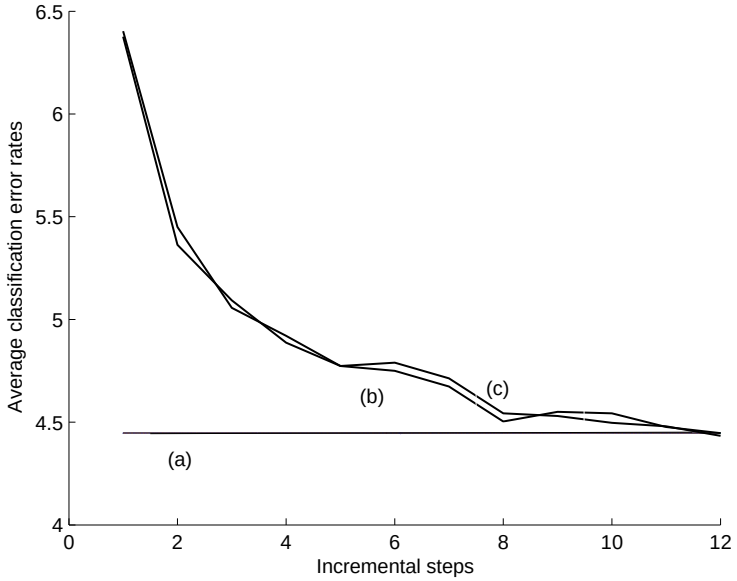


Figure 3.4: Performance of the training algorithms: The average error rate of 500 Monte Carlo runs after training the SVM for consecutive incremental steps applying the centralized algorithm (line (a)), DFP-SVM (curve (b)) and WDFP-SVM (curve b(c)).

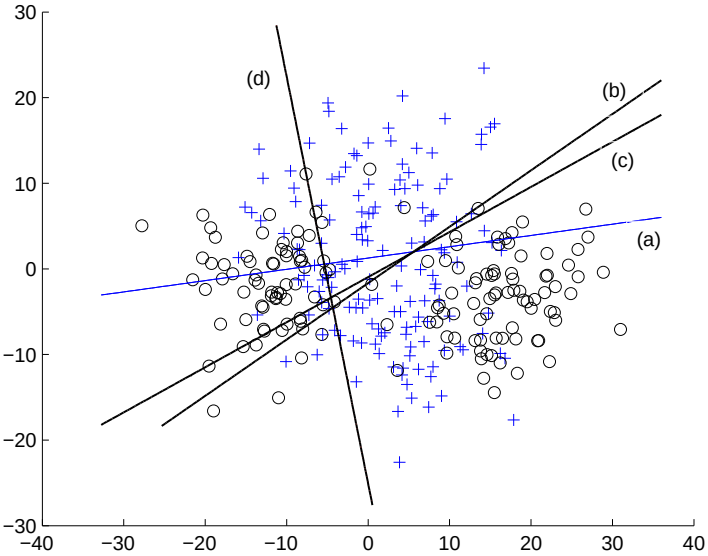


Figure 3.5: Concept drift: Discriminant planes obtained with the centralized algorithm (line (a)), the DFP-SVM algorithm (line (b)) and two consecutive steps of the WDFP-SVM (lines (c),(d)).

DFP-SVM and WDFP-SVM coincide (line (d)). At the next step, once the parameters of the estimated hyperplane are transmitted to the next cluster, in order to introduce the concept drift, we now assume that the next batch of sample vectors consists of samples from the following 2 classes: one class is the same first Gaussian of the previous batch (mean vector $\vec{\mu_1} = [2, 2]$), while the other class consists of a shifted Gaussian with mean vector $\vec{\mu_2} = [16, -3]$. Since the DFP-SVM algorithm ignores the hyperplane obtained from the fist batch of sample vectors, the resulting plane illustrated in Figure 3.5 (line (b)) almost corresponds to the decision boundary that would have been learned using only the second batch of samples alone. However, the WDFP-SVM constructs a plane (line (c)) that lies much closer to the result obtained in the centralized case (line (a)).

3.2.2 Energy efficiency of the DFP-SVM algorithm

At this point we would like to investigate the benefits in terms of energy in a wireless sensor network using these distributed algorithms for training a SVM. Specifically, we are interested in the comparison of energy consumed by the proposed distributed algorithm to a scheme where all sensors transmit their data to a fusion center for processing.

The total energy consumed in the distributed training can be expressed as the sum of the energy consumed in each cluster and the energy consumed for the transmission of the support vectors to the next clusterhead. The energy cost for the transmission of a measurement from node A to node B is proportional to the squared distance of node A to B.

Consider the arrangement of n sensors in a cubic lattice where each sensor is at distance d of a neighbor sensor. Now, separate the sensors in K clusters of $(2k+1) \times (2k+1)$ sensors each. $E_K(d)$ depicts the energy consumption at each cluster for the transmission of the measurements of the sensors (in the cluster) to the clusterhead. $E_{sv}(d)$ depicts the energy consumption for the transmission of the support vectors from one clusterhead to the next

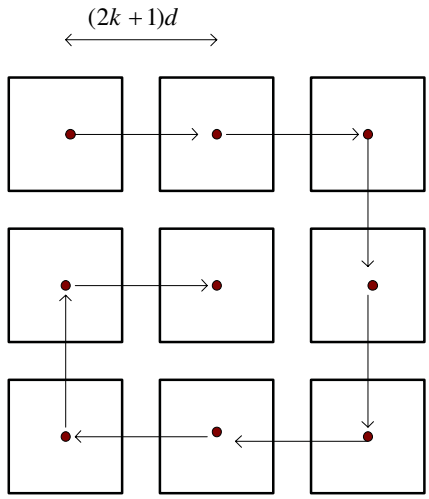


Figure 3.6: Transmission path: Each clusterhead (black dot) transmits the support vectors to the next clusterhead.

clusterhead.

The total energy consumed for the distributed training of a SVM after one pass of all K clusterheads through the path depicted in Figure 3.6 using the proposed algorithms is $E_d(d) = E_{sv}(d) + E_K(d)K$, or:

$$E_d(d) = (2k+1)d^2(N_1 + N_2 + ... + N_{K-1}) \\ +(6d^2k(k+1) + 8d^2\sum_{j=1}^{k-1}\sum_{i=1}^{k-j}2(k-i) + \sum_{j=1}^{k-1}j(k-j)) \cdot K \cdot l,$$

where $N_i,\ i = 1, \dots K$ depicts the number of support vectors at clusterhead i and l is the number of vectors collected at each sensor. On the other hand, the energy cost for the direct transmission of the measurements of n sensors to the base station, which we assume it is in the center of the cubic lattice, is given by the expression:

$$E_c(d) = \left(4\lfloor\frac{\sqrt{n}}{2}\rfloor d + 8(\lfloor\frac{\sqrt{n}}{2}\rfloor d)^2 + 8d^2\sum_{j=1}^{\lfloor\frac{\sqrt{n}}{2}-1\rfloor}\sum_{i=1}^{j} i^2 + j^2(j+1)^2\right) \cdot l$$

We simulated 500 Monte Carlo runs in order to estimate the energy consumed during the distributed training of a SVM. For a scenario of $n = 225$ sensors in a square grid arrangement separated in $K = 9$ clusters consisting of 25 sensors each (hence $k = 2$), the energy cost for the training of the SVM using the proposed distributed algorithm is $E_d(d) = 7505 \cdot d^2$, while in the centralized case the cost is $E_c(d) = 148200 \cdot d^2$.This simulation experiment shows that the proposed distributed algorithm is much more efficient in terms of energy consumption than the centralized algorithm, since it reduces the energy cost by more than 500%.

3.3 Conclusions

In this Chapter, we introduced the concept of distributed training of a SVM in a wireless sensor network. Our research was motivated by the need to have energy-efficient distributed algorithms to be used in large-scale WSNs, whose goal is to perform classification tasks. We presented two distributed algorithms for training a SVM in a WSN. The DFP-SVM algorithm constructs a hyperplane that converges to the plane, which is very close to the one obtained with a centralized algorithm. On the other hand, for the case where concept drift is present, we proposed the WDFP-SVM algorithm which adapts to the non-stationarity. We presented several simulation experiments in order to assess the performance of our proposed approaches. Both algorithms performed only one sequential pass through clusters of sensors.

Chapter 4

Gossip-based Distributed SVM training

Contents

In the previous Chapter, we presented two energy-efficient algorithms that involve a distributed incremental learning for the training of a SVM in a WSN. In all incremental techniques, the update of the estimate is diffused sequentially in the network and the convergence to the global estimate is reached at the final step of the algorithm. Hence, at each time slot only one node has the updated critical information and consequently the optimal estimate. In this case, the trained SVM classifier is constructed at the final step of the algorithm. However, nodes in a WSN, usually operate in environments that are prone to link and node failure. Hence, it is important to design algorithms that are robust to unexpected failures of nodes and consequently to changes in the topology. Thus, to maximize robustness, all nodes should ideally achieve convergence to the same optimal estimate.

Distributed consensus is broadly understood as agents (sensors) achieving a consistent view of the state of nature by interchanging information regarding their current state with their neighbors. Motivated by applications to sensor networks, gossip algorithms (Section 2.2.2) have been studied, for computation and information exchange in an arbitrarily connected network of nodes. Exhaustive research has been made mostly on the averaging problem, where each sensor updates its local estimate by appropriate weighting the estimates of its neighbors [37, 67]. Gossip algorithms are typically based on iterative schemes, whose energy consumption is proportional to the time necessary to achieve consensus and hence the topology of the network [1].

In this Chapter, we use the inherent characteristic specific to SVMs to propose two distributed consensus algorithms for the efficient training of SVM classifiers in WSNs, [68, 69]. Namely, we use the property that the decision hyperplane of a SVM is completely specified by a small fraction of the whole data vectors, the so-called support vectors, (Section 2.4). In the first scheme, each sensor updates its hyperplane at every iteration by combining its support vectors with the support vectors communicated by the neighbors. This results in a close-to-optimal efficient distributed scheme. In a second approach, the information exchanged between sensors describes uniquely and completely the convex hulls of the two classes. Section 4.1 presents the two proposed selective gossip algorithms. In Section 4.2, we

illustrate a set of simulation experiments in order to assess the performance of our proposed approaches and finally in Section 4.3 we present the conclusions of this work.

4.1 Selective Gossiping for SVM Training

Let us consider a deployment of n sensors taking measurements in a certain area. Our goal is to be able to train a SVM in an efficient and distributed fashion so that: a) we can get good classification results on test data, b) all sensors keep refining their estimate concurrently at each time slot in order to reach finally convergence (consensus) to a common global estimate.

In this work, we use gossip algorithms in the context of a SVM. There is a successive refinement to the estimate of each sensor based on communicating information with one-hop neighbors only. Therefore, at each time slot, the new estimate is diffused to the next-hop neighbors and finally at some point all sensors will reach a consensus. Hence, all sensors in the network converge to the same trained SVM classifier, and can classify any new measurements.

The question at this point is what kind of data should neighboring sensors exchange in order to get high classification accuracy but with low energy consumption? WSN nodes should exchange a sufficient amount of data in order to ensure or approximate optimality. On the other hand, the more data is exchanged, the more energy is consumed. The trade-off between optimality and energy consumption led our research to two different algorithms: a) the Minimum Selective Gossip algorithm (MSG-SVM) where the minimum amount of data is selected for diffusion and b) the Sufficient Selective Gossip algorithm (SSG-SVM) where sufficient data is diffused to achieve optimality, that is, same performance as a global centralized algorithm. The proposed algorithms are analyzed in Sections 4.1.1 and 4.1.2, respectively.

4.1.1 Minimum Selective Gossip Algorithm (MSG-SVM)

Communication links in the WSN comprised of n sensors, are represented by a graph whose vertices are the sensors and whose edges are formed by the available communication links. The set of sensors having an active link with the i-th sensor are denoted as the neighborhood N_i. The WSN is deployed to train the SVM using the distributed measurements $M_i(0) := \{\mathbf{x}_{i,j}(0)\}_{i=1}^{n}$, where $1 \leq j \leq k$ and k is the total number of measurements acquired by sensor node i.

We begin by taking k measurements at each node i and then training the SVM locally (for each sensor). The first estimate of the hyperplane is denoted by $\mathbf{w}_i(0),\ i = 1, \ldots, n$, for each node i. When training a SVM, only the support vectors determine the discriminant that separates the data collected by each sensor in two classes [70]. Therefore, the data of each node can be compressed to their corresponding estimated hyperplane and thus to the associated support vectors:

$$SV_i(0) = \{\mathbf{x}_i(0) : \sum_i \alpha_i y_i \mathbf{x}_i(0) = \mathbf{w}_i(0),$$
$$y_i = class\{1, -1\},\ \alpha_i \neq 0\}. \tag{4.1}$$

In general, it holds that $\mid SV_i(0) \mid << \mid M_i(0) \mid$, where $\mid SV_i(0) \mid$ and $\mid M_i(0) \mid$ denote the cardinality of $SV_i(0)$ and $M_i(0)$, respectively [65].

Our proposed MSG-SVM algorithm is a gossip-based algorithm, where the support vectors $SV_i(0)$ are communicated between one-hop neighbors. Therefore, for each node i, at time $t+1$, we update its estimate $\mathbf{w}_i(t+1)$ by using all the information available at that moment, namely, the previously estimated set of support vectors $SV_i(t)$ at node i, as well as the union of the sets of support vectors $SV_{N_i}(t)$ that have been previously estimated by the neighbor nodes. Notice that once we decide (at a given step $t+1$) what is the new set of support vectors $SV_i(t+1)$ at a given node i, this determines uniquely the corresponding estimate of the hyperplane $\mathbf{w}_i(t+1)$, so there is a one-to-one mapping. A description of the algorithm for each sensor i is the following:

INPUT

- *Data set S_i contains the measurements collected by the i-th sensor.*

PROCEDURE

1. *Initialize time slot t=0, and the set $S_i := S_i(0)$.*
2. *Train the SVM on the current data set S_i, and obtain optimal hyperplane $\boldsymbol{w}_i^*(t)$ ($\equiv$ set of support vectors $SV_i(0)$).*
3. *Transmit $SV_i(t)$ to neighboring sensors N_i. To save power, transmit only those vectors that were not transmitted to neighboring sensors in previous time slots.*
4. *Update $S_i(t+1) = \{SV_i(t) \cup SV_{N_i}(t)\}$.*
5. *Increment t, and return to Step 2, if $SV_i(t) \neq \emptyset$.*

OUTPUT

- *Hyperplane $\boldsymbol{w}_i^*(t)$ that classifies the data at sensor i.*

The proposed algorithm seems well suited for the distributed training of a SVM in a WSN. To begin with, MSG-SVM is concurrent for each sensor, so finally all n sensors get the measurements that are characterized as support vectors in the n sub-problems. Hence all sensors converge to the same discriminant constructed by those support vectors. Therefore, WSN nodes converge to the same trained SVM classifier, and can classify any new measurements. Additionally, it is an energy efficient algorithm since in order to reduce the energy consumption, each sensor transmits to its neighbors only the support vectors that have not been transmitted in previous steps.

On the other hand, it can be shown that MSG-SVM provides a sub-optimal discriminant hyperplane, with respect to a global centralized algorithm, while communicating the minimum necessary information at each step. As we already mentioned, the data of a node can be compressed to their corresponding support vectors. But it cannot be guaranteed that a vector $\mathbf{x}$ such that $\mathbf{x} \in M_i(t)$ and $\mathbf{x} \notin SV_i(t)$, is not a support vector in $M_i(t+1) = \{M_i(t) \bigcup_{j \in N_i} M_j(t)\}$. In other words, at each step, the set of support vectors associated with the entire data set is not always the same as the overall union of the support vectors obtained after training separately each of the two sets.

Lemma 4.1.1 *The MSG-SVM algorithm is sub-optimal, that is, the consensus achieved by training the SVM using only the support vectors from each of the sub-problems is sub-optimal.*

Proof. We only need to find a case where a support vector in the training set is not a support vector in any sub-problem. Consider the case of a network comprised of only two

sensors collecting measurements in two dimensions. Sensor 1 collects a set of measurements S_1 and the other one collects a set of measurements S_2. For the geometrical aspect of this problem, we need to find a vector in the union of the measurements of both sensors $S = S_1 \cup S_2$ that is a support vector in S, but not in set S_1 nor in set S_2. Let the smallest distance between the two convex hulls of set S_1 be d_1 and the corresponding distance of set S_2 be d_2, (cf. Figure 4.1). The convex hull of one class of set S is the smallest set that contains the measurements of set S, hence it also contains the convex hulls of the same class of sets S_1 and S_2. As a counter example one can find a point in the convex hull of set S that is a support vector but it is not a support vector in S_1 nor in S_2. In Figure 4.2, the squared point is a support vector in S but neither a support vector in S_1 nor in S_2, since the distance between the convex hulls is d, where $d < d_1$ and $d < d_2$. ■

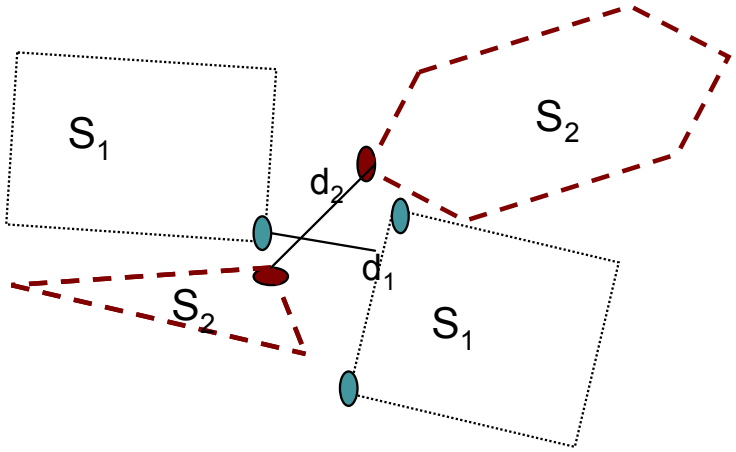


Figure 4.1: The closest distance between the two dotted convex hulls of S_1 is d_1. Thus, the circled points on the boundary of the convex hulls are the support vectors in set S_1. The closest distance between the two dashed convex hulls of S_2 is d_2. Thus, the circled points on the boundary of the convex hulls are the support vectors in set S_2.

4.1.2 Sufficient Selective Gossip Algorithm (SSG-SVM)

We propose an alternative algorithm, the Sufficient Selective Gossip Algorithm (SSG-SVM), in order to eliminate the possibility of not converging, such as in MSG-SVM, to the optimal solution. Each sensor sends the amount of data to the one-hop neighbors that guarantees convergence to the optimal solution. Which is the sufficient amount of data that sensors should exchange to converge to the optimal solution while training a SVM?

We can exploit the geometrical description of a SVM training, illustrated in Section 2.4, Figure 2.17. We examine the convex hull of the training data of each class, and construct the plane that bisects the two closest points of the convex hulls, [70]. This is an alternative equivalent perspective for training a SVM. The closest points can be found by solving the following dual quadratic problem:

$$\min_{\alpha} \frac{1}{2} \parallel \mathbf{c} - \mathbf{d} \parallel^2 \tag{4.2}$$

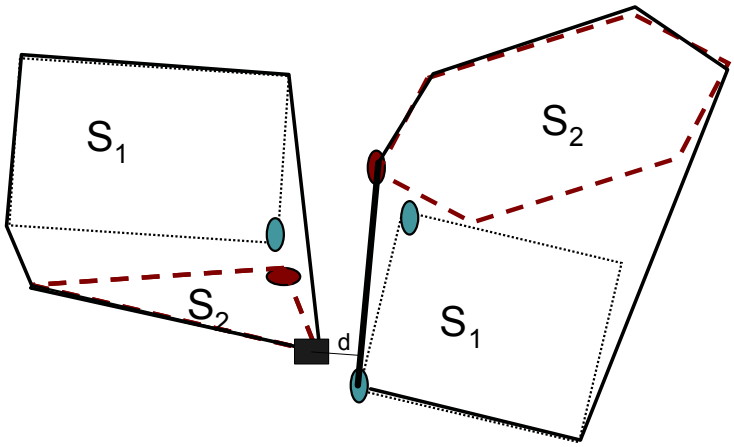


Figure 4.2: The convex hull depicted in solid line is the convex hull of set $S = S_1 \cup S_2$. The squared point on the convex hull is a support vector in S, since it is one of the closest points between the two convex hulls. Notice that this is not a support vector in S_1 nor in S_2.

$$\mathbf{c} = \sum_{y_i \in class\ 1} \alpha_i x_i, \quad \mathbf{d} = \sum_{y_i \in class\ -1} \alpha_i x_i,$$

subject to

$$\begin{array}{rcl} \sum_{y_i \in class\ 1} \alpha_i & = & 1, \ \sum_{y_i \in class\ -1} \alpha_i = 1 \\ \alpha_i & \geq & 0 \ \text{for } i = 1, 2, ...n. \end{array}$$

SSG-SVM takes advantage of the geometrical property of the SVM discriminant hyperplane. The sufficient amount of data for the hyperplane construction are the vectors that lie on the boundary of the convex hulls of the two classes. For each node, the SSG-SVM discards all the vectors of the WSN nodes, except those located at the boundary of the convex hulls. Thus, neighboring sensors exchange the sufficient data only. After some communication, all WSN nodes have the information to construct a separating plane identical to the plane that would have been constructed if all sensors had access to the entire information. A description of the algorithm for each sensor i is the following:

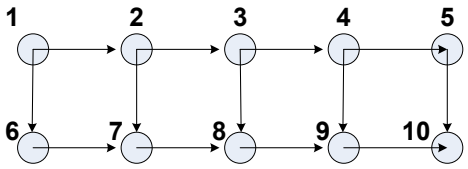


Figure 4.3: The sensor network is composed of $n = 10$ nodes distributed in a grid topology. The communication links for each sensor are depicted with arrows.

INPUT

- *Data set S_i contains the measurements collected by the i-th sensor.*

PROCEDURE

1. *Initialize time slot t=0, and the set $S_i := S_i(0)$.*
2. *Train the SVM on the current data set S_i, and obtain optimal hyperplane $\boldsymbol{w}_i^*(t)$. Calculate set $CONV_{S_i}(0)$, i.e., vectors lying on the convex hulls at time slot 0.*
3. *Transmit set $CONV_{S_i}$ to neighboring sensors N_i. To save power, transmit only those vectors that were not transmitted to neighboring sensors in previous time slots.*
4. *Update $S_i(t+1) = \{SV_i(t) \cup CONV_{N_i}(t)\}$.*
5. *Increment t, and return to Step 2, if $CONV_{N_i}(t) \neq \emptyset$.*

OUTPUT

- *Hyperplane $\boldsymbol{w}_i^*(t)$ that classifies the data at sensor i.*

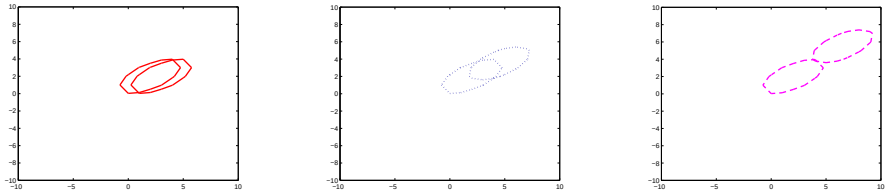
Both algorithms are energy efficient, since data need not be transmitted to a fusion center and the amount of data exchanged by sensors is substantially smaller than the overall generated data. Instead, WSN nodes diffuse partial information to neighboring sensors. Furthermore, each node communicates only information that has not been sent previously, thus the energy spent for transmission is reduced.

4.2 Results and Discussion

In this Section, we evaluate the performance of the two proposed distributed algorithms in terms of the average classification error rate and we compare them to the ideal case where WSN nodes have access to the entire information.

We consider a sensor network composed of $n = 10$ nodes distributed in a grid topology, where each sensor i, $i = 1, \ldots, 10$, collects $\mid M_i(0) \mid = 14$ sample vectors from two classes, at each step. WSN nodes communicate with their one-hop neighbors. The communication links between the sensors in the network are depicted in Figure 4.3. In our experiments, we generate three sample data sets of two different classes each, using Gaussian distributions with two different means. We choose three data sets, such that their Mahalanobis distance[1] is increasing, Figure 4.4.

[1]For linear classifiers the Mahalanobis distance is given by the expression $d = (\mu_2 - \mu_1)^T(\frac{\Sigma_1+\Sigma_2}{2})^{-1}(\mu_2 - \mu_1)$, where μ_1, μ_2 are the mean values and Σ_1, Σ_2 the covariance matrices of the two distributions, respectively.



(a) Data set 1 (b) Data set 2 (c) Data set 3

Figure 4.4: The representation ellipses of different data sets generated by Gaussian distributions.

We simulated 100 Monte Carlo runs in order to test the performance of the two proposed Selective Gossip Algorithms. Figure 4.5 represents the average classification error rates (%) for a randomly chosen sensor, as a function of the iteration steps. After only a few iterations, both algorithms result in trained SVM classifiers which exhibit similar performance to a centralized SVM trained using the entire data from all sensors. SSG-SVM gives an optimal estimate of the discriminant after at most 8 iterations using only partial data. The small divergence of SSG-SVM from the optimal solution (only 2% on average), can be diminished by tuning the parameters of the optimization problem (4.2). On the other hand, even though MSG-SVM is a sub-optimal solution, it gives a good approximation of the optimal separating plane. Most importantly, with both introduced distributed schemes, all n sensors reach, with a small finite number of steps, an agreement on the nearly optimal discriminant function. Both proposed algorithms behave similarly for all data sets.

The results also show that the difference in performance between MSG-SVM and SSG-SVM is very small. This happens because sensors collect measurements from the same distribution. Therefore, it is very rare to encounter the case where a measurement that is not a support vector in the data set of one sensor, happens to be a support vector in a set containing data from all sensors. In other words, the counter example in Figure 4.2 is actually an event of low probability; however, such an event may occur more often in scenarios where the class distributions are time-varying.

Moreover, we have also analyzed the trade-off between classification accuracy and energy consumption. Figure 4.6 illustrates the number of measurements that a particular sensor (tested in data set 2) transmits to its neighbors at each iteration. MSG-SVM gives a sub-optimal solution but uses less measurements than SSG-SVM, thus less energy. SSG-SVM, on the other hand, transmits more data at each iteration, in order to ensure optimality. One can notice that after 5 iterations, in both algorithms, nodes do not need to send any more measurements to their neighbors.

After *gossiping* in the network, WSN nodes have exchanged in previous steps all the necessary measurements. Hence, only after a few iterations sufficient amount of data has been diffused to all WSN nodes, each of whom can construct the same trained SVM with the minimum classification error.

4.3 Conclusions

In this Chapter, we propose two distributed selective gossip algorithms for training a SVM in a Wireless Sensor Network, based on successive refinement of local estimates. In both cases, information is communicated to one-hop neighbors in order to update the estimate

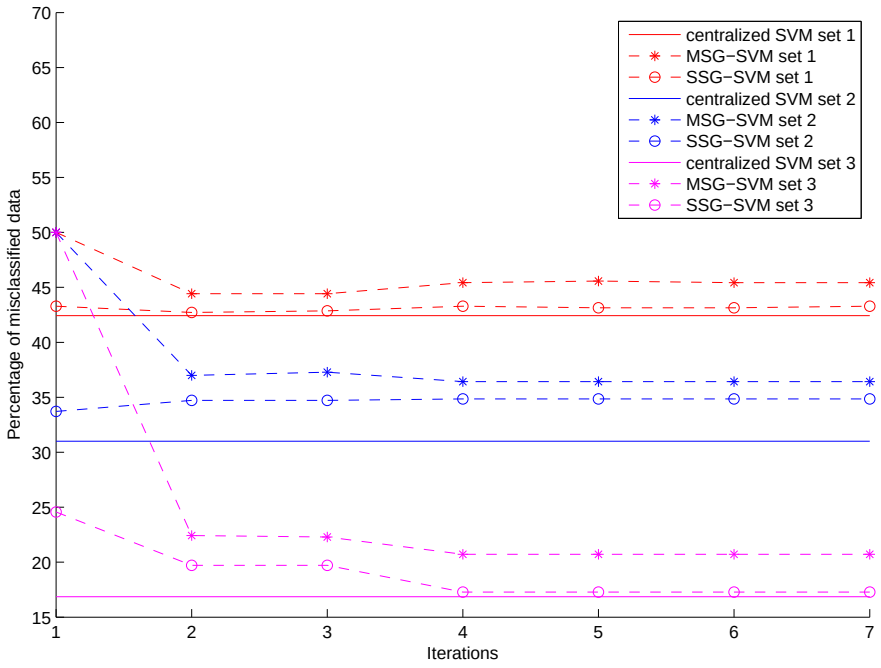


Figure 4.5: Performance, at a given particular sensor, of the training algorithms for three different data sets. SSG-SVM gives an optimal estimate of the discriminant after at most 4 iterations. MSG-SVM is a suboptimal solution but gives a good approximation of the optimal plane. The ideal case where sensors have access to the entire data is depicted by the straight line.

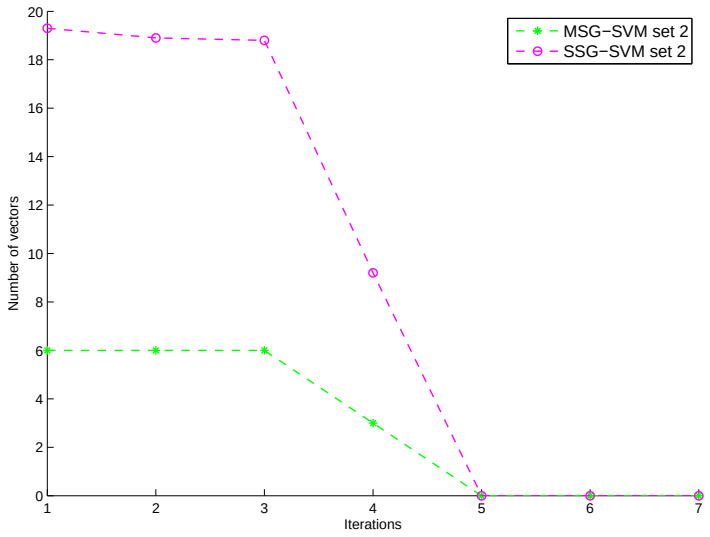


Figure 4.6: MSG-SVM gives a sub-optimal solution using less measurements than SSG-SVM, which reaches optimality using more data at each iteration.

at each iteration. The sub-optimal algorithm MSG-SVM, uses only the support vectors of each node to reach an agreement. The SSG-SVM, on the other hand, communicates larger amount of data, *i.e.*, vectors lying on the convex hull boundaries, but converges closer to the optimal solution in a few iterations.

Chapter 5

Optimality for Gossip-based Distributed SVM training

Contents

In this Chapter, we derive a novel mathematical characterization for the sparse representation of the most important measurements that neighboring sensors should exchange in order to reach an agreement to the optimal SVM classifier, [71]. We propose a selection function which ranks the training vectors in order of importance in the learning process. The amount of information exchange can vary, based on an appropriately chosen threshold value of the selection function, providing a desired trade-off between classification accuracy and power consumption. Through simulation experiments, we show that even though the proposed algorithm uses partial information for inter-node communication, all sensors converge to the same hyperplane obtained using a centralized SVM classifier that employs the entire sensor data at a fusion center. Tuning appropriately the threshold, the network can converge to the optimal solution, or to an estimate close to the optimal solution. We finally generalize our convergence conclusions for n- dimensional feature vectors and for a random network topology.

5.1 Adaptive Consensus SVM training

In the previous Chapter, we discussed the framework of the gossip-based distributed training of an SVM classifier by means of data sets collected with a WSN. Consider now a simple case where the network is composed of two sensors. Each sensor collects n and m

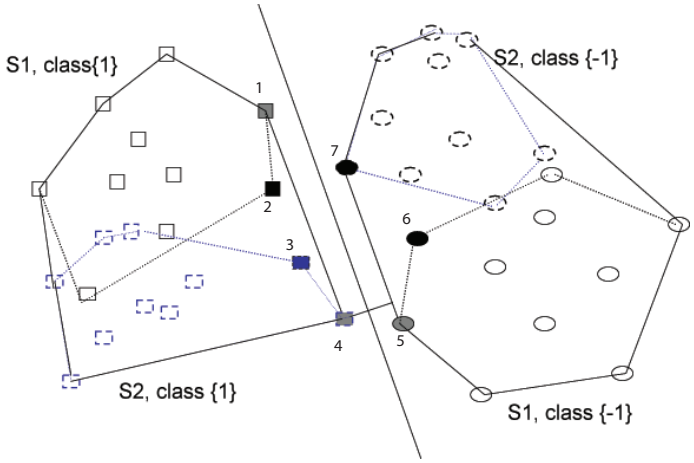


Figure 5.1: Example of a case where the global SVM solution makes use of a vector which is not a support-vector in neither of the two subproblems associated to each of the sensors. Vectors $1, 4$ and 5, become support vectors of the global SVM solution.

measurements from each class respectively. We define these two sets as,

$$S_1 := \{\mathbf{x}_1^{(1)}, \mathbf{x}_2^{(1)}, \ldots, \mathbf{x}_n^{(1)}, \mathbf{y}_1^{(1)}, \mathbf{y}_2^{(1)}, \ldots, \mathbf{y}_m^{(1)}\}$$
$$S_2 := \{\mathbf{x}_1^{(2)}, \mathbf{x}_2^{(2)}, \ldots, \mathbf{x}_n^{(2)}, \mathbf{y}_1^{(2)}, \mathbf{y}_2^{(2)}, \ldots, \mathbf{y}_m^{(2)}\}$$

We want to train a SVM, and therefore classify all the measurements in the network. In the ideal case with no power constraints in the network, each sensor sends their data to a fusion center where the SVM is trained on the whole data set $S := S_1 \cup S_2$, and the separating hyperplane is constructed from the support vectors. Let SV_S be the set that contains the support vectors obtained when the training is performed with all the data (centralized case). Clearly, $SV_S \subset S$. Moreover, notice that if $\mathbf{s} \in SV_S$, then $\mathbf{s} \in S_1$ or $\mathbf{s} \in S_2$.

Since the measurements of each node can be represented by their associated support vectors, one could expect that the support vectors are the sufficient data, *i.e,* the sufficient statistic, to send to the neighboring sensors in order to construct the optimal hyperplane (MSG-SVM). However, an important observation is that in general, a support vector in the centralized case, *i.e,* on the whole set S, might not be a support vector in the subproblems, *i.e,* on the sets $S_1, \ldots S_N$, or in other words $SV\Big(\bigcup_{i=1}^{N} S_i\Big) \neq SV\Big(\bigcup_{i=1}^{N} SV(S_i)\Big)$.

Transmitting only the support vectors is an obvious scheme for distributed learning although convergence to the optimal is not guaranteed. Caragea et al. give another mathematical example for the same purpose [72]. Syed et al. on the other hand, showed through simulation experiments that the support vectors chosen by the SVM algorithm is a minimal set and removing any more samples would result in the loss of vital information about the class distribution [73]. Therefore, even though Vandenberghe et al. claim in [61] that this scheme converges to the optimal classifier, this cannot be generalized for every data distribution.

Yet, it has been proved that $conv\Big(\bigcup_{i=1}^{N} S_i\Big) = conv\Big(\bigcup_{i=1}^{N} conv(S_i)\Big)$, where $conv$ is the convex set of set S_i, $i = 1, \cdots, N$, [74]. Hence, the convex hulls of two sets that belong to the two classes represent sufficient statistics for learning SVMs from distributed data, [72]. Therefore, the important vectors in order to achieve optimality, *i.e.,* the vectors that become

support vectors in the centralized case, lie on the facets of the convex hulls. For example, in Figure 5.1 the important vectors are vectors $1, 4$ and 5 which lie on the same facet with the local support vectors $2, 3, 6$ and 7. Although this inference cannot be generalized for high dimensional convex sets or for all data distributions, from the geometrical formulation of the SVM problem it can be inferred that except of the support vectors (which are the closest vectors to the hyperplane), vectors that are "close" to the hyperplane should be exchanged among neighboring sensors.

On the other hand, the complexity of the convex hull computation has a linear dependence on the number of facets of the convex hull and the number of facets can be exponential in the dimension of the space [74]. The energy-limited capacity of the sensors and the high dimensional data that sensors usually collect in real life applications, make this approach likely to be practical only when the convex hulls are simple (have few facets), but not in general.

The question at this point is what kind of information should neighboring sensors exchange in order to get the best possible classification accuracy while keeping the power consumption low. In the previous Chapter, we proposed two distributed algorithms, a) the MSG-SVM, where the minimum amount of data, corresponding to the support vectors of each node, is selected for diffusion and b) the SSG-SVM, where sufficient data corresponding to the vectors defining the convex hull of each class, is diffused to achieve optimality, that is, same performance as the one by a global centralized algorithm. MSG-SVM provides a sub-optimal solution, while SSG-SVM achieves optimality but with a greater power cost.

In the following, we first give an intuition for the optimal characterization of the measurements during gossiping among sensors, for the convergence to the optimal estimate of the classifier. Then, we define a novel function, the so-called *Selection Function*, that ranks the measurements in order of importance during the learning process. Finally, we discuss the geometrical interpretation of the selection function and we study how the choice of the threshold on this function affects the trade-off between learning accuracy and power computation.

5.1.1 Motivation

In the previous Section, we discussed about data exchange during distributed training of a SVM. Communicating more data during gossiping among sensors, the network reaches convergence to the optimal estimate of the classifier. In Figure 5.1, we gave an example that clearly shows that when only the support vectors are exchanged during gossiping (MSG-SVM), the network reaches a sub-optimal solution. Therefore additional data should be exchanged in order to achieve convergence to the optimal solution. Observing Figure 5.1, important vectors, besides support vectors, are vectors lying close to the hyperplane. More specifically, vectors lying on the facets of the convex hulls (SSG-SVM) provide the sufficient statistics for optimality, (cf. Section 5.1). We enumerate the reasons that necessitate a further exploration of data exchange during distributed training of a SVM.

- As mentioned in Section 5.1, the SSG-SVM algorithm is not applicable in real life applications where sensors collect high dimensional data, because of the complexity of the construction of the convex hulls.
- Even in the case of low dimensionality, there exist vectors that carry extra information that is not necessary for optimality, *i.e.,* vectors lying away from the hyperplane.
- The energy-limited capacity of sensors forces the need for optimal selection of data. Therefore, the amount of data to be exchanged must be selected depending on a desired trade-off between energy consumption and classification accuracy.

- Intuitively, one can say that the important vectors are the ones lying close to the hyperplane. Yet, this heuristic approach is not flexible and powerful in general, and also lacks systematic formulation in mathematics. Thus, there is a need for an optimization learning approach, through which the amount of data that sensors should exchange is selected in a systematic way, given a desired trade-off between classification accuracy and power consumption.

5.1.2 The Selection Function

In the framework of distributed SVM training, each sensor trains a SVM with its current data locally. Training the SVM involves solving the optimization problem (2.64), where $\{\mathbf{x}_i, \mathbf{y}_j\},\ i = 1,\ldots,n,\ j = 1,\ldots,m$, are the measurements from two different classes collected by the sensor. Hence, each sensor determines the optimal Lagrange multipliers $(\boldsymbol{\theta}^*, \boldsymbol{\gamma}^*)$, that correspond to the support vectors of each class. Therefore, after local SVM training the variables $(\boldsymbol{\theta}^*, \boldsymbol{\gamma}^*)$ and their corresponding measurements $\{\mathbf{x}_i, \mathbf{y}_j\},\ i = 1,\ldots,n,\ j = 1,\ldots,m$, are known for each sensor. We formally define the selection function for each measurement $\mathbf{x}_i$ and $\mathbf{y}_j$ from both classes respectively.

Difinition 1 *The* Selection Function $F(\boldsymbol{x}_i)$ *is given by the following expressions for each measurement* $\boldsymbol{x}_i$*, (similarly for* $\boldsymbol{y}_j$*):*

$$F(\boldsymbol{x}_i) = 2\textstyle\sum_k \theta_k^* < \boldsymbol{x}_i, \boldsymbol{x}_k > -2\sum_l \gamma_l^* < \boldsymbol{x}_i, \boldsymbol{y}_l >$$

$$F(\boldsymbol{y}_j) = 2\textstyle\sum_l \gamma_l^* < \boldsymbol{y}_l, \boldsymbol{y}_j > -2\sum_i \theta_i^* < \boldsymbol{x}_i, \boldsymbol{y}_j > .$$

The selection function can be calculated for each measurement vector locally at each sensor, that is, the corresponding processing is distributed. In the following Section, we provide the geometrical interpretation of the selection function and we show that it can be used for ranking the training vectors in order of importance in the learning process. Here, we provide the detailed mathematical development of the selection function.

Consider the optimization problem as defined by (2.64). The Lagrangian of (2.64) is expressed as follows:

$$\begin{aligned} L(\boldsymbol{\theta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\xi}, \boldsymbol{\psi}, \nu_1, \nu_2) = &< \sum_{i=1}^{n} \theta_i \mathbf{x}_i - \sum_{j=1}^{m} \gamma_j \mathbf{y}_j,\ \sum_{k=1}^{n} \theta_k x_k - \sum_{l=1}^{m} \gamma_l y_l > \\ &- \sum_{i=1}^{n} \lambda_i \theta_i - \sum_{j=1}^{m} \mu_j \gamma_j + \sum_{i=1}^{n} \xi_i(\theta_i - \delta) \\ &+ \sum_{j=1}^{m} \psi_j(\gamma_j - \delta) + \nu_1\Big(\sum_{i=1}^{n} \theta_i - 1\Big) + \nu_2\Big(\sum_{j=1}^{m} \gamma_j - 1\Big). \end{aligned} \tag{5.1}$$

where $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\xi}, \boldsymbol{\psi})$ are the optimal Lagrange multipliers (dual variables) corresponding to the inequality constraints, and (ν_1, ν_2) are the Lagrange multipliers corresponding to the equality constraints of (2.64).

Since this problem is convex, optimality is achieved when the *Karush Kuhn Tucker* (KKT) conditions are satisfied [43]. In other words, if $(\boldsymbol{\theta}^*, \boldsymbol{\gamma}^*)$ are the optimal values, then

for $i = 1, \ldots, n$, and $j = 1, \ldots, m$:

$$\begin{aligned}
-\theta_i^* &\leq 0, \\
-\gamma_j^* &\leq 0, \\
\sum_{i=1}^{n} \theta_i^* &= 1, \\
\sum_{j=1}^{m} \gamma_j^* &= 1, \\
\lambda_i^* &\geq 0, \\
\mu_j^* &\geq 0, \\
\xi_i^* &\geq 0, \\
\psi_j^* &\geq 0, \\
\nabla L(\boldsymbol{\theta}^*, \boldsymbol{\gamma}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*, \boldsymbol{\xi}^*, \boldsymbol{\psi}^*, \nu_1^*, \nu_2^*) &= 0, \\
-\lambda_i^* \theta_i^* &= 0, \\
-\mu_j^* \gamma_j^* &= 0, \\
\xi_i^* (\theta_i - \delta) &= 0, \\
\psi_j^* (\gamma_j - \delta) &= 0,
\end{aligned}$$

The last four equalities are known as the *complementary slackness conditions*. Setting the partial derivatives of the Lagrangian in 5.1 with respect to θ_i and γ_j equal to zero we get that $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ and $\boldsymbol{\gamma} = \boldsymbol{\gamma}^*$, the KKT conditions must hold, which results in :

$$0 = 2\theta_i^* \parallel \mathbf{x}_i \parallel^2 + 2 \sum_{k \neq i} \theta_k^* < \mathbf{x}_i, \mathbf{x}_k > -2 \sum_{l} \gamma_l^* < \mathbf{x}_i, \mathbf{y}_l > -\lambda_i^* + \xi_i^* + \nu_1^*, \tag{5.2}$$

$$0 = 2\gamma_j^* \parallel \mathbf{y}_j \parallel^2 + 2 \sum_{l \neq j} \gamma_l^* < \mathbf{y}_l, \mathbf{y}_j > -2 \sum_{i} \theta_i^* < \mathbf{x}_i, \mathbf{y}_j > -\mu_j^* + \psi_j^* + \nu_2^*. \tag{5.3}$$

From complementary slackness, at optimality $\lambda_i^* \theta_i^* = 0$, $\mu_j^* \gamma_j^* = 0$, $\xi_i^* (\theta_i^* - \delta) = 0$, $\psi_j^* (\gamma_j^* - \delta) = 0$. Therefore $\lambda_i^* = 0$ when $\theta_i^* \neq 0$ and $\xi_i^* = 0$ when $\theta_i^* = 0$, $i = 1, \ldots, n$. Similarly for the other class, $\mu_j^* = 0$ when $\gamma_j^* \neq 0$ and $\psi_j^* = 0$ when $\gamma_j^* = 0$, $j = 1, \ldots, m$. In conclusion, if $\mathbf{x}_i, \mathbf{y}_j$ are not support vectors, *i.e.,* when $\theta_i^* = 0$, then (5.2), (5.3) become:

$$2 \sum_{k \neq i} \theta_k^* < \mathbf{x}_i, \mathbf{x}_k > -2 \sum_{l} \gamma_l^* < \mathbf{x}_i, \mathbf{y}_l > -\lambda_i^* + \nu_1^* = 0,$$

$$2 \sum_{l \neq j} \gamma_l^* < \mathbf{y}_l, \mathbf{y}_j > -2 \sum_{i} \theta_i^* < \mathbf{x}_i, \mathbf{y}_j > -\mu_j^* + \nu_2^* = 0.$$

On the other hand, if $\mathbf{x}_i, \mathbf{y}_j$ are support vectors, *i.e.,* when $\theta_i^* \neq 0$, then (5.2), (5.3) become:

$$2\theta_i^* \parallel \mathbf{x}_i \parallel^2 + 2 \sum_{k \neq i} \theta_k^* < \mathbf{x}_i, \mathbf{x}_k > -2 \sum_{l} \gamma_l^* < \mathbf{x}_i, \mathbf{y}_l > +\xi_i^* + \nu_1^* \; = \; 0,$$

$$2\gamma_j^* \parallel \mathbf{y}_j \parallel^2 + 2 \sum_{l \neq j} \gamma_l^* < \mathbf{y}_l, \mathbf{y}_j > -2 \sum_{i} \theta_i^* < \mathbf{x}_i, \mathbf{y}_j > +\psi_j^* + \nu_2^* \; = 0 \; .$$

This mathematical analysis concludes to the definition of the selection function $F(\mathbf{x}_i)$ and $F(\mathbf{y}_j)$ for each vector $\mathbf{x}_i$ and $\mathbf{y}_j$, respectively as follows:

$$F(\mathbf{x}_i) = 2\theta_i^* \parallel \mathbf{x}_i \parallel^2 +2\sum_{k\neq i} \theta_k^* < \mathbf{x}_i, \mathbf{x}_k > -2\sum_{l} \gamma_l^* < \mathbf{x}_i, \mathbf{y}_l >, \tag{5.4}$$

$$F(\mathbf{y}_j) = 2\gamma_j^* \parallel \mathbf{y}_j \parallel^2 +2\sum_{l\neq j} \gamma_l^* < \mathbf{y}_l, \mathbf{y}_j > -2\sum_{i} \theta_i^* < \mathbf{x}_i, \mathbf{y}_j > . \tag{5.5}$$

This selection function can be calculated for each measurement vector locally at each sensor, that is, the corresponding processing is distributed. In the following Section, we provide the geometrical interpretation of the selection function and we show that it can be used for ranking the training vectors in order of importance in the learning process.

5.1.3 Geometrical Interpretation of the Selection Function

In Section 2.4, we emphasized that in SVMs, the resulting separating hyperplane is expressed by means of a linear combination of the support vectors, (cf. Equation (2.58)). The support vectors are the closest points to the hyperplane among all the measurements of each class. In other words, the important vectors are the ones lying close to the hyperplane. The following theorem allows us to use the selection function for ranking the measurements in order of importance during the learning process.

Theorem 5.1.1 *The selection function is monotonically increasing with respect to the distance of a measurement from the hyperplane.*

Proof.

The distance of a measurement $\mathbf{x}$ of class {1} from the optimal hyperplane is given by:

$$d^2(\mathbf{x}, \mathbf{w}^*) = \frac{\parallel \mathbf{w}^{*T}\mathbf{x} + b^* \parallel^2}{\parallel \mathbf{w}^* \parallel^2}, \tag{5.6}$$

where $\mathbf{w}^*$ and b^* are the variables that define the hyperplane. For a measurement $\mathbf{x}$ of class {1}, since the hyperplane is determined by (2.58), it follows that:

$$\mathbf{x}^T \cdot \mathbf{w}^* = \mathbf{x}^T \cdot (\sum_{i=1}^{n^*} \theta_i^* \mathbf{x}_i - \sum_{j=1}^{m^*} \gamma_j^* \mathbf{y}_j) = \frac{1}{2} F(\mathbf{x}). \tag{5.7}$$

Therefore, from (5.6) and (5.7), it can be concluded that:

$$d^2(\mathbf{x}, \mathbf{w}^*) = \frac{(F(\mathbf{x})/2 + b^*)^2}{\parallel \mathbf{w}^* \parallel^2}. \tag{5.8}$$

For a fixed $b^* \in R$, the function $(F(\mathbf{x})/2+b^*)^2$ is monotonically increasing since by definition $\mathbf{w}^{*T}\mathbf{x} + b^* > 1$. Similarly, for a vector y from class {-1} it can be shown that:

$$\mathbf{y}^T \mathbf{w}^* = -\frac{1}{2} F(\mathbf{y}), \tag{5.9}$$

and

$$d^2(\mathbf{y}, \mathbf{w}^*) = \frac{(F(\mathbf{y})/2 - b^*)^2}{\parallel \mathbf{w}^* \parallel^2} \tag{5.10}$$

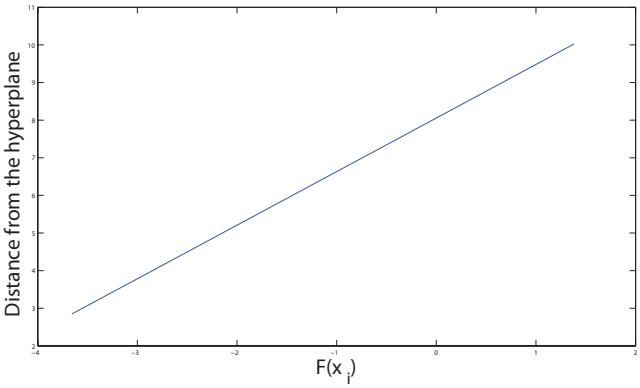


Figure 5.2: $F(\mathbf{x}_i)$ increases as the distance of $\mathbf{x}_i$ from the hyperplane increases. Hence the measurements that are closer to the hyperplane have smaller values of the selection function.

Again, the function $(F(\mathbf{y})/2 - b^*)^2$ is monotonically increasing for a fixed $b^* \in R$, since ${\mathbf{w}^*}^T\mathbf{y} + b^* < -1$.

Therefore, as the value of $F(\mathbf{x})$ increases, so does $d^2(\mathbf{x}, \mathbf{w}^*)$. Since $d(\mathbf{x}, \mathbf{w}^*) > 1$ by definition, then as $d^2(\mathbf{x}, \mathbf{w}^*)$ increases, the distance increases as well. Similarly for class {-1}, as the value of $F(\mathbf{y})$ increases, so does the distance of $\mathbf{y}$ from the hyperplane. ■

We draw the same conclusions through simulation experiments. Consider one sensor taking 10 measurement pairs $\{(\mathbf{x}_i, \mathbf{y}_j)\}$ from two linearly inseparable Gaussian distributions. This node trains a SVM and constructs a discriminant of the data set in two classes. Figure 5.2 illustrates the variation of the selection function $F(\mathbf{x}_i)$ versus the distance of measurement $\mathbf{x}_i$ from the hyperplane obtained at the local training, after 100 Monte Carlo runs. We obtain similar results for the variation of $F(\mathbf{y}_j)$ with respect to the distance of each measurement $\mathbf{y}_j$ from the hyperplane. It is clear that the value of the selection function is directly proportional to the distance of the data points from the SVM hyperplane.

In other words, by calculating the selection function for each measurement, we have information about the distance of each measurement to the hyperplane (cf. Figure 5.3). Notice that the support vectors which by definition are the measurements closer to the discriminant, have the smallest value of the selection function. Clearly this way, all the training vectors can be ranked in order of importance in the learning process.

5.1.4 Thresholds

In real life applications, one can determine a priori the percentage of the measurements of each sensor they can afford to use at each iteration. This percentage corresponds to threshold values, F_1 and F_2 for classes $\{1\}$, $\{-1\}$ respectively. In the general case where sensors collect a different number of measurements, we can determine different thresholds for each sensor k as $F_1^{(k)}$ and $F_2^{(k)}$, for both classes respectively. Given the thresholds $F_1^{(k)}$ and $F_2^{(k)}$, we choose to transmit to the neighboring sensors those vectors with selection function value less than this threshold (cf. Figure 5.3).

Given a desired trade-off between classification accuracy and power consumption, the amount of information exchange can vary, based on user-defined threshold values $F_1^{(k)}$ and

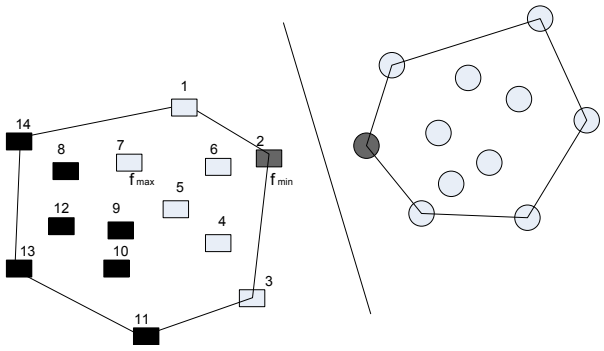


Figure 5.3: All training vectors are ranked through the selection function as following: $F(x_2) < F(x_6) < F(x_1) < F(x_3) < F(x_4) < F(x_5) < F(x_7) < F_1$. Vector x_2 is a support vector and hence it has the minimum value. The rest vectors are discarded because $F(x_i) > F_1$, for $i = 8, 9, 10, 11, 12, 13, 14$.

$F_2^{(k)}$ of the respective selection functions. The larger the values of the thresholds, the more data will be exchanged among neighboring sensors. The range of the values of the threshold for sensor k is $F_{1min} \leq F_1^{(k)} \leq F_{1max}$, and $F_{2min} \leq F_2^{(k)} \leq F_{2max}$. Notice that for simplicity, we assume that the sample vectors of each class are enumerated such that the support vectors correspond (in any pre-agreed order) to the first l and m sample vectors, respectively. The next Lemma follows immediately from Theorem 5.1.1.

Lemma 5.1.2 *Among all measurements (vectors) of the same class, the support vectors minimize the selection function.*

Proof. Support vectors SV_k by definition, are the closest vectors to the hyperplane. Thus, since $F(\mathbf{x})$ is monotonically increasing with respect to the distance of a measurement from the hyperplane, the minimum value of $F(\mathbf{x})$ is taken at $\{\mathbf{x} : \mathbf{x} \in SV_k\}$. Similarly for $F(\mathbf{y})$. In other words $F_{1min} = F_l^{(k)}$ and $F_{2min} = F_m^{(k)}$. ■

From the above Lemma it follows that the range of the threshold values is $F_l^{(k)} \leq F_1^{(k)} \leq F_{1max}$, and $F_m^{(k)} \leq F_2^{(k)} \leq F_{2max}$. Notice that for $F_{1min} = F_l^{(k)}$ and $F_{2min} = F_m^{(k)}$, the ASG-SVM coincides with the MSG-SVM.

5.1.5 The ASG-SVM Algorithm

In this Section, we introduce the Adaptive Selective Gossip (ASG-SVM) algorithm. Consider a network where communication links are represented by a graph whose vertices are the sensors and whose edges are formed by the available communication links. The set of sensors having an active link with the k-th sensor is denoted as the neighborhood N_k.

Given threshold values $F_1^{(k)}$ and $F_2^{(k)}$, the set of vectors to be transmitted and exchanged by a certain sensor at the initial time step $t = 0$, is determined as: $Selected\_Set_k(0) = \{\mathbf{x}_i : F(\mathbf{x}_i) < F_1, i = 1, \ldots, n\} \cup \{\mathbf{y}_j : F(\mathbf{y}_j) < F_2, j = 1, \ldots, m\}$. The vectors of the set $Selected\_Set_k(t)$ selected by each sensor at time t, are communicated to all its one-hop neighbors. Therefore, for each node k, at time $t+1$, we update its estimate $\mathbf{w}_k^*(t+1)$ by using all the information available at that moment, namely, the previously estimated set of the support vectors $SV_k(t)$ at node k, as well as the union of the sets $Selected\_Set_{N_k}(t)$ that have been previously generated by the neighboring nodes using the selection functions.

Notice that each sensor can apply the selection functions in a completely autonomous manner. A description of the algorithm for each sensor k is the following:

INPUT

- *Data set S_k contains the measurements collected by the k-th sensor.*
- *Set the thresholds $F_1^{(k)}$ and $F_2^{(k)}$.*

PROCEDURE

1. *Initialize time slot t=0 and the set $S_k := S_k(0)$.*
2. *Train the SVM on the current data set S_k, and obtain optimal hyperplane $\boldsymbol{w}_k^*(t)$ ($\equiv$ set of support vectors $SV_k(t)$).*
3. *Calculate $F(\boldsymbol{x}_i)$ and $F(\boldsymbol{y}_j)$ for the measurements $\boldsymbol{x}_i$ of class $\{1\}$ and $\boldsymbol{y}_j$ of class $\{-1\}$, respectively.*
4. *Determine $Selected\_Set_k(t)$*
5. *Transmit $Selected\_Set_k(t)$ to neighboring sensors N_k. To save power, transmit only those vectors that were not transmitted to neighboring sensors in previous time slots.*
6. *Update $S_k(t+1) = \{SV_k(t) \cup Selected\_Set_{N_k}(t)\}$.*
7. *Increment t, and return to Step 2, if $Selected\_Set_k(t) \neq \emptyset$.*

OUTPUT

- *Hyperplane $\boldsymbol{w}_k^*(t)$ that classifies the data at sensor k.*

Notice that the algorithm is applied at every node. After some gossiping has taken place, all nodes have the sufficient information to construct a plane close to the plane that would have been constructed if all sensors had access to the entire information. So, all sensors reach convergence near the optimal solution. Consensus in the network to the optimal discriminant is shown through simulations in the following Section.

5.1.6 Optimality of the ASG-SVM algorithm

The performance of the proposed algorithm can adapt depending on the selection of the threshold values. In the following Theorem we prove that optimality is achieved with the appropriate threshold values.

Theorem 5.1.3 *For appropriately chosen thresholds, the distributed ASG-SVM algorithm actually converges to the optimal classifier.*

Proof. In Section 5.1.4 we argued that for the minimum threshold values $F_{1min} = F_l^{(k)}$ and $F_{2min} = F_m^{(k)}$, the ASG-SVM algorithm coincides with the MSG-SVM algorithm, which converges to a suboptimal solution (Section 4.1.1). Therefore, the ASG-SVM algorithm exhibits the same performance for $F_{1min} = F_l^{(k)}$ and $F_{2min} = F_m^{(k)}$. The following Lemma shows that for greater threshold values *i.e.,* $F_1^{(k)} \geq F_l^{(k)}$ and $F_2^{(k)} \geq F_m^{(k)}$ the algorithm converges to the optimal solution.

Lemma 5.1.4 *The objective value of the SVM training subproblem at each sensor monotonically increases when sensors receive data in each iteration from neighboring sensors.*

Proof. See reference [61]. ■

This simply implies that when more sample vectors are added (and therefore the value of the thresholds increase), the optimal objective value cannot decrease. Notice that for $F_1^{(k)} = F_{1max}$ and $F_2^{(k)} = F_{2max}$ the ASG-SVM coincides with the centralized case where sensors exchange all their data and hence optimality is guaranteed. Therefore, on the interval $[F_l^{(k)}, F_{1max}]$ and $[F_m^{(k)}, F_{2max}]$ the ASG-SVM will finally converge to the optimal solution. ■

5.2 Results

In this section we present the performance of the proposed algorithm in terms of classification error rate. We test our algorithm for different threshold values and compare it with the distributed approach in [61] and the centralized case, where the entire data set is available. Through simulation experiments we show that for different threshold values, our algorithm actually converges to the optimal classifier. We also investigate the effect of the network topology, in terms of connectivity among sensors, on the time of convergence. In our experiments, we use a variety of data sets following multidimensional Gaussian distributions and test the various algorithms under several network topologies.

5.2.1 Performance

We consider a WSN with a grid topology, where each sensor collects $\{\mathbf{x}_i\}_{i=1}^{40}$, $\{\mathbf{y}_j\}_{j=1}^{40}$ sample vectors from two classes, at each time slot. WSN nodes communicate synchronously with their one-hop neighbors every time slot. In our experiments, we generated the data set for the general case of two linearly inseparable classes using multi-dimensional Gaussian distributions with two different means.

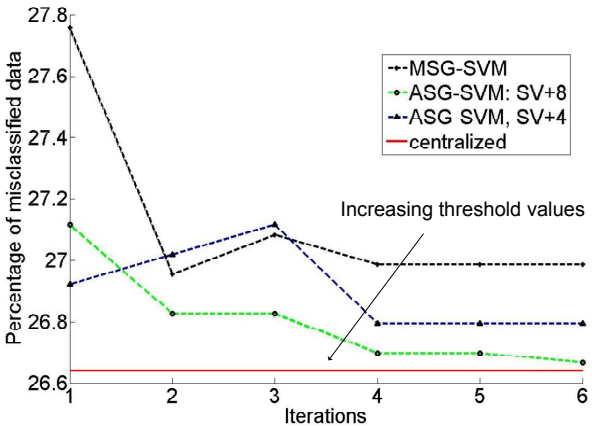


Figure 5.4: Performance, at a given particular sensor, of the distributed training algorithm. MSG-SVM is a suboptimal solution, hence it does not converge to the optimal hyperplane. The ideal case where sensors have access to the entire data is depicted by the straight line. ASG-SVM gives an optimal estimate of the discriminant when sensor exchange the support vectors and 8 more vectors. The classification error rate decreases as the threshold value increases.

We simulated 100 Monte Carlo runs in order to compare the performance of the pro-

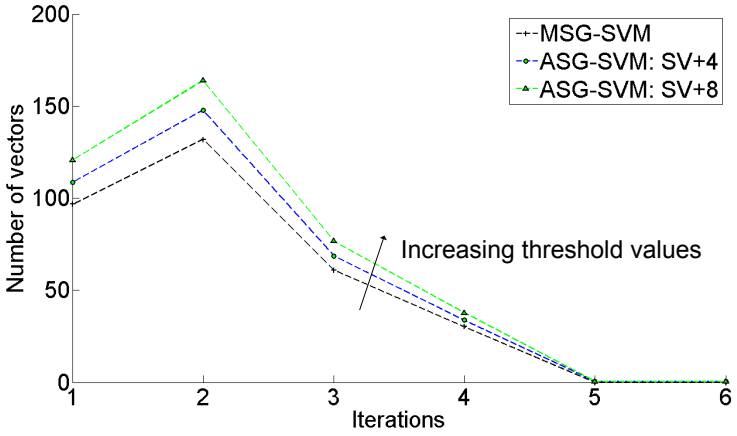


Figure 5.5: MSG-SVM achieves a sub-optimal solution using less measurements than the proposed algorithm. Using ASG-SVM, sensors exchange more measurements, hence they reach convergence closer to the optimal solution. The more measurements they exchange, the less the classification error rate is at convergence.

posed ASG-SVM algorithm with MSG-SVM [68] and a centralized SVM training scheme where sensors send all their data to a fusion center. Figure 5.4 illustrates the average classification error rates (%) for a randomly chosen sensor, as a function of the iteration steps. The straight line depicts the classification error rate for the centralized case. Since the data set contains linearly nonseparable measurements, the achieved classification error rate is expected not to be equal to zero. The black dotted curve (with the stars) depicts the performance of the MSG-SVM distributed algorithm [68], where neighboring sensors exchange only their support vectors. MSG-SVM is a suboptimal algorithm, as we argued in Section 4.1.1, hence it exhibits worse performance than the proposed algorithm. The other two dotted curves illustrate the performance of the proposed ASG-SVM distributed algorithm for different threshold values. As the threshold increases, the performance of the ASG-SVM improves, since the amount of information exchange among neighboring sensors increases. Notice that all sensors exhibit similar behavior in terms of classification error rates for the three algorithms. This is illustrated in the following Section, were we provide simulation results for the mean and the variance error of the estimates of the variables that determine the classifier.

To quantify the amount of data exchange required, Figure 5.5 illustrates the number of measurements that a particular sensor transmits to its neighbors at each iteration. At time slot $t = 0$ each node has collected $|S_k(0)| = 80$ vectors and at time slot $t = 1$ they start communicating. As expected, MSG-SVM transmits less measurements, (only the support vectors) than the ASG-SVM, thus consumes less power. ASG-SVM, on the other hand, transmits more data than the MSG-SVM at each iteration, in order to achieve more accurate training. Comparing Figures 5.4 and 5.5 , one can notice that the more measurements are exchanged, the lower the classification error rate per iteration and finally at convergence.

5.2.2 Comparison of ASG-SVM with DPSVM

In this Section we compare the proposed algorithm with the approach proposed in [61]. Independently of our work, Vandenberghe et al. proposed a distributed parallel SVM

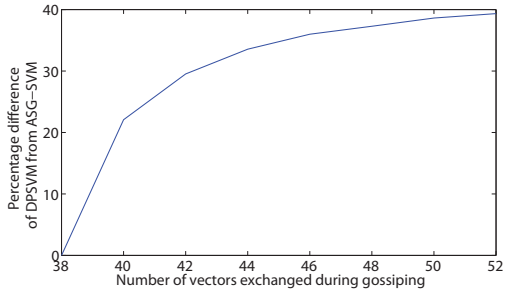


Figure 5.6: Percentage difference of achieved cost function values between DPSVM and ASG-SVM at convergence. The more measurements are included in the training set, the better solution achieved by the ASG-SVM.

training mechanism (DPSVM) based on the same idea of exchanging support vectors among multiple servers in a strongly connected network.

In these simulation experiments we use the WSN scenario described in Section 5.2.1. We run 100 Monte Carlo runs in order to calculate the cost function value for a randomly selected sensor for both ASG-SVM and DPSVM algorithms. C_{DP} denotes the cost function value (cf. (2.60)) achieved by the DPSVM algorithm at convergence, *i.e.,* at the final iteration step. We then calculate the same cost function values achieved by the ASG-SVM algorithm for different selection function threshold values. C_i depicts the cost function value for the ASG-SVM with threshold values F_{l+i} and F_{m+i}, *i.e.,* when neighboring sensors exchange the support vectors and i extra measurements from each class. Now, we calculate the percentage difference of cost function values between DPSVM and ASG-SVM at convergence as

$$\Delta C = \frac{C_{DP} - C_i}{C_{DP}} \ \%. \tag{5.11}$$

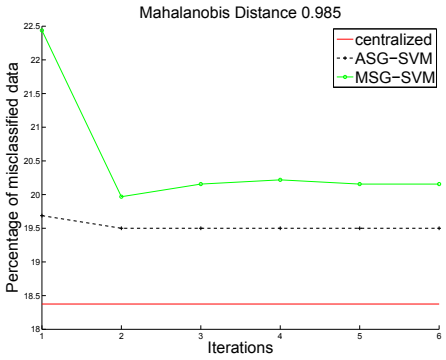
The percentage difference increases as the difference of the achieved cost function values between DPSVM and ASG-SVM increases. Figure 5.6 depicts the percentage difference of the DPSVM from the proposed algorithm at convergence, for a randomly selected node. The experiment begins for $i = 0$ and $m + n = 38$, *i.e.,* only the support vectors are exchanged. Apparently for $i = 0$ it holds that $C_{DP} = C_0$, since sensors exchange only the support vectors in both algorithms. On the other hand, when sensors exchange the support vectors and a few extra vectors during gossiping, ASG-SVM achieves a better solution, *i.e.,* a lower cost function value for the optimization problem (2.60) than DPSVM. Consequently, this simulation experiment shows that the ASG-SVM algorithm goes down the error performance surface by 40% more than what DPSVM does, by exchanging 8 (or 20%) more vectors than DPSVM.

5.2.3 Effect of the distribution of the measurements

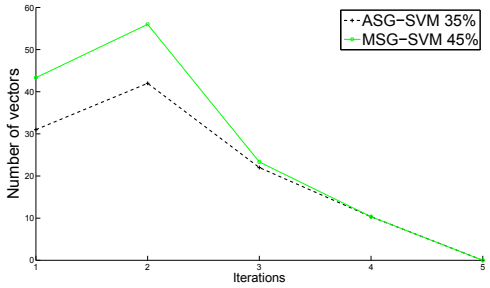
We test the performance of the algorithms in terms of the classification error rate for several data sets. We generate three sample data sets of two different classes each, using 4-dimensional Gaussian distributions with two different means, such that their Mahalanobis distance is increasing. This simply implies that the first data set contains measurements from two nonlinearly separable distributions, the second one, data with less correlated data and the third data set contains measurements from two linearly separable distributions.

We simulated 100 Monte-carlo runs for the network topology, in all three data sets. In all cases, the MSG-SVM algorithm does not converge to the optimal solution. Clearly, for a

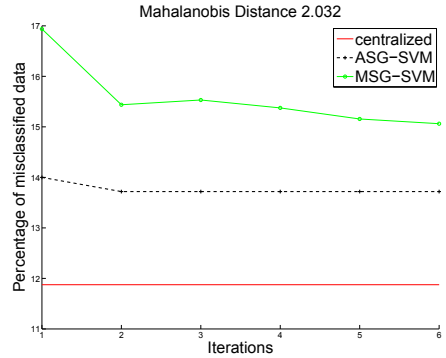
randomly chosen sensor in the network, exchanging only the support vectors of each sensor which corresponds to the 20% of the measurements of the sensor, leads to a suboptimal solution as the classification error rate of the MSG-SVM is 18% while the classification error rate of the centralized case is only 4%, (cf. Fig. 5.7(a), (c) and (e)). On the other hand, the ASG-SVM algorithm with 35% of the measurements exchanged at each step, exhibits better performance than the MSG-SVM in all cases.



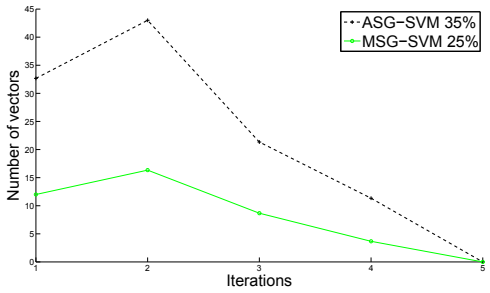
(a)



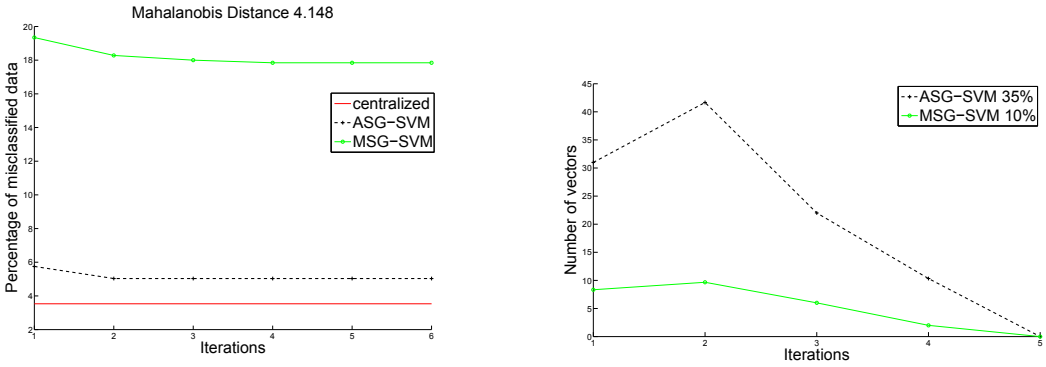
(b)



(c)



(d)



(e) (f)

Figure 5.7: Three data set of two classes with Mahalanobis distance. On the left, the performance of ASG-SVM algorithm is depicted compared to the MSG-SVM and the centralized case, for a randomly selected sensor in the network. The number of the measurements exchanged at each iteration is depicted on the right.

Figures 5.7 (b), (d) and (f) illustrate the number of vectors that are exchanged in a randomly selected sensor in the network. One can notice in Figure 5.7 (b) (which corresponds to data set 1), that the MSG-SVM transmits a larger number of vectors than that of the ASG-SVM algorithm. This is because the misclassified data are also support vectors. Hence, the number of the support vectors in such data sets is large when the two classes are highly correlated. This also explains Figure 5.7 (b) where the MSG-SVM and ASG-SVM exhibit similar performance. For distributions with larger Mahalanobis distance and thus less misclassified data, the MSG-SVM communicates less data than ASG-SVM.

5.2.4 Effect of network topology

Size and connectivity of the network have an impact on the performance of the distributed algorithm. These two parameters determine the number of the nodes that may work concurrently and the frequency at which a node receives training data from neighboring nodes.

In distributed SVM applications, one can notice that upon convergence, each node must at least contain the hole set of the support vectors, *i.e.,* the support vectors after a centralized training with the entire data set. Therefore, it is not very meaningful to have more than $\frac{n+m}{N_{SV}}$ nodes, where $(n+m)$ are the number of measurements for both classes respectively, and N_{SV} denotes the number of support vectors for the centralized training. In this section we illustrate through simulation experiments the convergence of the distributed algorithm ASG-SVM to the global optimal classifier for grid topology and random sparse topology networks, in terms of connectivity among sensors.

Figure 5.8 depicts the consensus achieved by all sensors to the optimal hyperplane in a grid topology. We measure the mean of the error of the estimates for the variables that define the hyperplane, *i.e.,* two components of vectors $\mathbf{w}$ and parameter b at convergence. The vertical lines indicate the error variance of the variables that define the hyperplane. It is clear that as the threshold for the selection function increases, *i.e.,* the amount of information that neighboring sensors exchange increases, the estimation error decreases. The mean and the variance estimates for the variables $\mathbf{w}$ and b upon convergence tend to

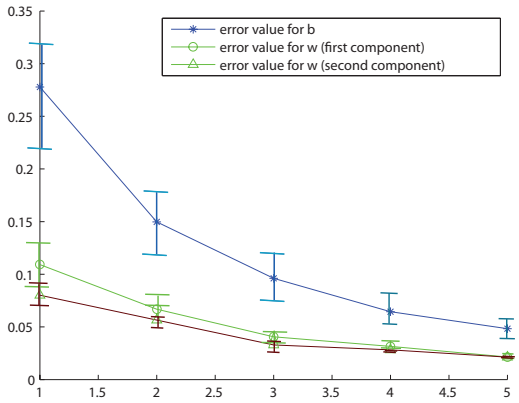


Figure 5.8: The mean and variance error of the estimates of parameters $\mathbf{w}$ and b. As the threshold increases, both the mean and the variance error tends to zero.

zero, hence all sensors converge to the global optimal solution.

(a) (b)

Figure 5.9: Different network topologies in terms of connectivity among sensors a) Sparse network topology b) Dense network topology.

In the following, we investigate how the connectivity of the network topology affects the convergence of the proposed distributed algorithm. Consider two random network topologies with random sparse connectivity and random dense connectivity, as in Figures 5.9 (a) and (b) respectively. The network topology depicted in Figure 5.9 (a) is constructed from a grid topology of $N = 25$ nodes, where the node activity is 60% of the total number of nodes (sparse connectivity). The network topology depicted in Figure 5.9 (b) is constructed from the same grid topology, where the node activity is 90% of the total number of nodes (dense connectivity). We estimate at each time step of the algorithm the variables $\mathbf{w}$ and b for the different topologies. We expect that the convergence to the optimal classifier is faster for the dense topologies. This is illustrated in Figure 5.10. Again the error estimates of the mean and the variance (vertical lines) of the variables that determine the classifier tend to zero for all connectivities, hence all sensors converge to the global classifier.

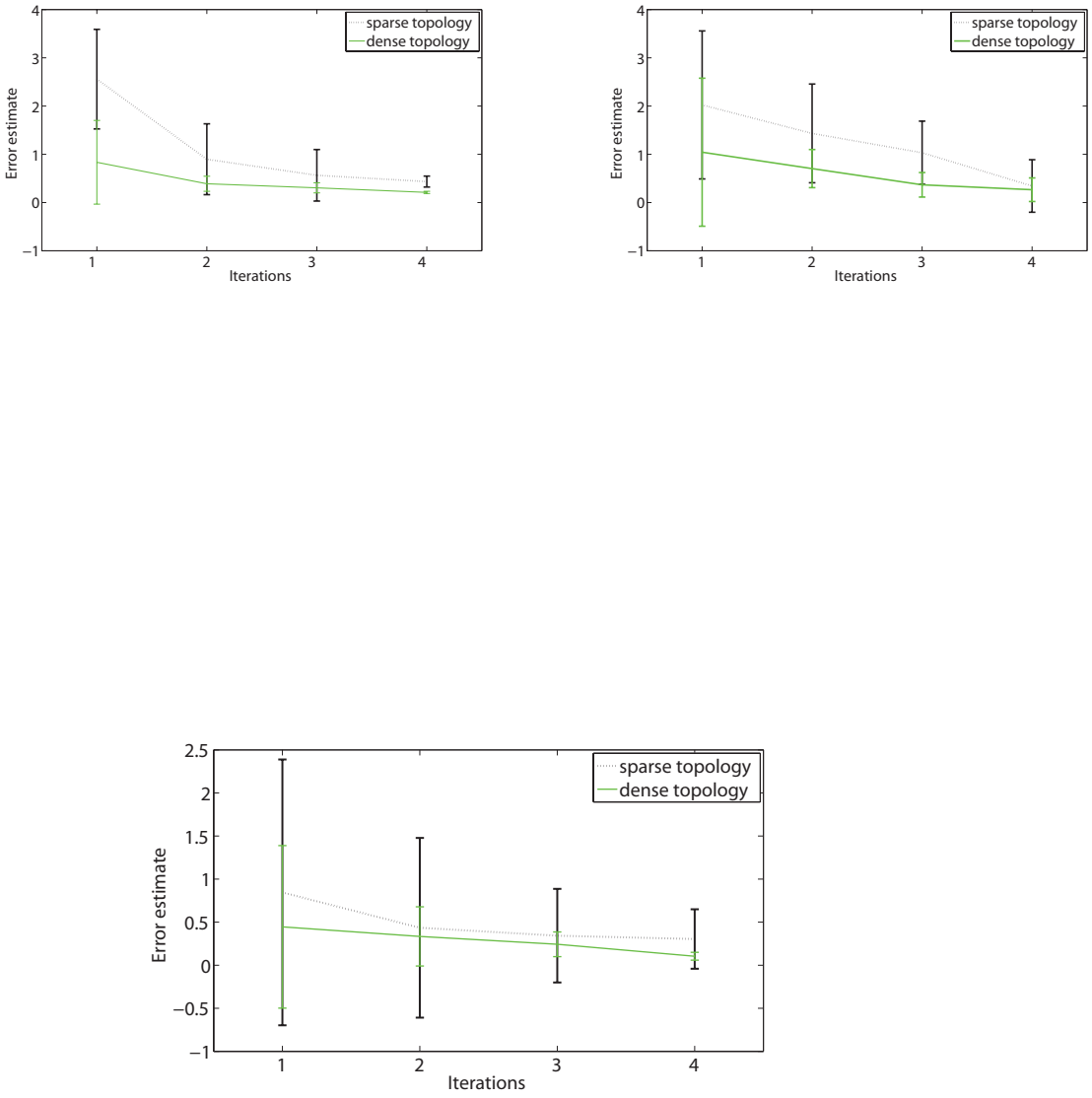


Figure 5.10: The mean and variance error estimate of parameters $\mathbf{w}$ and b at each time step tends to zero. Denser topologies exhibit faster convergence.

5.3 Conclusions

In this Chapter, we derived a novel mathematical characterization for the sparse representation of the most important measurements that neighboring sensors should exchange in order to reach an agreement to the optimal linear classifier. The proposed selection function is associated with the distance from the hyperplane. This function ranks the training vectors in order of importance in the learning process. The amount of information to be exchanged is controlled by a user defined threshold, depending on the desired trade-off between classification accuracy and power consumption. The more measurements communicated between neighboring sensors, the lower the classification error rate.

The motivation for the development of the proposed algorithm was the need for optimality during the distributed training of a SVM. More specifically, since the most obvious procedure of transmitting only the support vectors leads to a sub-optimal solution, the sufficient amount of data for convergence to the optimal classifier needs to be determined. We proposed a systematic way to optimally select partial information during gossiping so that the network reaches convergence to the optimal solution. We proved that a threshold value exists for which the proposed algorithm converges to the optimal classifier. We explored the effect of the network topology to the time convergence. Finally, we compared the proposed algorithm with the DPSVM approach [61]. This simulation experiment showed that the ASG-SVM algorithm goes down the error performance surface by 40% more than what DPSVM does, by exchanging or 20% more vectors than DPSVM.

Chapter 6

Conclusions

The purpose of this thesis was to answer a crucial question in WSN application: what kind of information should sensors communicate in order to achieve the desired trade-off between computation accuracy and energy consumption. This thesis comprised a study of distributed optimization techniques for in-network data processing so as to eliminate the need to transmit raw data to a central point. We focused on a very interesting classification tool (SVM) and studied it as a quadratic problem that involves optimization of a convex cost function. In order to apply the SVM training in a WSN application one needs to perform the training in a distributed fashion. This is exactly our contribution: the design of distributed algorithms for SVM training in the context of a WSN.

Taking advantage of the sparse representation that SVMs provide for the decision boundaries, we presented classes of incremental and gossip-based distributed consensus algorithms for training the classifier. In all incremental algorithms, the update of the estimate is diffused sequentially in the network. In our proposed algorithms (DFP-SVM, WDFP-SVM), we proved through simulation experiments that after only one pass through all nodes, the network converges to the optimal estimate of the classifier, at the final step. Therefore, only one node, at the final incremental step, has the updated information and therefore the optimal estimate. This is not ideal for a WSN application because WSNs usually operate in environments that are prone to link or node failures. Therefore, we used gossip-based algorithms, that are robust to unexpected failures of nodes and consequently to changes in the topology. We proposed the MSG-SVM algorithm that gives a suboptimal solution, but consumes less possible energy, and the SSG-SVM algorithm that guarantees convergence to the optimal solution but consumes more energy since more data is communicated among neighboring sensors.

The key question we answered in this thesis, is what kind of data should sensors exchange in order to reach convergence to the optimal solution given a desired trade-off between energy consumption and classification accuracy. We derived a rigorous mathematical characterization of the importance of a vector sample as to be selected for exchanging it with other neighboring sensors. We provided a selection function which ranks the training vectors in order of importance in the learning process of the SVM linear classifier. Through simulation experiments, we showed that even though the proposed algorithm uses partial information for inter-node communication, all sensors converge to the same hyperplane obtained using a centralized SVM classifier that employs the entire sensor data at a fusion center. Tuning appropriately the threshold, the network can converge to the optimal solution, or to an estimate close to the optimal solution. The classification error rate decreases as the threshold value increases, as expected.

We finally investigated the parameters that influence the convergence of this approach. First, we measured the mean of the error of the estimates for the variables that define the hyperplane. It was clear that as the threshold for the selection function increases, i.e., the amount of information that neighboring sensors exchange increases, the estimation error decreases. Hence all sensors converge to the global optimal solution. The size and

connectivity of the network have an impact on the performance of the distributed algorithm. Simulation experiments illustrated that the convergence to the optimal classifier is faster for dense topologies and slower for sparse topologies. This work resulted in several publications:

- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Optimal Gossip Algorithm for Distributed Consensus SVM Training in Wireless Sensor Networks," in Proc. 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, July 5-7, 2009.
- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Distributed Consensus Algorithms for SVM training in Wireless Sensor Networks," in Proc. 16th European Signal Processing Conference (EUSIPCO '08), Lausanne, Switzerland August 25-29, 2008.
- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Selective Gossiping for SVM training in Wireless Sensor Networks," in Proc. 5th European Conference on Wireless Sensor Networks (EWSN '08), Bologna, Italy, January 30-1, 2008.
- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Training a SVM-based Classifier in Distributed Sensor Networks," in Proc. 14nd European Signal Processing Conference (EUSIPCO '06), Florence, Italy, September 4-8, 2006.
- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Energy-Efficient Distributed Support Vector Machines for Wireless Sensor Networks," in Proc. 2006 European Workshop on Wireless Sensor Networks (EWSN '06), Zurich, Switzerland, February 13-15, 2006.
- K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Adaptive consensus SVM training in wireless sensor networks with power-aware gossip algorithms," to be submitted to the IEEE Trans. on Signal Processing.

Chapter 7

Future Work Directions

In this work we focused on the specific problem of distributed SVM training in WSNs. The variety of the applications and the nature of the optimization problem of SVM, trigger many research questions in this area. In the following, we present some future work directions.

In many applications of machine learning, abundant amounts of data can be cheaply and automatically collected. However, manual labelling for the purposes of training learning algorithms is often a slow, expensive, and error-prone process. Due to its wide applicability, the problem of semi-supervised classification is attracting increasing attention in machine learning. Semi-Supervised Support Vector Machines are based on applying the margin maximization principle to both labelled and unlabelled examples. Unlike SVMs, where all measurements are labelled, their formulation leads to a non-convex optimization problem. Therefore, one can address this problem by solving the standard SVM problem while treating the unknown labels as additional optimization variables [75, 76]. Our approach could be applied in a distributed semi-supervised SVM in combination with the already existing algorithms. It would be very interesting to investigate the convergence of the algorithm for different network topologies and how convergence is influenced by the number of labelled and unlabelled data.

Furthermore, one can also study the case of *active learning* of a SVM. Active learning is the procedure during which the learner can improve the classifier by actively choosing the optimal data from the potential training data set and adding it into the current labelled training set, after getting its label during the processes, [77, 78]. The key point of active learning is again the sample selection criteria. Since the optimization problem is the same, then a modified version of the presented selection function could rank the data in order of importance so as to optimize the classification problem. Another research question arising in active learning that can be included in this scenario is to try to take into account the redundancy among examples. There is no need to select multiple examples in the training procedure that are similar (or even identical) to each other.

Networks of interconnected devices with storage and processing capabilities are widespread: internet, intranets, computing grids, sensor networks, etc. In the internet for example one can find an increasing number of databases (such as weather, oceanographic, remote sensing, financial, etc.) becoming on line and distributed. Similar examples can easily be found in other networks. Distributed scenarios naturally emerge when data are captured in many places and their transport and storage to a unique location in infeasible or suboptimal. Consider examples like image or video where images can even reach the terabyte range (such as astronomy telescope images), high dimensional data, such as documents. In the problem studied in this work and especially for high dimensional data and classes that are not linearly separable, even the number of support vectors might increase rapidly during the training procedure. Therefore, other techniques could also be applied for further reduction of the measurements to be selected for transmission [62].

Finally, the optimization techniques presented in this work can be also applied in other cases where distributed optimization needs to take place. Many engineering tasks can be

expressed as optimization problems of a certain cost function. Therefore, it would be very interesting to follow relative strategies and optimization techniques for similar distributed optimization problems.

Bibliography

[1] Lin Xiao and Stephen Boyd, "Fast Linear Iterations for Distributed Averaging," in *Proc. 42th Conf. on Decision and Control*, Hawaii, USA, December 2003, pp. 4997–5002. 1, 17, 29, 30, 31, 63

[2] Hans-Joachim Hof, "Applications of sensor networks," in *Algorithms for Sensor and Ad Hoc Networks*, 2007, pp. 1–20. 7

[3] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. Wireless sensor networks for habitat monitoring WSNA'02*, 2002. 7

[4] ," TinyOS community forum: `http://www.tinyos.net/`. 7

[5] K. Martinez, P. Padhy, A. Riddoch, R. Ong, and J. Hart, "Glacial environment monitoring using sensor networks," in *Proc. of Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, 2005. 8

[6] T. He, S. Krishnamurthy, J.A Stankovic, T.F Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh, "An energy-efficient surveillance system using wireless sensor networks„" in *Proc. Second International Conference on Mobile Systems, Applications, and Services (MobiSys),*, 2004. 9

[7] K. Lorincz, D.J. Malan, T.R.F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, and M. Welsh, "Sensor networks for emergency response: Challenges and opportunities," *IEEE Pervasive Computing*, 2004. 10

[8] D. Ganesan, D. Estrin, and J. Heidemann, "Dimensions: Why do we need a new data handling architecture for sensor networks," in *ACM SIGCOMM Computer Communication Review*, 2003, pp. 143–148. 11

[9] T. Joachims, "Text categorization with support vector machines," in *Proc. 10th Eur. Conf. on Machine Learning (ECML'98)*, Chemnitz, Germany, April 1998, pp. 137–142. 13

[10] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. of the 1997 Conference on Comp. Vision and Pattern Recogn.*, Washington, DC, USA, 1997, p. 130, IEEE Computer Society. 13

[11] A. Bulut, P. Shin, and L. Yan, "Real-time nondestructive structural health monitoring using support vector machines and wavelets," in *Proc. of the Conf. on Advanced Sensor Technologies for Nondestructive Evaluation and Structural Health Monitoring (NDE'05)*, San Diego, CA, USA, March 2005. 13

[12] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines," in *Proc. Int. Conf. on Machine Learning*, Stanford, CA, USA, 2000, pp. 487–494. 14, 55

[13] M. Rabbat and R. Nowak, "Distributed Optimization in Sensor Networks," in *Information Processing in Sensor Networks (IPSN '04)*, Berkeley, CA, USA, April 2004. 17, 18

[14] A. Olshevsky and J. N. Tsitsiklis, "Convergence Speed in Distributed Consensus and Averaging," in *45th IEEE Conf. on Desicion and Control*, San Diego, CA, USA, December 14-20, 2006. 17

[15] L. Xiao and S. Boyd, "Optimal Scaling of a gradient method for distributed resource allocation," *Optimization Theory and Applications*, vol. 129, (3), pp. 469–488, 2006. 17

[16] A. Razavi and Zhi-Quan Luo, "Distributed Optimization in an Energy-Constrained Network," in *Acoustics, Speech and Signal Processing, ICASSP'07*, Honolulu, Hawaii, USA, April 14-20, 2007. 17

[17] M. G. Rabbat, R. D. Nowak, and J. A. Bucklew, "Generalized Consensus Computation in Networked Systems With Erasure Links„" in *Proc. of the 6th Workshop on Sig. Processing Advances inWireless Communications*, New York, NY, June 5-8 2005, pp. 1088–1092. 18

[18] V. Delouille, R. Neelamani, and R. Baraniuk, "Robust Distributed Estimation in Sensor Networks using the Embedded Polygons Algorithm„" in *Proc. of the 3rd Intl. Symp. on Info.*, Berkeley, CA„ April 2004. 18

[19] D. P. Spanos, R. O. Saber, and R. M. Murray, "Distributed Sensor Fusion Using Dynamic Consensus," in *Proc. of the 16th IFAC World Congress*, Prague„ July 2005. 18

[20] P. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed Expectation-Maximization algorithm for density estimation and classification using wireless sensor networks," in *Proc. of Conf. on Acoustics, Speech and Signal Processing*, Las Vegas, Nevada, March 30-April 4 2008. 18

[21] S. Sundaram and C. N Hadjicostis, "Distributed Function Calculation and Consensus using Linear Iterative Strategies," *Selected areas in Communication*, vol. 26, (4), pp. 650–660, May 2008. 18

[22] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T.S. Furey, M. Ares, and D. Haussler, "Knowledge-based Analysis of Microarray Gene Expression Data Using Support Vector Machines," in *Proceedings of the National Academy of Sciences*, 2000, pp. 97(1), 262–267. 18

[23] H. Drucker, D. Wu, and V. N. Vapnik, "Support Vector Machines for Spam Categorization," in *IEEE Trans. on Neural Networks*, 1999, vol. 10, pp. 1048–1054. 18

[24] Christopher J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998. 18, 44

[25] C. E. Perkins, *Ad Hoc Networking*, Addison-Wesley, Upper Saddle River, NJ, 2001. 19

[26] I. Stojmenovic, *Handbook of Wireless Networks and Mobile Computing*, Wiley, 2002. 19

[27] C. K. Toh, *Ad Hoc Mobile Wireless Networks*, Prentice Hall PTR, Upper Saddle River, NJ, 2002. 19

[28] G. T. Huang, "Casting the wireless sensor net," *Technology Review*, pp. 51–56, 2003. 21

[29] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, and W. J. Kaiser, "Wireless integrated network sensors: Low power systems on a chip," in *Proceedings of the of the 1998 European Solid State Circuits Conference*, The Hague, Netherlands, 1998. 21

[30] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002. 22

[31] D. Estrin and R. Govindan, "Next century challengesnext century challenges: scalable coordination in sensor networks," in *MobiCom*, Seattle, WA, August 1999. 22

[32] Holger Karl and Andreas Willig, *Protocols and architectures for Wireless Sensor Networks*, John Wiley and Sons Ltd, 2005. 25

[33] Deepankar Medhi, *Network Routing: Algorithms, Protocols, and Architectures*, The Morgan Kaufmann Series in Networking, 2007. 25

[34] Azzedine Boukerche, *Algorithms and Protocols for Wireless, Mobile Ad Hoc Networks*, Wiley Series on Parallel and Distributed Computing, 2009. 25

[35] Kemal Akkaya and Mohamed Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, pp. 325–349, 2005. 25

[36] S. Hedetniemi and A. Liestman, "A survey of gossiping and broadcasting in communication networks," in *Networks*, 1998. 27

[37] A. Ghosh S. Boyd, B. Prabhakar, and D. Shah, "Gossip Algorithms: Design, Analysis and Applications," in *INFOCOM, 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Miami, USA, March 2005, pp. 1653–1664. 27, 63

[38] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS'03*, Washington, DC, USA, 2003. 27

[39] D. Spanos, R. Olfati-Saber, and R. Murray, "Distributed kalman filtering in sensor networks with quantifiable performance," in *Fourth International Symposium on Information Processing in Sensor Networks*, 2005. 28

[40] L. Xiao, S. Boyd, and S. Lall, "A scheme for asynchronous distributed sensor fusion based on average consensus," in *Fourth International Symposium on Information Processing in Sensor Networks*, 2005. 28

[41] V. Saligrama, M. Alanyali, and O. Savas, "Distributed detection in sensor networks with packet losses and finite capacity links," in *Trans. on Signal Processing*, 2006, pp. 4118–4132. 28

[42] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *Trans. on Information Theory*, pp. 2508–2530, June 2006. 28

[43] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambirdge, UK: Cambridge University Press, 2004. 35, 36, 45, 76

[44] D. P Bertsekas, *Nonlinear Programming*, Belmont, MA: Athena Scientific, 2nd edition, 1999. 36

[45] A. Nedic and D.P. Bertsekas, "Incremental Subgradient Methods for Nondifferentiable Optimization," Tech. Rep. LIDS-P-2460, MIT, Cambridge, 1999. 37

[46] S Resnick, "Convergence rate of incremental subgradient algorithms," in *Stochastic Optimization: Algorithms and Applications*, S. Uryasev and P.M. Pardalos, Eds., pp. 263–304. Kluwer, 2000. 37, 38, 41

[47] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Information Processing in Sensor Networks (IPSN '04)*, Berkeley, CA, USA, April 2004. 37, 39

[48] Dimitri P. Bertsekas, *Nonlinear Programming*, Athena Scientific, 1999. 37, 38

[49] N.Z. Shor, *Minimization Methods for Non-differentiable Functions*, Springer Series in Computational Mathematics, 1985. 38

[50] V. N. Vapnik, *The nature of statistical learning theory*, New York: Wiley, 1996. 44, 45

[51] T. Evgeniou, M. Pontil, and T. Poggio, "Regularization networks and support vector machines, advances in large margin classifiers," *Cambridge, MA, MIT Press*, pp. 171–203, 2000. 44

[52] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer-Verlag, NY, 1999. 44

[53] B.E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. of the fifth annual workshop on Computational learning theory,*, Pennsylvania, US, 1992, pp. 144–152. 44

[54] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," in *Machine Learning*, 1995, pp. 273–297. 45

[55] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 3nd edition, 2006. 48, 49

[56] S. Ruping, "Incremental learning with support vector machines," in *Proc. IEEE Int. Conf. on Data Mining*, San Jose, CA, USA, November 2001, pp. 641–642. 51, 53, 56

[57] N. Syed, H. Liu, and K. Sung, "Incremental learning with support vector machines," in *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence, IJCAI'99*, July 1999. 51, 53, 54

[58] C. Domeniconi and D. Gunopoulos, "Incremental support vector machine construction," in *IEEE Int. Conf. on Data Mining, ICDM'01*, San Jose, CA, USA, November 2001. 51, 53, 54

[59] C. P. Diehl and G. Cauwenberghs, "Support vector machine incremental learning, adaptation and optimization," in *Proc. Int. Joint Conf. on Neural Networks*, Portland, OR, July 2003. 51, 53

[60] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Proc. IEEE Workshop on Neural Networks and Signal Processing, NNSP'97*, Amelia Island, FL, September 1997, pp. 276–285. 51, 57

[61] Y. Lu, V. Roychowdhury, and L. Vandenberghe, "Distributed Parallel Support Vector Machines in Strongly Connected Networks," *IEEE Transactions on Neural Networks*, vol. 19,7, pp. 1167–1178, 2008. 51, 74, 83, 84, 90

[62] E. Parrado-Hernandez A. Navia-Vazquez, D. Gutierrez-Gonzalez and J.J Navarro-Abellan, "Distributed Support Vector Machines," *IEEE Transactions on Neural Networks*, vol. 17,4, pp. 1091–1097, 2006. 51, 93

[63] E. Kokiopoulou and P. Frossard, "Distributed SVM applied to image classification," *IEEE Int. Conf. on Multimedia and Expo*, pp. 1753–1756, 2008. 51

[64] A. Wang and A. Chandrakasan, "Energy Efficient DSPs for Wireless Sensor Networks," *IEEE Signal Proc. Magazine*, 2002. 54

[65] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Training a SupportVector Machine-based Classifier in Distributed Sensor Networks," in *14nd European Signal Processing Conference (EUSIPCO '06)*, Florence, Italy, September 4-8, 2006. 54, 64

[66] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Energy-Efficient Distributed SupportVector Machines for Wireless Sensor Networks," in *Proc. 2006 European Workshop on Wireless Sensor Networks (EWSN '06)*, Zurich, Switzerland, February 13-15, 2006. 54

[67] A. Ghosh S. Boyd, B. Prabhakar, and D. Shah, "Randomized Gossip Algorithm," *IEEE/ACM Transactions on Networking*, vol. 52,6, pp. 2508–2530, 2006. 63

[68] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Distributed Consensus Algorithms For SVM Training in Wireless Sensor Networks," in *Proc. 16th European Signal Processing Conf.*, Lausanne, Switzerland, August 25-29, 2008. 63, 84

[69] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Selective gossiping for svm training in wireless sensor networks," in *Proc. 5th European Conference on Wireless Sensor Networks (EWSN '08)*, Bologna, Italy, January 30-1, 2008. 63

[70] K. Bennett and C. Campbell, "Support vector machines: hype or hallelujah," *SIGKDD Explorations*, vol. 2,2, pp. 1–13, 2000. 64, 66

[71] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Optimal gossip algorithm for distributed consensus svm training in wireless sensor networks," in *Proc. 16th International Conference on Digital Signal Processing(DSP'09)*, Santorini, Greece, July 5-7, 2009. 73

[72] D. Caragea, A. Silvescu, and V. Honavar, "Agents that Learn from Distributed Dynamic Data Sources„" in *Proc. of the Workshop on Learning Agents, Agents 2000/ECML*, Barcelona, Spain, June 2000, pp. 53–61. 74

[73] Huan Liu Nadeem Ahmed Syed and Kah Kay Sung, "Incremental Learning with Support Vector Machines„" in *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI),*, Stockholm, Sweden, July 31 - August 6 1999. 74

[74] P. Gruber and J. Wills, *Handbook of Convex Geometry*, Elsevier Science Publishers B.V., 1993. 74, 75

[75] O. Chapelle, V. Sindhwani, and Sathiya S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 203–233, 2008. 93

[76] Kristin P. Bennett and A. Demiriz, "Semi-supervised support vector machines," *Advances in Neural Information Processing Systems*, vol. 12, pp. 368–374, 1998. 93

[77] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research*, vol. 2, pp. 45–66, 2001. 93

[78] Jun Jiang and Horace Ho-Shing Ip, "Dynamic distance-based active learning with svm," in *MLDM*, 2007, pp. 296–309. 93